Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO DE FIN DE GRADO

Grado en Matemáticas

Redes Neuronales Convolucionales. Aplicación a la detección de objetos.

Autor: Sergio García Pajares

Tutor: Eustasio del Barrio Tellado

Año 2024

Trabajo de Fin de Grado, Universidad de Valladolid

Depositado el día: 24 de junio de 2024

ABSTRACT

Convolutional Neural Networks is a family of parametric functions that has proven useful in many applications such as computer vision. Their theoretical grounds are based on statistics, but they rely on numerical analysis to find the right parameters to perform on a given task. We explain the general foundations of machine learning. Afterwards, we explain how convolutional neural networks are built and the main algorithms for their numerical optimization. Finally, we explain in detail YOLO (You Only Look Once) algorithm, which is able to identify objects in images and generate bounding boxes for them in real time.

Keywords: Object Detection, YOLO, Computer Vision, Convolutional Neural Networks.

RESUMEN

Las redes neuronales convolucionales son una familia paramétrica de funciones que han demostrado ser útiles en gran variedad de aplicaciones como la visión artificial. Sus fundamentos teóricos están basados en la estadística, pero también dependen del análisis numérico para encontrar los parámetros adecuados para resolver una tarea concreta. Explicamos los fundamentos del aprendizaje automático. Después, explicamos cómo se construyen las redes neuronales convolucionales y los principales algoritmos para su optimización numérica. Finalmente, explicamos en detalle el funcionamiento del algoritmo YOLO (mira una única vez), que es capaz de identificar objetos en imagen y generar rectángulos delimitadores para ellos en tiempo real.

Palabras clave: Detección de objetos, YOLO, Visión Artificial, Redes Neuronales Convolucionales.

Any sufficiently advanced technology
is indistinguishable from magic.

Arthur C. Clarke

AGRADECIMIENTOS

Pido perdón a los lectores de este trabajo por entregar una memoria que, en mi humilde opinión, está incompleta. Me hubiese gustado haber podido tratar en este TFG otros temas que eran de mayor interés para mi como el seguimiento de objetos (*e.g.* con Deep SORT) o las métricas de rendimiento y sus problemas asociados. También, pido perdón por haber incluido teoremas y proposiciones sin proporcionar demostración alguna. Para describir la causa de semejante descalabro y sin intentar excusarme, me gustaría suscribir estas palabras del escritor Alfonso Reyes: *«Publicamos para no pasarnos la vida corrigiendo los borradores»*.

Además de al lector, me gustaría pedirme perdón a mi mismo por no sentir este trabajo como propio, pues gran parte de sus contenidos nunca los quise incluir. Este trabajo ha sido una pasarela de contratiempos e imprevistos desde que comencé a escribirlo en julio de 2023 con la intención de defenderlo en febrero de 2024. No me arrepiento, sin embargo, del tema escogido. Mi personalidad rebelde y desafiante siempre va a preferir un arriesgado salto al vacío con un trabajo de investigación que traspasa las fronteras de lo estudiado en el grado frente a un trabajo bibliográfico que persiga reproducir las demostraciones de uno o dos libros dentro de la zona de confort creada por las asignaturas del grado.

Lo principal, ante un trabajo de esta envergadura, es cerrar definitivamente mi etapa universitaria. Ha sido una etapa de fuertes altibajos que pueden nublar las aventuras vividas. Seis años en los que he tenido tiempo de pasar por siete pisos de estudiantes, vivir en cinco ciudades y estudiar en cuatro universidades de dos países distintos. También, he tenido tiempo para trabajar: mozo de almacén, peón de obra, camarero, webmaster, técnico de cronometraje o *speaker* deportivo son algunas de las cosas que he sido. Siempre recordaré con especial cariño mi etapa como profesor ayudante en la *University of Dundee*. Cuando empecé la carrera, mi objetivo era –ya no lo es– convertirme en profesor de universidad. Aunque, sin duda, alguno de los mejores momentos me los llevo de las clases de salsa, bachata y kizomba o de las excursiones del *Canoe club*.

Todo esto no hubiera sido posible sin determinadas personas clave que me han acompañado durante todo este tiempo. Han sido mi inspiración y mi cobijo. Por ello, quisiera dar un agradecimiento especial:

A mi hermana, Mária, por pelear por mi durante todos estos años. Te necesito más de lo que estoy dispuesto a admitir.

A mis padres, Celes y María Jesús, por enseñarme, cada uno a su manera, su forma de entender la vida.

A mi tía, María Cristina, por gritarme una y otra vez que hiciera los deberes de matemáticas cuando era un niño.

A mi abuelo, Jesús, que un día decidió que eramos suficientemente mayores para caminar solos -cuanto te equivocabas- y decidió irse. Me enseñó que ante la adversidad no cabe otra cosa sino una sonrisa.

A mis abuelos; Juana, Celestino y Juliana; por enseñarme el significado de la palabra *amor*.

A los profesores del Bachillerato Internacional del IES Jorge Manrique de Palencia y, en especial, a Miguel Ángel y Arturo; por haber sido mis mentores en una de las épocas más difíciles de mi vida. Sin vosotros, con total certeza, no sería quien soy.

Aún recuerdo el primer día que llegué a la Universidad de Oviedo, primera parada del trayecto, con una mochila de ilusión y esperanza y dispuesto comerme el mundo. Seis años después, ese día con el que tanto soñaba, ha llegado.

Índice general

Capítulo 1

INTRODUCCIÓN

La Inteligencia Artificial (*AI*, por sus siglas en inglés) está en boca de todos desde la llegada de grandes modelos de uso público capaces de realizar tareas cotidianas, *e.g.* ChatGPT nos da conversación, DALL·E 2 dibuja por nosotros y SORA genera increíbles vídeos. También se encuentra en nuestros dispositivos móviles, donde seguramente pase más desapercibido. Reconocimiento y sintetización de voz o detección de caras son algunas de tecnologías que dependen fuertemente de AI.

Pese al reciente boom, la AI nace en 1943 con el primer modelo matemático de redes neuronales, siendo el término *Inteligencia Artificial* acuñado oficialmente en 1956 (Haenlein *et al.*, 2019). No fue hasta 1980 que nace la Asociación Americana para Inteligencia Artificial (Buchanan, 2006). El primer gran hito llegó en 1997 cuando el equipo de Deep-Blue ganó al entonces campeón mundial de ajedrez Gary Kasparov (Buchanan, 2006). En 2015, comienza la revolución con la llegada de AlphaGo, desarrollada por Google, que ganó al campeón mundial del juego de mesa Go, unas 20 veces más complejo que el Ajedrez (Haenlein *et al.*, 2019).

En este capítulo, veremos los objetivos de este trabajo. Después, trataremos los fundamentos del aprendizaje automático y, dentro de este, el *aprendizaje supervisado*. Cerraremos el capítulo hablando de los problemas de visión artificial.

1.1 OBJETIVOS Y MOTIVACIÓN

Este trabajo nace de mis prácticas de empresa en la Fundación CIDAUT. La Fundación CIDAUT es una fundación de investigación privada y desarrollo en los sectores de transporte y energía. Tiene su sede en Boecillo (Valladolid) y cuenta con alrededor de doscientos empleados actualmente. Dentro de sus diversas líneas de investigación, destacan su amplia experiencia en la seguridad de los vehículos y la infraestructura vial, así como el automóvil autónomo y conectado.

Este trabajo está motivado por la actual y demandante carrera tecnológica hacia la conducción autónoma. Ya existen en el mercado vehículos que son capaces de aparcar autonomamente o mantener el vehículo dentro del carril. También hay lectores de señalización. La detección y posicionamiento de objetos en una imagen es una eslabón dentro

de una larga cadena que constituye la conducción autónoma. Para lograrlo, se emplean técnicas basadas en redes neuronales convolucionales.

Los objetivos de este trabajo son:

1. Entender y explicar las nociones elementales del aprendizaje automático.
2. Entender y explicar qué son las redes neuronales y, más concretamente, las redes neuronales convolucionales (CNN).
3. Entender y explicar elementalmente los algoritmos para la optimización de redes neuronales
4. Entender y explicar un algoritmo novedoso para al detección de objetos en tiempo real, concretamente, You Only Look Once (YOLO) (Redmon; Divvala *et al.*, 2016).

No son objetivos de este trabajo la optimización de un modelo CNN para la detección de objetos por tener un privativo coste computacional ni el desarrollo de nuevos modelos en este campo. Junto a la elaboración de este trabajo, se implementan diversos algoritmos para la detección a tiempo real en imágenes. Ese trabajo y el código asociado se omiten en esta memoria por ser propiedad industrial de la Fundación CIDAUT.

1.2 APRENDIZAJE AUTOMÁTICO

Comenzaremos por una definición conceptual e intuitiva del concepto de aprendizaje, sin dar definiciones formales. Lo hacemos siguiendo a Goodfellow *et al.* (2016, p. 99): *«Se dice que un programa de ordenador aprende de la experiencia E con respecto a las tarea T y la medida de rendimiento P, si el rendimiento en T mejora con la experiencia E»*.

La tarea

Es sencillo imaginar todo tipo de tareas que puede realizar una máquina. Como la lista sería demasiado amplia, vamos a dar las tareas en términos de cómo actúa el algoritmo sobre un *ejemplo*. Un ejemplo es un conjunto de atributos, típicamente un vector $\boldsymbol{x} \in \mathbb{R}^n$. Por ejemplo, una imagen está dada por un vector con el valor de cada uno de sus píxeles. Aquí, cada píxel es un atributo. Otro ejemplo sería las coordenadas de un punto en el espacio o el valor de un conjunto de sensores de una máquina.

A modo de ejemplo y sin pretender proporcionar una lista exhaustiva o una taxonomía, algunas tareas comunes son:

- Clasificación: Queremos saber a que categoría de C posibles pertenece un ejemplo. Para resolver esta tarea, generalmente pedimos al algoritmo que especifique una función $h : \mathbb{R}^n \longrightarrow \{0, \ldots, C-1\}$. En particular, cuando $C = 2$ hablaremos de *clasificación binaria*.
- Regresión: Queremos predecir un valor numérico dado un ejemplo. Para resolver esta tarea, generalmente pediremos al algoritmo una función $h : \mathbb{R}^n \longrightarrow \mathbb{R}$.

- Transcripción: Queremos convertir una representación desestructurada de algún tipo de información, *e.g.* una imagen o una grabación de voz, en una forma discreta y escrita. Ejemplos de esta tarea son el reconocimiento óptico de caracteres y el reconocimiento de voz en las cuales queremos convertir una imagen con algo de texto o una onda de sonido en una secuencia de caracteres.

- Detección de anomalías: Queremos saber si un evento o actividad debe ser marcado como usual o atípico. Un ejemplo es la detección de fraude en el uso de las tarjetas de crédito. A partir de tus hábitos de compra, el algoritmo debe predecir si una compra realizada es inusual para suspender la operación.

- Síntesis y muestreo: Queremos generar nuevos ejemplos que son similares a los que podemos encontrar en los ejemplos de entrenamiento, en algunos casos delimitando los ejemplos en función de algún parámetro de entrada. Ejemplos de esta tarea son la síntesis de voz, producimos una onda de sonido a partir de un texto de entrada; la generación de una imagen dada una descripción, como hace *DALL·E 2*; o el completado de un texto, como hace *ChatGPT*.

- Reducción de ruido: Queremos conocer qué ejemplo $\mathbf{x} \in \mathbb{R}^n$ ha generado un ejemplo corrupto $\tilde{\mathbf{x}} \in \mathbb{R}^n$ a través de un proceso de corrupción desconocido o, más generalmente, la distribución de probabilidad condicionada $\mathbb{P}(\mathbf{x} \mid \tilde{\mathbf{x}})$ Un ejemplo, es la reducción de ruido de una imagen digital.

La experiencia

Todos los algoritmos que trataremos en este trabajo partirán de un conjunto de datos (del inglés *dataset*), *i.e.* un conjunto $\mathcal{X}$ de ejemplos. En función de cómo sea nuestro conjunto de datos clasificaremos los algoritmos de aprendizaje automático en dos categorías: aprendizaje supervisado y aprendizaje no supervisado.

- Aprendizaje **no** supervisado: Disponemos únicamente del conjunto de ejemplos $\mathcal{X}$ y queremos aprender propiedades sobre la estructura de $\mathcal{X}$ en base a los atributos de sus ejemplos.

  Un ejemplo es el *agrupamiento* (en inglés *clustering*) en el cual queremos agrupar los ejemplos por atributos similares.

- Aprendizaje supervisado: Además del conjunto de ejemplos $\mathcal{X}$ disponemos de un conjunto de etiquetas $\mathcal{Y}$ y una secuencia $S = \big((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)\big)$ de elementos de $\mathcal{X} \times \mathcal{Y}$.

  Por ejemplo, para la tarea de clasificación de cada elemento $\mathbf{x} \in \mathcal{X}$ disponemos de la categoría $\mathbf{y} \in \{1, \ldots, C\}$ a la que pertenece.

La medida de rendimiento

Para poder medir las capacidades de nuestro algoritmo de aprendizaje automático necesitamos una manera cuantitativa de medir su rendimiento. Para algunas tareas como la clasificación podemos usar la *tasa de error*, *i.e.* el porcentaje de ejemplos para los que producimos una respuesta incorrecta. En otras tareas, como síntesis o transcripción puede no tener sentido hablar de tasas de error.

Un aspecto importante sobre las medidas de rendimiento es sobre qué conjunto de datos las calculamos. ¿Qué sucede si entrenamos la red con los mismos datos con los que comprobamos su rendimiento? La capacidad de extrapolación de estos resultados a cualquier nuevo ejemplo que le podamos proporcionar a nuestra red se verá limitada. Por tanto, deberemos separar nuestro conjunto de datos en dos subconjutos disjuntos:

- Conjunto de entrenamiento
- Conjunto de validación

Elegir una medida de rendimiento es una tarea mucho más difícil de de lo que podría parecer *a priori*[1] y su elección determinará el desempeño de nuestro algoritmo de aprendizaje.

1.3 APRENDIZAJE SUPERVISADO

Como referencias principales a lo largo del capítulo seguiremos los textos de Goodfellow *et al.* (2016) y Shalev-Shwartz *et al.* (2014). Como el objetivo de este trabajo es abordar la detección automática de objetos en imágenes, vamos a restringir el marco teórico a los problemas de clasificación, detección y regresión (que usaremos para algunos ejemplos).

Nuestro modelo estadístico va a constar de tres partes principales:

- Información de entrada.
  - Dominio o conjunto de ejemplos
  - Conjunto de etiquetas
  - Conjunto de datos
- Información de salida
- Medida de rendimiento

Definición 1.1. Llamaremos *dominio* o *conjunto de ejemplos* a cualquier conjunto $\mathcal{X}$ sobre el que queramos predecir las etiquetas.

Típicamente, los elementos de $\mathcal{X}$ serán vectores de atributos, es decir, existen n conjuntos arbitrarios de atributos $\mathcal{X}_1, \ldots, \mathcal{X}_n$ tales que $\mathcal{X} \subseteq \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$.

Definición 1.2. Llamaremos *conjunto de etiquetas* a cualquier conjunto $\mathcal{Y}$ al que se puedan asignar elementos del dominio.

El conjunto de etiquetas podrá estar formado por vectores de etiquetas, es decir, pueden existir n conjuntos arbitrarios $\mathcal{Y}_1, \ldots, \mathcal{Y}_n$ tales que $\mathcal{Y} \subseteq \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_n$.

Definición 1.3. Llamaremos *conjunto de datos* S a cualquier secuencia $\big((\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\big)$ de elementos de $\mathcal{X} \times \mathcal{Y}$ que han sido generados por una distribución de probabilidad conjunta $\mathcal{D}$.

[1] Para información más detallada consúltese Goodfellow *et al.* (2016, p.103–104)

▶ **Definición 1.4.** La *información de entrada* disponible para el aprendizaje es una terna $(\mathcal{X}, \mathcal{Y}, S)$ formada por un dominio $\mathcal{X}$, un conjunto de etiquetas $\mathcal{Y}$ y un conjunto de entrenamiento S. No forma parte de la información de entrada la distribución de probabilidad $\mathcal{D}$ que genera el conjunto de datos.

Ejemplo 1.5 (clasificación binaria)**.** Supongamos que desea clasificar una imagen de una mancha en la piel entre cancerígena y no cancerígena. Las imágenes están en color, tamaño 640×480 píxeles, y cada píxel toma valores en el conjunto $\{0, 1, \ldots, 255\}$. En este caso, el dominio son todas las posibles imágenes a color $\mathcal{X} = \{0, 1, \ldots, 255\}^{640\times480\times3}$ y el conjunto de etiquetas es $\mathcal{Y} = \{\text{cáncer}, \text{no cáncer}\}$. No tenemos ningún cocimiento sobre la distribución de probabilidad $\mathcal{D}$.

Ejemplo 1.6 (regresión)**.** Supongamos que queremos calibrar un termómetro. Para ello, hemos medido la resistencia eléctrica del termómetro a diferentes temperaturas. En este caso, nuestro dominio son los posibles valores de la resistencia eléctrica $\mathcal{X} = [0, \infty) \subseteq \mathbb{R}$ y nuestro conjunto de etiquetas las posibles temperaturas que podemos medir $\mathcal{Y} = (0, \infty)$. El conjunto de datos son pares resistencia-temperatura $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ medidos durante el proceso de calibración.

▶ **Definición 1.7.** La *información de salida* es una función $h : \mathcal{X} \longrightarrow \mathcal{Y}$ a la que también llamaremos *regla* o *predictor*. El conjunto de las reglas, que llamaremos *clase de reglas* vamos denotarlo $\mathcal{H}$.

Medidas de rendimiento

▶ **Definición 1.8.** Dado un conjunto de datos $\mathcal{X}$ generado por una distribución de probabilidad $\mathcal{D}$ y un conjunto de etiquetas $\mathcal{Y}$, definimos la *función de pérdida* $\ell(y, x, h)$ como cualquier función $\ell : \mathcal{Y} \times \mathcal{X} \times \mathcal{H} \longrightarrow [0, \infty)$. Mide la discrepancia entre el valor predicho por un predictor $h : \mathcal{X} \longrightarrow \mathcal{Y}$, para un atributo x con etiqueta y.

Ejemplo 1.9 (regresión)**.** La pérdida habitual en el problema de regresión es la pérdida cuadrática

$$\ell(y, x, h) = \big(y - h(x)\big)^2$$

Ejemplo 1.10 (clasificación binaria)**.** La función de pérdida habitual en el problema de clasificación binaria es la pérdida 0-1

$$\ell(y, x, h) = \begin{cases} 0 & \text{si } h(x) = y \\ 1 & \text{si } h(x) \neq y \end{cases}$$

Habitualmente, al igual que en estos ejemplos, la función de perdida tiene la forma $\ell(y, x, h) = \ell(y, h(x))$.

Nótese que, dado que los ejemplos que observamos vienen generados por la distribución $\mathcal{D}$, los valores obtenidos al evaluar la función de pérdida para un ejemplo $(y, x) \sim \mathcal{D}$ son una variable aleatoria. Tiene sentido entonces considerar el valor esperado de esta variable al que llamaremos riesgo.

▶ **Definición 1.11.** Dado un conjunto de ejemplos $\mathcal{X}$ generados por una distribución $\mathcal{D}$ y un conjunto de etiquetas $\mathcal{Y}$, definimos el *riesgo* $\mathcal{R}_\mathcal{D}$ asociado a un predictor h como el valor esperado

$$\mathcal{R}_\mathcal{D}(h) \stackrel{\text{def}}{=} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\big(\ \ell(\mathbf{y},\mathbf{x},h)\ \big).$$

Como no tenemos acceso a la distribución $\mathcal{D}$ entonces el riesgo $\mathcal{R}_\mathcal{D}$ no puede ser medido. Únicamente tenemos acceso al conjunto de datos $S = \{(\mathbf{x}_1,\mathbf{y}_1),\dots,(\mathbf{x}_n,\mathbf{y}_n)\}$ que asumimos que es una muestra independiente e identicamente distribuida. Vamos a sustituir el valor esperado por la media muestral. Llamamos a esta nueva medida el *riesgo empírico*:

▶ **Definición 1.12.** Dado un conjunto de ejemplos $\mathcal{X}$, un conjunto de etiquetas $\mathcal{Y}$ y un conjunto de datos $S = \{(\mathbf{x}_1,\mathbf{y}_1),\dots,(\mathbf{x}_n,\mathbf{y}_n)\}$ generados por una distribución $\mathcal{D}$, definimos el *riego empírico* $\mathcal{R}_S$ asociado a una regla h como

$$\mathcal{R}_S(h) \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{y}_i,\mathbf{x}_i,h).$$

De esta forma, $\mathcal{R}_S$ es un estimador insesgado de $\mathcal{R}_\mathcal{D}$ y la ley de los grandes números nos garantiza que si S es grande, entonces $\mathcal{R}_S$ y $\mathcal{R}_\mathcal{D}$ se parecerán.

El problema del aprendizaje supervisado consiste en encontrar un predictor h que minimice el riesgo empírico $\mathcal{R}_S(h)$ dentro de un conjunto de predictores $\mathcal{H}$. Es decir, buscamos

$$\hat{h} = \operatorname*{argmin}_{h\in\mathcal{H}} \mathcal{R}_S(h).$$

A este conjunto de reglas se las conoce como reglas ERM (del inglés, *Empirical Risk Minimization*)

Compromiso estimación–aproximación

▶ **Definición 1.13.** Dado un conjunto de ejemplos $\mathcal{X}$ generados por una distribución $\mathcal{D}$ y un conjunto de etiquetas $\mathcal{Y}$, llamamos *regla de Bayes* a la que minimiza el riesgo. Es decir,

$$h_B \stackrel{\text{def}}{=} \operatorname*{argmin}_{h\in\mathcal{H}} \mathcal{R}_\mathcal{D}(h).$$

La regla de Bayes proporciona el mínimo riesgo posible para un problema dado, pero no puede calcularse pues depende de la distribución desconocida $\mathcal{D}$. Podemos entonces estudiar el exceso de riesgo en que incurrimos por usar una regla ERM $\hat{h} \in \mathcal{H}$ distinta de la regla de Bayes h_B. Para cada $h \in \mathcal{H}$ se tiene

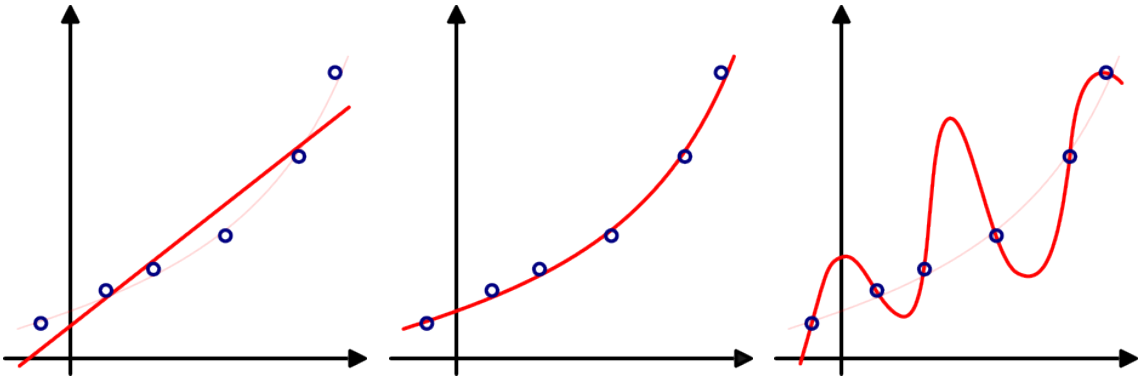
$$\begin{aligned}
\mathcal{R}_\mathcal{D}(\hat{h}) - \mathcal{R}_\mathcal{D}(h_B) &= \mathcal{R}_\mathcal{D}(\hat{h}) - \mathcal{R}_S(\hat{h}) + \mathcal{R}_S(\hat{h}) - \mathcal{R}_\mathcal{D}(h) + \mathcal{R}_\mathcal{D}(h) - \mathcal{R}_\mathcal{D}(h_B) \\
&\le \mathcal{R}_\mathcal{D}(\hat{h}) - \mathcal{R}_S(\hat{h}) + \mathcal{R}_S(h) - \mathcal{R}_\mathcal{D}(h) + \mathcal{R}_\mathcal{D}(h) - \mathcal{R}_\mathcal{D}(h_B) \\
&\le \sup_{h\in\mathcal{H}} |\mathcal{R}_\mathcal{D}(h) - \mathcal{R}_S(h)| + \sup_{h\in\mathcal{H}} |\mathcal{R}_S(h) - \mathcal{R}_\mathcal{D}(h)| + \mathcal{R}_\mathcal{D}(h) - \mathcal{R}_\mathcal{D}(h_B)
\end{aligned}$$

En particular, esto también se verifica para la regla que minimiza $|\mathcal{R}_\mathcal{D}(h) - \mathcal{R}_\mathcal{D}(h_B)|$. Por

tanto,

$$\mathcal{R}_{\mathcal{D}}(\hat{h}) - \mathcal{R}_{\mathcal{D}}(h_B) \leq 2 \sup_{h \in \mathcal{H}} |\mathcal{R}_S(h) - \mathcal{R}_{\mathcal{D}}(h)| + \min_{h \in \mathcal{H}} |\mathcal{R}_{\mathcal{D}}(h) - \mathcal{R}_{\mathcal{D}}(h_B)|$$

donde el término $\sup_{h \in \mathcal{H}} |\mathcal{R}_S(h) - \mathcal{R}_{\mathcal{D}}(h)|$ se conoce como *error de estimación* y el término $\min_{h \in \mathcal{H}} |\mathcal{R}_{\mathcal{D}}(h) - \mathcal{R}_{\mathcal{D}}(h_B)|$ como *error de aproximación*. El error de estimación representa el error cometido por no tener acceso a la distribución $\mathcal{D}$ y usar solo el conjunto S. Este término aumenta con el tamaño de $\mathcal{H}$. El error de aproximación se debe a que las funciones de $\mathcal{H}$ pueden no aproximarse suficiente a h_B. Este término se reduce cuando aumenta el tamaño de $\mathcal{H}$. Nótese que $\mathcal{R}_{\mathcal{D}}(\hat{h}) - \mathcal{R}_{\mathcal{D}}(h_B) \geq 0$ porque $\mathcal{R}_{\mathcal{D}}(h_B)$ es el mínimo riesgo posible. La figura 1.1 ilustra estos errores.



(a) $\mathcal{H}$ pequeño: error de aproximación

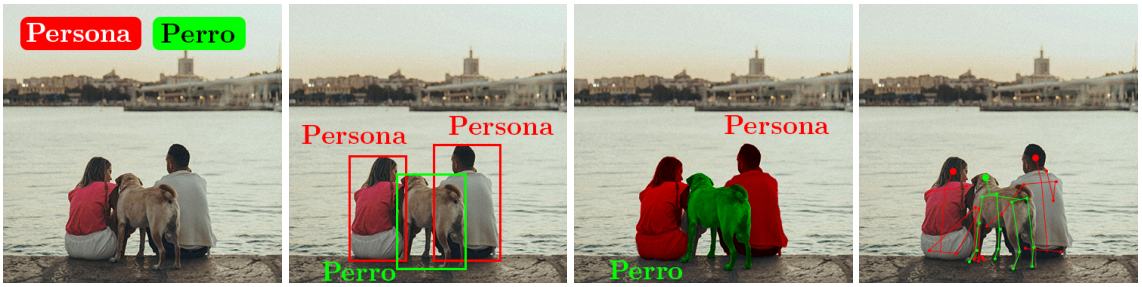
(b) $\mathcal{H}$ adecuado

(c) $\mathcal{H}$ grande: error de estimación

Figura 1.1: Compromiso aproximación-estimación. Comparación entre (a) *error de aproximación*, (b) adecuado, y (c) *error de estimación*.

1.4 PROBLEMAS DE VISIÓN ARTIFICIAL

Clásicamente existen cuatro tareas de visión artificial: *clasificación*, *detección*, *segmentación* y *estimación de postura*. La clasificación busca decidir que elementos están presentes en una imagen. La detección persigue determinar la posición de un objeto en una imagen y la categoría de este objeto. La segmentación busca determinar qué píxeles corresponden a un objeto o categoría. La estimación de postura busca determinar la posición de las articulaciones de una persona o animal.



(a) Clasificación (b) Detección (c) Segmentación (d) Postura

Figura 1.2: Tareas clásicas de visión artificial.

Este trabajo se centra en la tarea de detección en dos dimensiones. Nuestro objetivo será encontrar *rectángulos delimitadores* o *cajas* como los de la figura 1.2b.

Capítulo 2

REDES NEURONALES

Las redes neuronales son una clase de funciones construida mediante la composición sucesiva de funciones lineales y funciones no lineales. La introducción de las funciones no lineales es estrictamente necesaria pues la composición de funciones lineales resulta nuevamente en una función lineal. En este capítulo estudiamos las redes neuronales prealimentas o FNN (del inglés *Feed-forward Neural Networks*) y, dentro de estas, las redes neuronales convolucionales. También, estudiaremos el entrenamiento mediante los algoritmos de *descenso del gradiente* (GD), *descenso estocástico del gradiente* (SGD) y *descenso estocástico del gradiente con momentos*. Cerraremos el capítulo con algunos ejemplos con relevancia histórica de modelos basados en CNN.

2.1 CONSTRUCCIÓN DE REDES NEURONALES

Vamos a suponer en todo momento que el conjunto de atributos es $\mathcal{X} = \mathbb{R}^d$ y que el conjunto de etiquetas es $\mathcal{Y} = \mathbb{R}^n$.

▶ **Definición 2.1.** Una *red neuronal prealimentada* es una función $h_{\boldsymbol{\theta}} : \mathbb{R}^d \longrightarrow \mathbb{R}^n$ que verifica ser composición de funciones de la forma

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = (\mathbf{F}_L \circ W_L) \circ (\mathbf{F}_{L-1} \circ W_{L-1}) \circ (\mathbf{F}_{L-2} \circ W_{L-2}) \circ \cdots \circ (\mathbf{F}_1 \circ W_1)\, \mathbf{x} \tag{2.1}$$

donde $\mathbf{F}_l : \mathbb{R}^{d'_l} \longrightarrow \mathbb{R}^{d_l}$, $1 \leq l \leq L$ es una función no lineal y W_l, $1 \leq l \leq L$ es una matriz $(w_{i,j})_l \in \mathcal{M}_{d'_l,\, d_{l-1}}(\mathbb{R})$ con d'_l filas y d_{l-1} columnas y $\boldsymbol{\theta} = (W_1, \ldots, W_L)$. Abusando de notación, se denota igual a las matrices W_l que a la aplicación lineal $W_l : \mathbb{R}^{d_{l-1}} \longrightarrow \mathbb{R}^{d'_l}$

A los elementos $w_{i,j}^{(l)}$ se los denomina *pesos* de la red. Cada una de las funciones intermedias $(F_l \circ W_l)$ se denomina *capa de la red*. El vector de entrada $\mathbf{x}$ recibe el nombre de *capa de entrada* y la última capa $\mathbf{F}_L \circ W_L$ de *capa de salida*. Las capas intermedias reciben el nombre de *capas ocultas*.

La función anterior puede expresarse de forma recursiva como

$$\mathbf{a}_l = \mathbf{F}_l\big(W_l \mathbf{a}_{l-1}\big), \text{ para cada } l = 1, \ldots, L;\ \mathbf{a}_0 = \mathbf{x} \tag{2.2}$$

y tomar $h_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{a}_L$.

Lo más habitual será que $d_l = d'_l$ y la función $\mathbf{F}$ se construya como aplicación en cada elemento de una función $f : \mathbb{R} \longrightarrow \mathbb{R}$ que recibe el nombre de *función de activación*. Es decir,

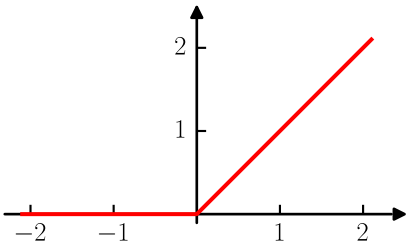
$$\mathbf{F}(x_1, \ldots, x_{d_l}) = \big(f(x_1), \ldots, f(x_{d_l})\big). \tag{2.3}$$

2.1.1 ALGUNOS EJEMPLOS DE FUNCIONES DE ACTIVACIÓN

Sin proporcionar una lista exhaustiva de funciones de activación comunes, se presentan aquí algunas de las más utilizadas:

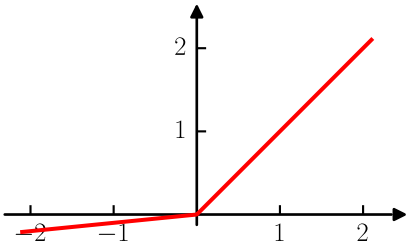
1. ReLU (Rectified Linear Unit)

$$f(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases} \tag{2.4a}$$



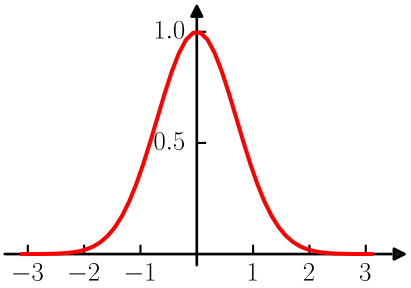
2. Leaky ReLU

$$f(x) = \begin{cases} 0.1x & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases} \tag{2.4b}$$



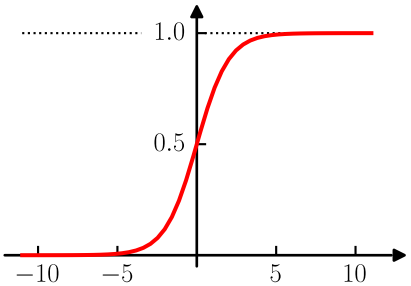
3. Gaussiana

$$\phi_{0,1}(x) = e^{-x^2} \tag{2.4c}$$



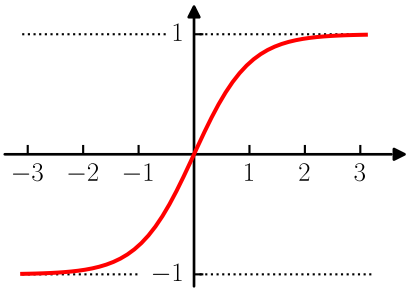
4. Logística o Sigmoide

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4d}$$



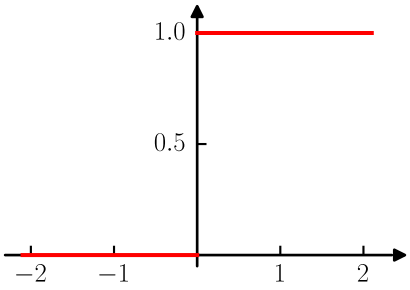
5. Tangente hiperbólica

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4e}$$



6. Binaria

$$I_{(0,\infty)}(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 & \text{si } x > 0 \end{cases} \qquad (2.4f)$$



7. Softmax $\boldsymbol{\sigma} : \mathbb{R}^n \longrightarrow [0,1]^n$

$$\boldsymbol{\sigma}_i(x_1, \ldots, x_n) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad (2.4g)$$

2.1.2 GRAFO ASOCIADO A UNA RED NEURONAL

Es habitual representar las redes neuronales mediante el uso de grafos, partiendo de una representación mínima llamada *neurona*. Una neurona es una función $g_{\mathbf{w},f}(\mathbf{x}) = f\left(\sum_{k=1}^{d_l} w_k x_k\right)$ que viene representada esquemáticamente en la figura 2.1.

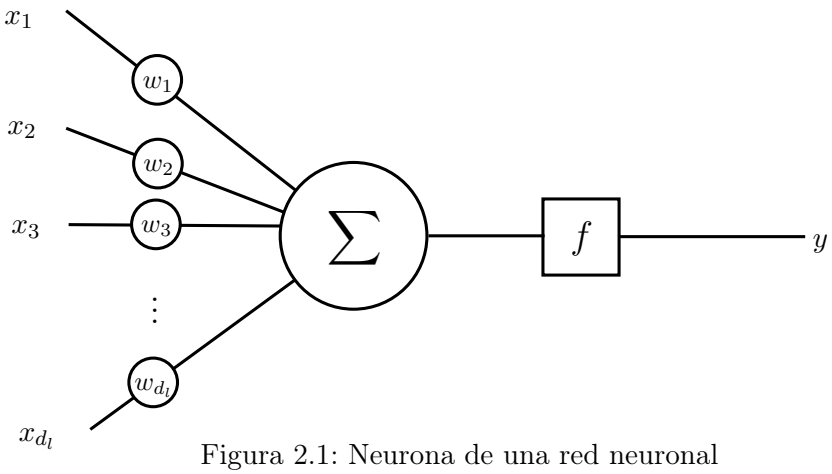


Figura 2.1: Neurona de una red neuronal

Después, concatenaremos estas funciones de acuerdo a un grafo ordenado en capas como el de la figura 2.2. Cada arista del grafo corresponde a un peso w no nulo. De esta forma, en cada capa l, las d_l' sumas ponderadas $\sum_{k=1}^{d_l} w_k x_k$ se agrupan en la función lineal W_l y las funciones de activación en la función no lineal $\mathbf{F}$ en la forma (2.3). Vemos que esta forma de expresar las redes neuronales verifica la definición 2.1.

▶ **Definición 2.2.** Diremos que una capa es *densa* si entre cualquiera dos neuronas de dos capas existe una arista. Una red formada únicamente por capas densas se llama *perceptrón multicapa* (MLP) (del inglés *mulitilayer perceptron*).

Sin embargo, como en este trabajo emplearemos imágenes más adelante, en ocasiones, nos será más conveniente hablar en términos de matrices o tensores. Para nosotros un tensor será simplemente un arreglo multidimensional de números. Operar con matrices o tensores no es un problema en la definición 2.1 de redes neuronales. Siempre que estemos trabajando con un espacio vectorial sobre $\mathbb{R}$, podremos fijar una base y operar con sus coordenadas. Por ello, emplearemos diagramas con cajas como muestra la figura 2.3 para

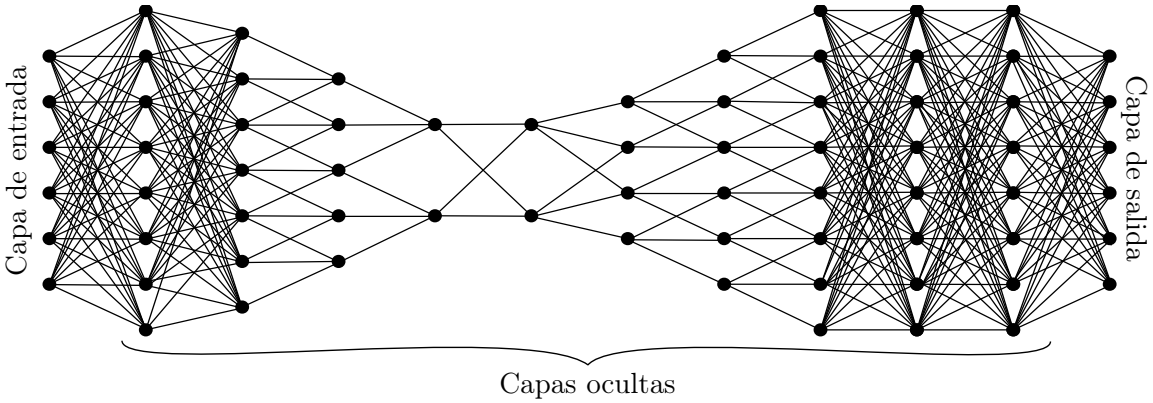


Figura 2.2: Ejemplo de grafo asociado a una red neuronal. Los nodos representas las neuronas y las aristas pesos no nulos. Las columnas de nodos son las capas de la red.

ilustrar más intuitivamente los grafos asociados a la red neuronal.

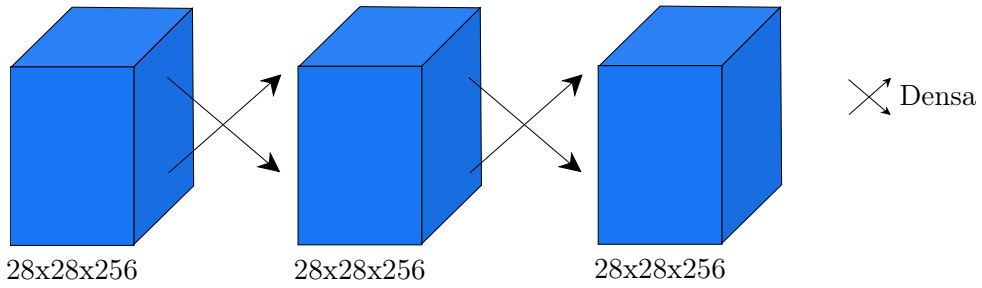


Figura 2.3: Ejemplo de grafo de una red neuronal con 3 capas en las que todas los nodos están conectados con todos los nodos de la siguiente capa.

2.2 REDES NEURONALES CONVOLUCIONALES

Las redes neuronales convolucionales son aquellas que cuentan con, al menos, una capa convolucional. El término *convolucional* hace referencia a la forma precisa que tendrá la aplicación lineal W_ℓ de esa capa.

Para estudiar las capas convolucionales, en lugar de dar un definición operativa de estas, vamos a estudiar la operación de convolución, que seguramente ya sea familiar para lector. Usando esta operación, vamos a construir las capas convolucionales. Además, estudiaremos una función no lineal $\mathbf{F}_\ell$ que suele usarse junto a las capas convolucionales: el pooling. Vamos a emplear como referencias principales el artículo de Dumoulin *et al.* (2016) y el libro de Goodfellow *et al.* (2016).

2.2.1 CONVOLUCIÓN

▶ **Definición 2.3.** Dadas dos funciones medibles $f, g : \mathbb{R}^n \longrightarrow \mathbb{R}$ su *producto de convolución* $f * g : \mathbb{R}^n \longrightarrow \mathbb{R}$ es una función dada por

$$(f * g)(\mathbf{x}) = \int_{\mathbb{R}^n} f(\mathbf{y}) g(\mathbf{x} - \mathbf{y})\, d\mathbf{y}$$

cuando la integral tenga sentido. A la función g es habitual referirse como *núcleo* de la convolución.

Para el caso discreto sustituiremos las integrales por sendas sumas.

▶ **Definición 2.4.** Dadas dos funciones $f, g : \mathbb{Z}^n \longrightarrow \mathbb{R}$, definimos su producto de convolución $f * g$ como

$$(f * g)(i_1, \ldots, i_n) = \sum_{k_1=-\infty}^{\infty} \cdots \sum_{k_n=-\infty}^{\infty} f(k_1, \ldots, k_n) g(i_1 - k_1, \ldots, i_n - k_n)$$

cuando el sumatorio tenga sentido.

Ejemplo 2.5 (Convolución discreta 1D)**.** Sean $\mathbf{a} = (a_1, a_2, \ldots, a_5)^t$ y $\mathbf{b} = (b_1, b_2, b_3)^t$. Extendemos estos a funciones como

$$f(i) = \begin{cases} a_{i+1} & \text{si } 0 \leq i \leq 5 - 1 \\ 0 & \text{en cualquier otro caso} \end{cases} \qquad g(i) = \begin{cases} b_{i+1} & \text{si } 0 \leq i \leq 3 - 1 \\ 0 & \text{en cualquier otro caso} \end{cases}$$

podemos calcular la convolución de ambas funciones como

$$(f * g)(i) = \sum_{k=-\infty}^{\infty} f(k) g(i - k)$$

Los elementos no nulos vienen dados por

$$(f * g)(i) = \begin{pmatrix} \vdots \\ (f*g)(-1) \\ (f*g)(0) \\ (f*g)(1) \\ (f*g)(2) \\ (f*g)(3) \\ (f*g)(4) \\ (f*g)(5) \\ (f*g)(6) \\ (f*g)(7) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ 0 \\ a_1 b_1 \\ a_1 b_2 + a_2 b_1 \\ a_1 b_3 + a_2 b_2 + a_3 b_1 \\ a_2 b_3 + a_3 b_2 + a_4 b_1 \\ a_3 b_3 + a_4 b_2 + a_5 b_1 \\ a_4 b_3 + a_5 b_2 \\ a_5 b_3 \\ 0 \\ \vdots \end{pmatrix}$$

Podemos dar una representación visual de esta operación en la cual vamos moviendo el segundo vector conforme avanzamos por el primer vector.

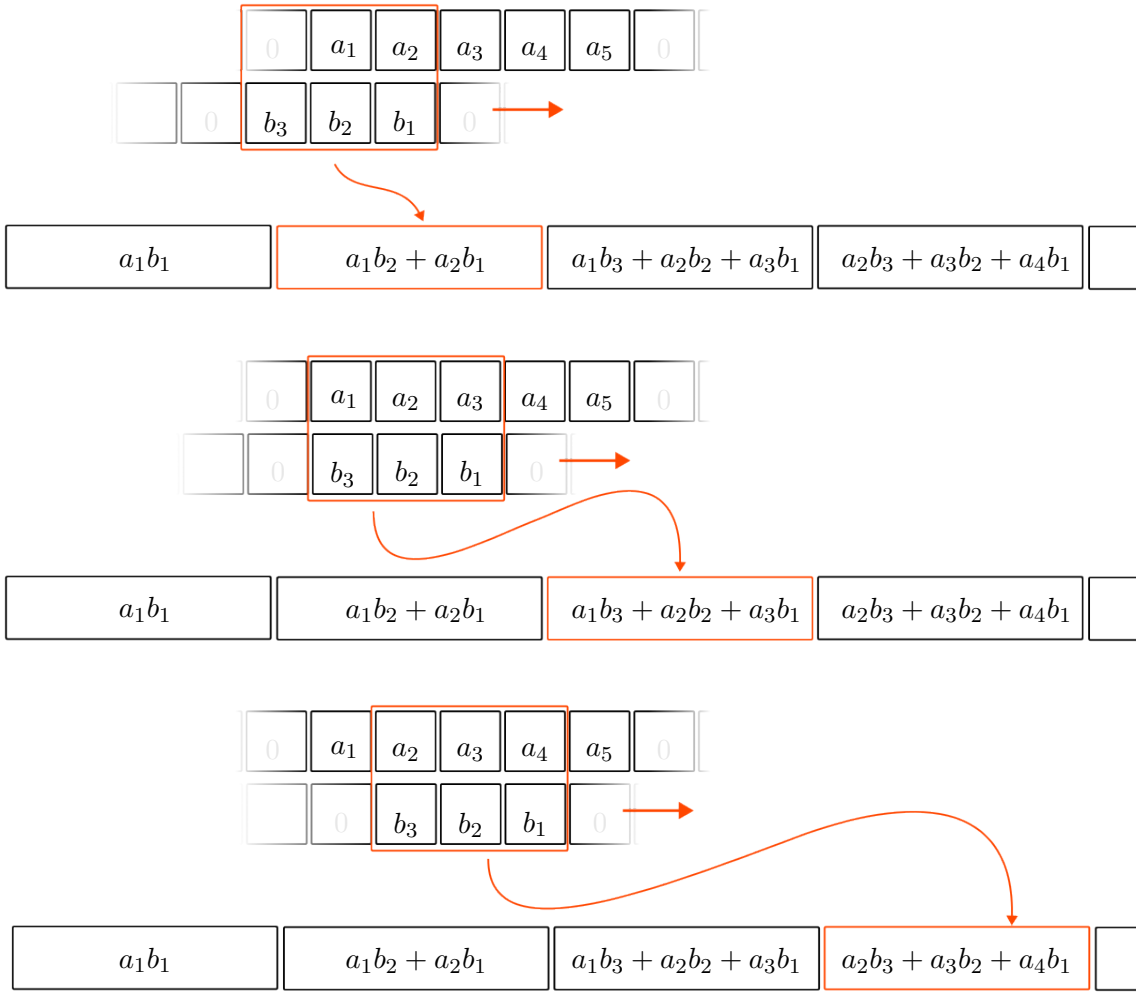


Figura 2.4: Representación visual de la convolución discreta en 1D.

Siguiendo el ejemplo 2.5 podemos extender la definición de convolución discreta 2.4 a vectores si formamos el vector resultado a partir de los elementos no nulos de $f * g$.

► **Definición 2.6.** Dados dos vectores $\mathbf{a} \in \mathbb{R}^n$ y $\mathbf{b} \in \mathbb{R}^m$ con $m \leq n$ definimos la *convolución discreta* de $\mathbf{a}$ y $\mathbf{b}$ con *relleno* (del inglés *padding*) $p \in \{0, 1, \ldots\}$ y *paso* $s \in \{1, 2, \ldots\}$ (del inglés *stride*), $(\mathbf{a} * \mathbf{b})_{p,s} \in \mathbb{R}^{\lfloor \frac{n-m+1+2p}{s} \rfloor}$ mediante el siguiente procedimiento:

1. Definimos $f : \mathbb{Z} \longrightarrow \mathbb{R}$ y $g : \mathbb{Z} \longleftrightarrow \mathbb{R}$ como

$$f(i) = \begin{cases} a_{i+1} & \text{si } 0 \leq i \leq n-1 \\ 0 & \text{en cualquier otro caso} \end{cases} \qquad g(i) = \begin{cases} b_{i+1} & \text{si } 0 \leq i \leq m-1 \\ 0 & \text{en cualquier otro caso} \end{cases}$$

2. Calculamos el producto de convolución $f * g$

3. Tomamos los elementos del vector $(\mathbf{a} * \mathbf{b})_{p,s}$ como

$$\big((\mathbf{a} * \mathbf{b})_{p,s}\big)_i = (f * g)\big((i-1)s + (m-1-p)\big), \quad 1 \leq i \leq \left\lfloor \frac{n-m+1+2p}{s} \right\rfloor$$

Ejemplo 2.7 (Convolución discreta de vectores)**.** Siguiendo con los vectores del ejemplo 2.5, obtenemos las funciones f, g y su convolución $f*g$. Ahora debemos elegir los elementos de $f*g$ que conforman el vector $(\mathbf{a}*\mathbf{b})_{p,s}$.

- Relleno $p=0$ y paso $s=1$. No tomamos ningún 0 de la extensión de $\mathbf{a}$ y avanzamos de uno en uno por $f*g$:

$$(\mathbf{a}*\mathbf{b})_{0,1} = \begin{pmatrix} a_1b_3+a_2b_2+a_3b_1 \\ a_2b_3+a_3b_2+a_4b_1 \\ a_3b_3+a_4b_2+a_5b_1 \end{pmatrix}$$

- Relleno $p=1$ y paso $s=1$. Tomamos un cero a cada lado de $\mathbf{a}$ y avanzamos de uno en uno por $f*g$:

$$(\mathbf{a}*\mathbf{b})_{1,1} = \begin{pmatrix} a_1b_2+a_2b_1 \\ a_1b_3+a_2b_2+a_3b_1 \\ a_2b_3+a_3b_2+a_4b_1 \\ a_3b_3+a_4b_2+a_5b_1 \\ a_4b_3+a_5b_2 \end{pmatrix}$$

- Relleno $p=2$ y paso $s=1$. Tomamos un cero a cada lado de $\mathbf{a}$ y avanzamos de uno en uno por $f*g$:

$$(\mathbf{a}*\mathbf{b})_{2,1} = \begin{pmatrix} a_1b_1 \\ a_1b_2+a_2b_1 \\ a_1b_3+a_2b_2+a_3b_1 \\ a_2b_3+a_3b_2+a_4b_1 \\ a_3b_3+a_4b_2+a_5b_1 \\ a_4b_3+a_5b_2 \\ a_5b_3 \end{pmatrix}$$

- Relleno $p=0$ y paso $s=2$. No tomamos ningún 0 de la extensión de $\mathbf{a}$ y avanzamos de dos en dos por $f*g$:

$$(\mathbf{a}*\mathbf{b})_{0,2} = \begin{pmatrix} a_1b_3+a_2b_2+a_3b_1 \\ a_3b_3+a_4b_2+a_5b_1 \end{pmatrix}$$

- Relleno $p=1$ y paso $s=2$. Tomamos un cero a cada lado de $\mathbf{a}$ y avanzamos de dos en dos por $f*g$:

$$(\mathbf{a}*\mathbf{b})_{1,2} = \begin{pmatrix} a_1b_2+a_2b_1 \\ a_2b_3+a_3b_2+a_4b_1 \\ a_4b_3+a_5b_2 \end{pmatrix}$$

Proposición 2.8. Sean $n, m, s, p \in \mathbb{N}$ tales que $m \leq n$, $\mathbf{a} \in \mathbb{R}^n$ y $b \in \mathbb{R}^m$. La aplicación dada por $\mathbf{a} \mapsto (\mathbf{a}*\mathbf{b})_{p,s}$ es lineal.

Demostración. Sean $n, m, s, p \in \mathbb{N}$ tales que $m \leq n$. Sean $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, $\lambda, \mu \in \mathbb{R}$. Definimos las funciones f_1, f_2 y g con los vectores $\mathbf{a}_1, \mathbf{a_2}, \mathbf{b}$ respectivamente siguiendo la definición 2.6. La función f asociada a $\lambda\mathbf{a}_1 + \mu\mathbf{a}_2$ es $f = \lambda f_1 + \mu f_2$. Si tenemos en cuenta que la convolución es distributiva y asociativa con la multiplicación por escalares, entonces para cada $1 \leq i \leq \left\lfloor \frac{n-m+1+2p}{s} \right\rfloor$

$$\begin{aligned} \big(([\lambda\mathbf{a}_1+\mu\mathbf{a}_2]*\mathbf{b})_{p,s}\big)_i &= (f*g)\big((i-1)s+(m-1-p)\big) \\ &= \lambda(f_1*g)\big((i-1)s+(m-1-p)\big) + \mu(f_2*g)\big((i-1)s+(m-1-p)\big) \\ &= \lambda\big((\mathbf{a}_1*\mathbf{b})_{p,s}\big)_i + \mu\big((\mathbf{a}_2*\mathbf{b})_{p,s}\big)_i \end{aligned}$$

□

Ejemplo 2.9 (Convolución de vectores como aplicación lineal)**.** Continuamos el ejemplo 2.7. El núcleo es el vector $\mathbf{b} = (b_1, b_2, b_3)^t \in \mathbb{R}^3$, que actúa como pesos de la aplicación lineal asociada.

- Relleno $p = 0$ y paso $s = 1$:

$$W = \begin{pmatrix} b_3 & b_2 & b_1 & 0 & 0 \\ 0 & b_3 & b_2 & b_1 & 0 \\ 0 & 0 & b_3 & b_2 & b_1 \end{pmatrix}$$

- Relleno $p = 1$ y paso $s = 1$:

$$W = \begin{pmatrix} b_2 & b_1 & 0 & 0 & 0 \\ b_3 & b_2 & b_1 & 0 & 0 \\ 0 & b_3 & b_2 & b_1 & 0 \\ 0 & 0 & b_3 & b_2 & b_1 \\ 0 & 0 & 0 & b_3 & b_2 \end{pmatrix}$$

- Relleno $p = 2$ y paso $s = 1$:

$$W = \begin{pmatrix} b_1 & 0 & 0 & 0 & 0 \\ b_2 & b_1 & 0 & 0 & 0 \\ b_3 & b_2 & b_1 & 0 & 0 \\ 0 & b_3 & b_2 & b_1 & 0 \\ 0 & 0 & b_3 & b_2 & b_1 \\ 0 & 0 & 0 & b_3 & b_2 \\ 0 & 0 & 0 & 0 & b_3 \end{pmatrix}$$

- Relleno $p = 0$ y paso $s = 2$:

$$W = \begin{pmatrix} b_3 & b_2 & b_1 & 0 & 0 \\ 0 & 0 & b_3 & b_2 & b_1 \end{pmatrix}$$

- Relleno $p = 1$ y paso $s = 2$:

$$W = \begin{pmatrix} b_2 & b_1 & 0 & 0 & 0 \\ 0 & b_3 & b_2 & b_1 & 0 \\ 0 & 0 & 0 & b_3 & b_2 \end{pmatrix}$$

Observación 2.10. La proposición 2.8 es la propiedad fundamental que liga la definición 2.1 de la red neural con definición 2.6 de la convolución discreta de vectores. De esta forma, vemos como en las redes neuronales convolucionales, el término *convolucionales* hace referencia a una forma particular de elegir las aplicaciones lineales de la red. El núcleo de la convolución son los pesos que deberemos encontrar para minimizar el riesgo empírico. Estos pesos se deberán encontrar satisfaciendo las condiciones de igualdad que se imponen.

La convolución de vectores se puede extender de la misma manera a la convolución de matrices siguiendo el mismo procedimiento.

▶ **Definición 2.11.** Dados dos matrices $A \in \mathbb{R}^{n_1 \times n_2}$ y $B \in \mathbb{R}^{m_1 \times m_2}$ con $m_1 \leq n_1$ y $m_2 \leq n_2$ definimos la *convolución discreta* de A y B con *relleno* (del inglés *padding*) $p_1, p_2 \in \{0, 1, \ldots\}$ y *paso* $s_1, s_2 \in \{1, 2, \ldots\}$ (del inglés *stride*),

$$(A * B)_{p_1,p_2,s_1,s_2} \in \mathbb{R}^{\lfloor \frac{n_1 - m_1 + 1 + 2p_1}{s_1} \rfloor \times \lfloor \frac{n_2 - m_2 + 1 + 2p_2}{s_2} \rfloor}$$

mediante el siguiente procedimiento:

1. Definimos $f : \mathbb{Z}^2 \longrightarrow \mathbb{R}$ y $g : \mathbb{Z}^2 \longleftrightarrow \mathbb{R}$ como

$$f(i,j) = \begin{cases} A_{i+1,\,j+1} & \text{si } 0 \leq i \leq n_1 - 1,\ 0 \leq j \leq n_2 - 1 \\ 0 & \text{en cualquier otro caso} \end{cases}$$

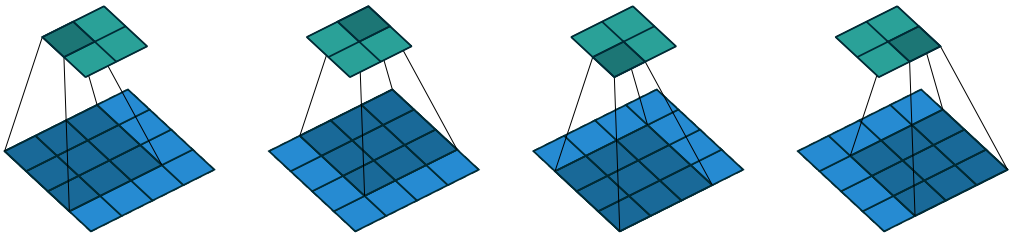
$$g(i,j) = \begin{cases} B_{i+1,\,j+1} & \text{si } 0 \leq i \leq m_1 - 1,\ 0 \leq j \leq m_2 - 1 \\ 0 & \text{en cualquier otro caso} \end{cases}$$

2. Calculamos el producto de convolución $f * g$
3. Tomamos los elementos de la matriz $(\mathbf{A} * \mathbf{B})_{p_1,p_2,s_1,s_2}$ como

$$\big((A*B)_{p_1,p_2,s_1,s_2}\big)_{i,j} = (f*g)\big((i-1)s_1 + (m_1 - 1 - p_1), (j-1)s_2 + (m_2 - 1 - p_2)\big),$$
$$1 \leq i \leq \left\lfloor \frac{n_1 - m_1 + 1 + 2p_1}{s_1} \right\rfloor,\ 1 \leq j \leq \left\lfloor \frac{n_2 - m_2 + 1 + 2p_2}{s_2} \right\rfloor$$

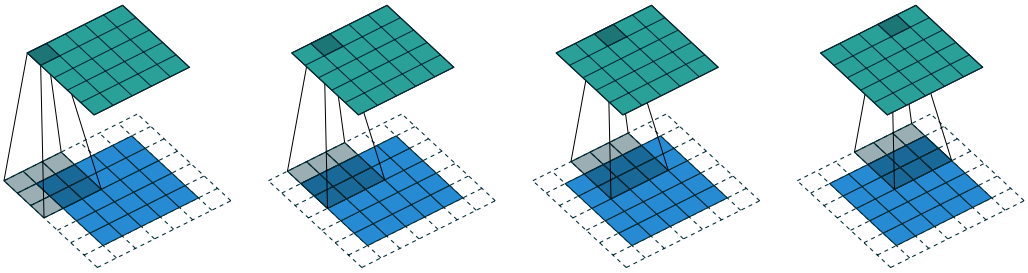
Ejemplo 2.12 (Convolución discreta de matrices)**.** Vamos a realizar la convolución de A (inferior, azul) con el núcleo B (inferior, sombreado) para obtener $(A*B)_{s_1,s_2,p_1,p_2}$ (superior, verde)

- Relleno $p_1 = p_2 = 0$, paso $s_1 = s_2 = 1$.



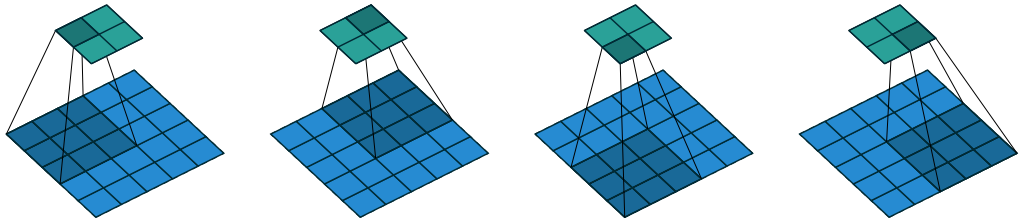
Crédito: Dumoulin *et al.* (2016).

- Relleno $p_1 = p_2 = 1$, paso $s_1 = s_2 = 1$.



Crédito: Dumoulin *et al.* (2016).

- Relleno $p_1 = p_2 = 0$, paso $s_1 = s_2 = 2$.



Crédito: Dumoulin *et al.* (2016).

Proposición 2.13. Sean $n_1, n_2, m_1, m_2, s_1, s_2, p_1, p_2 \in \mathbb{N}$ tales que $m_i \leq n_i \ \forall i = 1, 2$; $A \in \mathbb{R}^{n_1 \times n_2}$ y $B \in \mathbb{R}^{m_1 \times m_2}$. La aplicación dada por $A \mapsto (A * B)_{p_1,p_2,s_1,s_2}$ es lineal.

Demostración. La demostración es totalmente análoga a la de la proposición 2.8. Basta hacer uso de que la convolución es distributiva y asociativa con el producto por escalares. □

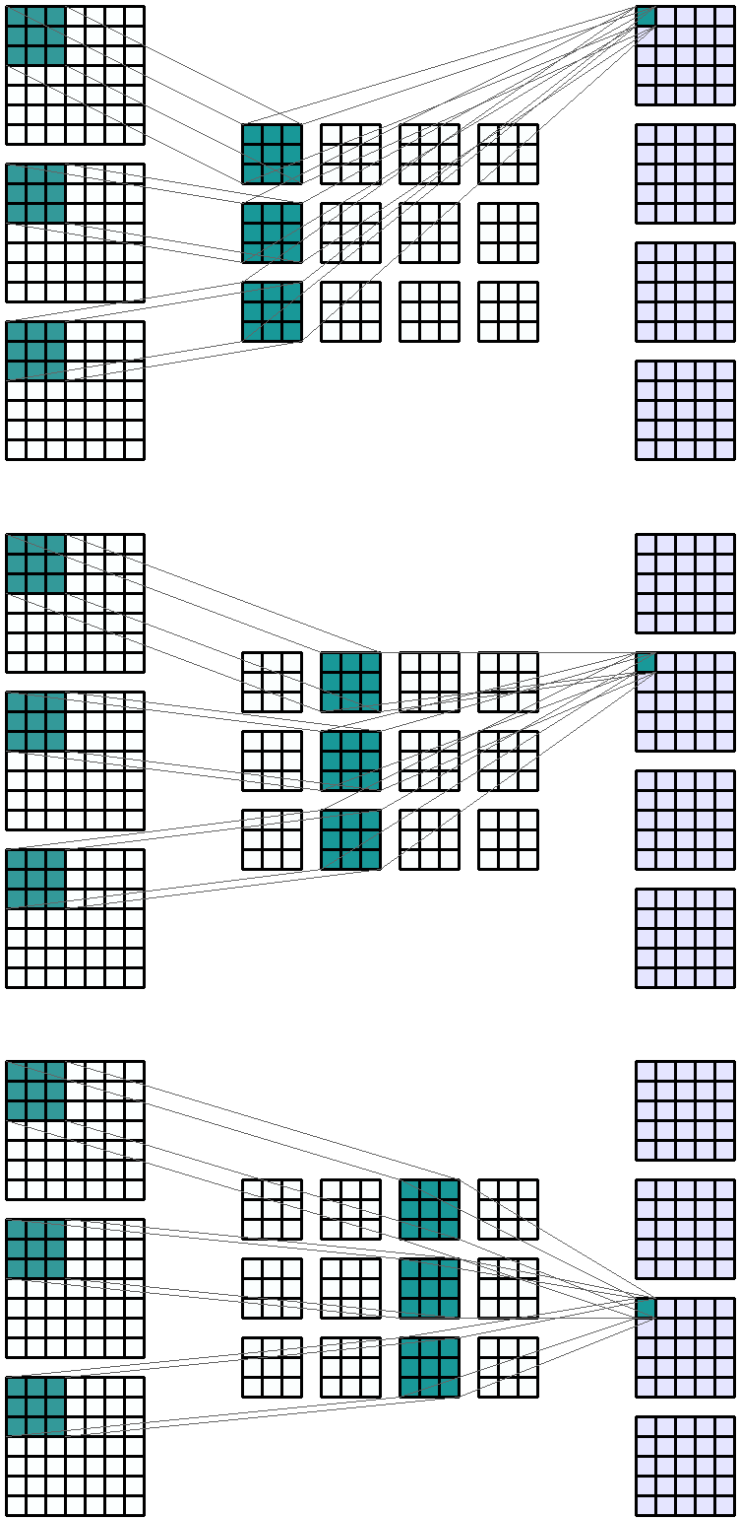
Ejemplo 2.14 (Convolución de matrices como aplicación lineal)**.** Vamos a usar como núcleo una matriz $B \in \mathbb{R}^{3\times3}$. Si usamos la base canónica para el espacio de matrices, la aplicación lineal $W : \mathbb{R}^{4\times4} \longrightarrow \mathbb{R}^{2\times2}$ dada por $A \mapsto (A * B)_{0,0,1,1}$ (relleno nulo y paso unidad) se representa por la matriz:

$$W = \begin{pmatrix} b_{33} & b_{32} & b_{31} & 0 & b_{23} & b_{22} & b_{21} & 0 & b_{13} & b_{12} & b_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & b_{33} & b_{32} & b_{31} & 0 & b_{23} & b_{22} & b_{21} & 0 & b_{13} & b_{12} & b_{11} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & b_{33} & b_{32} & b_{31} & 0 & b_{23} & b_{22} & b_{21} & 0 & b_{13} & b_{12} & b_{11} & 0 \\ 0 & 0 & 0 & 0 & 0 & b_{33} & b_{32} & b_{31} & 0 & b_{23} & b_{22} & b_{21} & 0 & b_{13} & b_{12} & b_{11} \end{pmatrix}$$

Capas convolucionales

En este trabajo vamos a emplear capas de convolución en 2D por lo que nos centraremos en estas. Las capas de convolución se emplean aplicando convoluciones con diversos núcleos a cada una de los canales de la capa anterior y sumando todos las salidas como en el siguiente ejemplo.

Ejemplo 2.15. Imaginemos que tenemos una primera imagen 7×7 píxeles con tres canales RGB y queremos obtener una nueva imagen 5×5 píxeles con 4 canales. Para reducir el tamaño de 7 a 5 emplearemos un núcleo de convolución 3×3 sin relleno y de paso 1. Aplicaremos un núcleo a cada canal. Entonces obtenemos 3 imágenes 5×5 y las sumamos todas elemento a elemento para obtener la primera capa. Para la segunda capa repetimos el procedimiento con otros 3 núcleos etc... como se muestra en la figura 2.5.



Crédito: Thevenot (2020).

Figura 2.5: Capa convolucional de 2 dimensiones con 3 canales de entrada y 4 canales de salida. A la izquierda, los canales de entrada; en el medio los núcleos de convolución y; a la derecha, los canales de salida.

2.2.2 CORRELACIÓN CRUZADA

Habitualmente, las librerías para inteligencia artificial implementan la *correlación cruzada* bajo el nombre de convolución.

▶ **Definición 2.16.** Dadas dos funciones medibles $f, g : \mathbb{R}^n \longrightarrow \mathbb{R}$ su *correlación cruzada* $f \star g : R^n \longrightarrow \mathbb{R}$ es una función dada por

$$(f \star g)(\mathbf{x}) = \int_{\mathbb{R}^n} f(\mathbf{y}) g(\mathbf{x} + \mathbf{y})\, d\mathbf{y}$$

cuando la integral tenga sentido. A la función g es habitual referirse como *núcleo*.

Vamos a ver como, bajo un cambio de índices, la convolución y la correlación cruzada son la misma operación

Proposición 2.17. Dadas dos funciones medibles $f, g : \mathbb{R}^n \longrightarrow \mathbb{R}$ tales que su correlación cruzada $f \star g$ existe y sea $\tilde{g}(\mathbf{x}) = g(-\mathbf{x})$, entonces el producto de convolución $\tilde{f} * g$ existe y verifica que $f \star g = \tilde{f} * g$.

Demostración. Es suficiente con realizar el cambio de variable $\mathbf{z} = \mathbf{x} + \mathbf{y}$:

$$(f \star g)(\mathbf{x}) = \int_{\mathbb{R}^n} f(\mathbf{y}) g(\mathbf{x} + \mathbf{y})\, d\mathbf{y} = \int_{\mathbb{R}^n} f(\mathbf{z} - \mathbf{x}) g(\mathbf{z}) d\,\mathbf{z} = \int_{\mathbb{R}^n} \tilde{f}(\mathbf{x} - \mathbf{z}) g(\mathbf{z}) d\,\mathbf{z} = (f * g)(\mathbf{x})$$

□

Esta diferencia hace que sea habitual en la jerga de la inteligencia artificial referirse a la convolución como *convolución con inversión de núcleo* y a la correlación cruzada como *convolución sin inversión de núcleo*. Debido al cambio de signo, la correlación cruzada $f \star g$ de dos funciones no es, en general, conmutativa.

Para el caso discreto, también podemos dar una definición:

▶ **Definición 2.18.** Dadas dos funciones $f, g : \mathbb{Z}^n \longrightarrow \mathbb{R}$, definimos su producto de convolución $f \star g$ como

$$(f \star g)(i_1, \ldots, i_n) = \sum_{k_1=-\infty}^{\infty} \cdots \sum_{k_n=-\infty}^{\infty} f(k_1, \ldots, k_n) g(i_1 + k_1, \ldots, i_n + k_n)$$

cuando el sumatorio tenga sentido. A la función g es habitual referirse como núcleo.

Análogamente, mediante un cambio de índices, obtenemos la equivalencia entre correlación cruzada y convolución:

Proposición 2.19. Dadas dos funciones $f, g : \mathbb{Z}^n \longrightarrow \mathbb{R}$ tales que su correlación cruzada existe y $\tilde{f}(i_1, \ldots, i_n) = f(-i_1, \ldots, -i_n)$ entonces $(f \star g)(i_1, \ldots, i_n) = (\tilde{f} * g)(i_1, \ldots, i_n)$

Demostración. Análoga a la demostración anterior realizamos los cambios $z_1 = i_1 + k_1, \ldots, z_n = i_n + k_n$. □

Dada esta equivalencia no necesitamos desarrollar más la correlación cruzada, pues todo lo estudiado para la convolución se traslada aquí.

2.2.3 POOLING

Una operación de *pooling* funciona de manera muy similar a una convolución discreta, con una ventana que se desliza por el vector/matriz original, pero reemplazamos la combinación lineal por algún otro tipo de función como puede ser el promedio o el máximo. La función que situemos dará lugar al tipo de pooling que hagamos.

► **Definición 2.20.** Dado un vector $(x_1,\dots,x_n)^t=\mathbf{x}\in\mathbb{R}^n$ definimos la aplicación con núcleo $f:\mathbb{R}^m\longrightarrow\mathbb{R}$, paso $s\in\{1,2,\dots\}$ y relleno $p\in\{0,1,\dots...\}$,
$\mathrm{Pool}_{f,s,p}:\mathbb{R}^n\longrightarrow\mathbb{R}^{\lfloor\frac{n-m+1+2p}{s}\rfloor}$ como

$$\big(\mathrm{Pool}_{f,s}(\mathbf{x})\big)_i=f(x_{1+s(i-1)-p},\dots,x_{m+s(i-1)-p}),\qquad 1\le i\le\left\lfloor\frac{n-m+1}{s}\right\rfloor$$

donde $x_j=0$ para todo $j\le 0$ y todo $j>n$.

Cuando f sea la función $\max\{x_1,\dots,x_m\}$ hablaremos de *pooling máximo* y lo denotaremos $\mathrm{Poolmax}_{m,\,s,\,p}$ cuando f sea el promedio $\frac{1}{m}\sum_{i=1}^m x_i$ hablaremos de *pooling promedio* y lo denotaremos $\mathrm{AvgPool}_{m,\,s,\,p}$.

Ejemplo 2.21 (Poolmax de un vector)**.** Sea $\mathbf{x}=(5,9,1,2,3)^t$ vamos a calcular diversos poolings con la función $f(x_1,x_2,x_3)=\max\{x_1,x_2,x_3\}$. Si tomamos paso $s=1$

$$\mathrm{Poolmax}_{3,1,0}(\mathbf{x})=\big(\max\{5,9,1\},\ \max\{9,1,2\},\ \max\{1,2,3\}\big)^t=\big(9,9,3\big)^t.$$

Ejemplo 2.22 (Avgpool de un vector)**.** Sea $\mathbf{x}=(5,9,1,2,3)^t$ vamos a calcular diversos poolings con la función $f(x_1,x_2,x_3)=\frac{1}{3}(x_1+x_2+x_3)$. Si tomamos paso $s=1$

$$\mathrm{AvgPool}_{3,1,0}(\mathbf{x})=\big(\tfrac{1}{3}(5+9+1),\ \tfrac{1}{3}(9+1+2),\ \tfrac{1}{3}(1+2+3)\big)^t=\big(5,4,2\big)^t$$

Podemos extender sin dificultad la definición al caso de pooling en dos dimensiones

► **Notación 2.23.** Dada una matriz $A\in\mathbb{R}^{n\times m}$ y cuatro números $1\le n_1\le n_2\le n$, $1\le m_1\le m_2\le m$ vamos a denotar $(a_{i,j})_{n_1\le i\le n_2,m_1\le j\le m_2}$ a la submatriz de A correspondiente

$$A=\begin{pmatrix}a_{11}&\cdots&\cdots&a_{1m}\\ \vdots&\ddots&&\vdots\\ \vdots&&\ddots&\vdots\\ a_{n1}&\cdots&\cdots&a_{nm}\end{pmatrix}\quad\longrightarrow\quad(a_{i,j})_{\substack{n_1\le i\le n_2,\\ m_1\le j\le m_2}}=\begin{pmatrix}a_{n_1m_1}&\cdots&a_{n_1m_2}\\ \vdots&\ddots&\vdots\\ a_{n_2m_1}&\cdots&a_{n_2m_2}\end{pmatrix}$$

► **Definición 2.24.** Dada una matriz $A\in\mathbb{R}^{n_1\times n_2}$ definimos la aplicación con núcleo $f:\mathbb{R}^{m_1\times m_2}\longrightarrow\mathbb{R}$, paso s_1,s_2 y relleno p_1,p_2;
$\mathrm{Pool}_{f,s_1,s_2,p_1,p_2}:\mathbb{R}^{n_1\times n_2}\longrightarrow\mathbb{R}^{\lfloor\frac{n_1-m_1+1+2p_1}{s_1}\rfloor\times\lfloor\frac{n_2-m_2+1+2p_2}{s_2}\rfloor}$ como

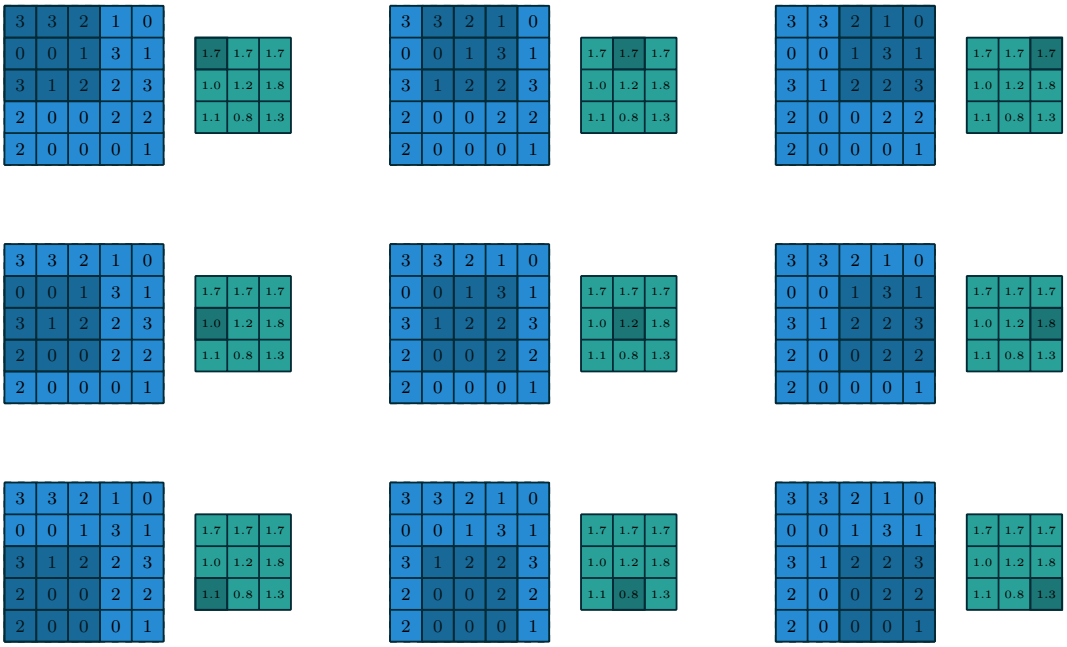
$$\big(\mathrm{Pool}_{f,s_1,s_2,p_1,p_2}(\mathbf{x})\big)_{ij}=f\left((a_{rk})_{\substack{1+s_1(i-1)-p_1\le r\le m_1+s_1(i-1)+p_1,\\ 1+s_2(j-1)-p_2\le k\le m_2+s_2(j-1)+p_2}}\right),$$
$$1\le i\le\left\lfloor\frac{n_1-m_1+1+2p_1}{s_1}\right\rfloor,\ 1\le j\le\left\lfloor\frac{n_2-m_2+1+2p_2}{s_2}\right\rfloor.$$

donde $x_{rk}=0$ siempre que se verifique cualquiera de las condiciones $r\le 0$, $r>n_1$,

$k \leq 0$, $k > n_2$.

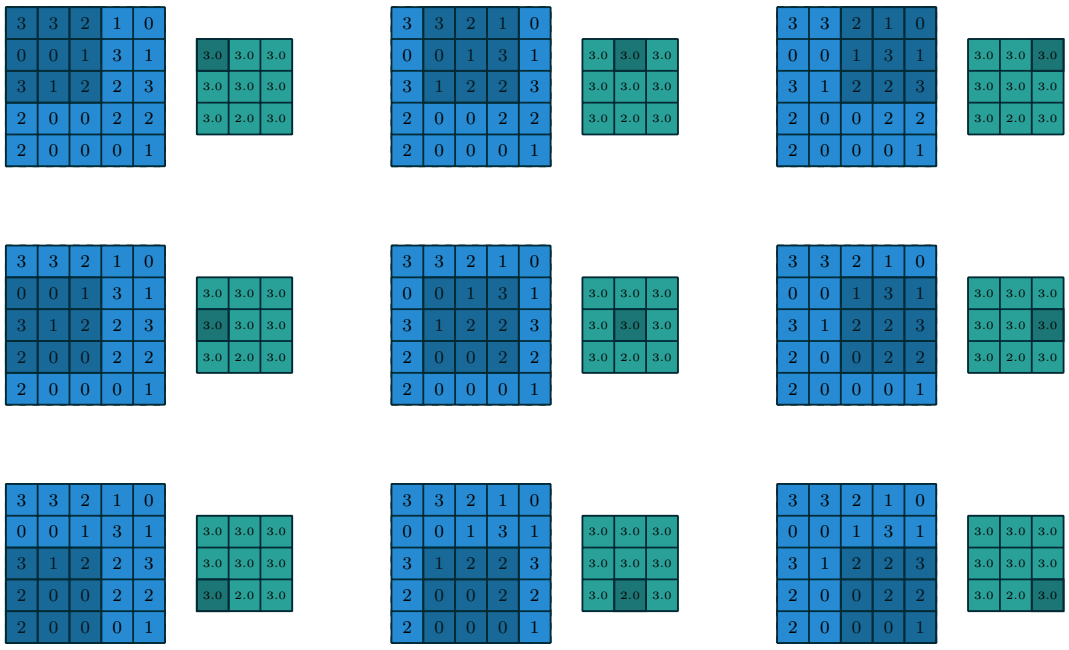
Cuando f sea la función máx$\{x_{11}, \ldots, x_{m_1,m_2}\}$ hablaremos de *pooling máximo* y lo denotaremos Poolmax$_{m_1,m_2,s_1,s_2,p_1,p_2}$ cuando f sea el promedio $\frac{1}{m_1 m_2} \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} x_{ij}$ hablaremos de *pooling promedio* y lo denotaremos AvgPool$_{m_1,m_2,s_1,s_2,p_1,p_2}$.

Ejemplo 2.25 (AvgPool de una matriz)**.** Partimos de una matrix $A \in \mathbb{R}^{5\times5}$ y aplicamos AvgPool de tamaño $m_1 = 3, m_2 = 3$ con paso 1 para obtener una nueva matrix 3×3:



Crédito: Dumoulin *et al.* (2016).

Ejemplo 2.26 (Poolmax de una matriz)**.** Partimos de una matrix $A \in \mathbb{R}^{5\times5}$ y aplicamos Poolmax de tamaño $m_1 = 3, m_2 = 3$ con paso 1 para obtener una nueva matrix 3×3:



Crédito: Dumoulin *et al.* (2016).

2.3 ENTRENAMIENTO

Como vimos en la sección 1.3, el problema del aprendizaje automático resulta en un problema de optimización para encontrar la regla ERM que minimiza el riego empírico. Es decir, una función $\hat{h} = \text{argmin}_{h\in\mathcal{H}}\ \mathcal{R}_S(h)$. En el caso particular de las redes neuronales, podemos fijar la *arquitectura de la red* fijando . De este modo, queda fijada la clase de funciones $\mathcal{H}$ y debemos encontrar los pesos de la red $\boldsymbol{\theta}$ que satisfagan

$$\hat{h} = \underset{\theta\in\Theta}{\text{argmin}}\ \mathcal{R}_S(h_{\boldsymbol{\theta}})$$

2.3.1 RETROPROPAGACIÓN

Vamos a tratar el problema empleando algoritmos de descenso del gradiente por lo que, en primer lugar, vamos a estudiar como efectuar el cálculo de dichos gradientes. Si partimos de una función $h_{\boldsymbol{\theta}}$ de la forma (2.1) y asumimos que la función de pérdida es de la forma $\ell(y, x, h) = \ell(y, h(x))$ la función a derivar es:

$$\mathcal{R}_S(h_{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, h_{\boldsymbol{\theta}}(x_i))$$

Para ayudarnos en el cálculo de las derivadas vamos a introducir las siguientes notacio-

nes:

▶ **Notación 2.27.** Vamos a emplear el símbolo $\odot$ para denotar el *producto de Hadamard* de matrices, esto es, el producto a elemento a elemento. Algunos ejemplos,

$$\begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix} \odot \begin{pmatrix} b_1 & \cdots & b_n \end{pmatrix} = \begin{pmatrix} a_1 b_1 & \cdots & a_n b_n \end{pmatrix} \qquad \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \odot \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ \vdots \\ a_m b_m \end{pmatrix}$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \odot \begin{pmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nm} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & \cdots & a_{1m}b_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1}a_{n1} & \cdots & a_{nm}b_{nm} \end{pmatrix}$$

▶ **Notación 2.28.** Dada una matriz $A = (a_{ij})_{1 \leq i \leq n,\ 1 \leq j \leq m}$ vamos a denotar por ∇_A la matriz (o vector fila/columna) de derivadas parciales

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \qquad \longrightarrow \qquad \nabla_A = \begin{pmatrix} \dfrac{\partial}{\partial a_{11}} & \cdots & \dfrac{\partial}{\partial a_{1m}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial}{\partial a_{n1}} & \cdots & \dfrac{\partial}{\partial a_{nm}} \end{pmatrix}$$

▶ **Notación 2.29.** Cuando estemos trabajando con una red neuronal (2.1) y expresemos $\mathbf{a}_l = \mathbf{F}_l(W_l a_{l-1})$, vamos a denotar $\mathbf{z}_l = W_l \mathbf{a}_l$ para cada $l = 1, \ldots, L$.

Observación 2.30. Vamos a asumir que ℓ y todas las funciones $\mathbf{F}_l$ son funciones diferenciables en todos los puntos de su dominio y que las expresiones analíticas para ellas y sus derivadas son conocidas.

También, vamos a asumir que todos los pesos $w_{ij}^{(l)}$ son independientes entre sí y ninguno está fijado a ningún valor. Es decir, que estamos trabajando con un perceptrón multicapa. Esta pérdida de generalidad se asume para evitar complicar innecesariamente los siguientes cálculos sin aportar mayor claridad sobre lo que se está haciendo. De esta forma, hemos dejado fuera a las capas convolucionales.

Vamos a usar la expresión recursiva (2.2) y derivarla para obtener las derivadas de $\mathcal{R}_S$. Suponemos que $\ell = \ell(y_1, \ldots, y_n, \hat{y}_1, \ldots, \hat{y}_n)$.

$$\frac{\partial \ell}{\partial w_{ij}^{(L)}}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{d_L} \frac{\partial \ell}{\partial \hat{y}_k}(\mathbf{a}_L) \frac{\partial a_k^{(L)}}{\partial w_{ij}^{(L)}}(\mathbf{a}_{L-1})$$

Necesitamos calcular $\frac{\partial a_k^{(L)}}{\partial w_{ij}^{(L)}}(\mathbf{a}_{L-1})$

$$\begin{aligned}
\frac{\partial a_k^{(L)}}{\partial w_{ij}^{(L)}}(\mathbf{a}_{L-1}) &= \frac{\partial}{\partial w_{ij}^{(L)}} F_k\Big(W_L \mathbf{a}_{L-1}\Big) \\
&= \sum_{r=1}^{d_L} \frac{\partial F_k^{(L)}}{\partial z_r}(\mathbf{z}_L) \frac{\partial z_r}{\partial w_{ij}^{(L)}}(\mathbf{a}_{L-1}) \\
&= \sum_{r=1}^{d_L} \frac{\partial F_k^{(L)}}{\partial z_r}(\mathbf{z}_L) \frac{\partial}{\partial w_{ij}^{(L)}} \left(\sum_{s=1}^{d_{L-1}} w_{rs}^{(L)} a_s^{(L-1)} \right) \\
&= \sum_{r=1}^{d_L} \frac{\partial F_k^{(L)}}{\partial z_r}(\mathbf{z}_L) \delta_{ir} a_j^{(L-1)} \\
&= \frac{\partial F_k^{(L)}}{\partial z_i}(\mathbf{z}_L)\, a_j^{(L-1)}
\end{aligned}$$

Finalmente,

$$\frac{\partial \ell}{\partial w_{ij}^{(L)}}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{d_L} \frac{\partial \ell}{\partial \hat{y}_k}(\mathbf{a}_L)\, \frac{\partial F_k^{(L)}}{\partial z_i}(\mathbf{z}_L)\; a_j^{(L-1)}$$

Usando la notación vectorial 2.27 y 2.28 podemos reescribir esta ecuación como

$$\nabla_{W_L} \ell(\mathbf{y}, \hat{\mathbf{y}}) = \left(\nabla_{\hat{\mathbf{y}}^t} \ell(\mathbf{a}_L) \cdot \frac{\partial \mathbf{F}_L}{\partial \mathbf{z}_L}(\mathbf{z}_L) \right)^t \odot \mathbf{a}_{L-1}^t \tag{2.5}$$

Procedemos a calcular las derivadas para la siguiente matriz parámetros

$$\begin{aligned}
\frac{\partial \ell}{\partial w_{ij}^{(L-1)}}(\mathbf{y}, \hat{\mathbf{y}}) &= \sum_{k=1}^{d_L} \frac{\partial \ell}{\partial \hat{y}_k}(\mathbf{a}_L) \frac{\partial a_k^{(L)}}{\partial w_{ij}^{(L-1)}}(\mathbf{a}_{L-1}) \\
&= \sum_{k=1}^{d_L} \frac{\partial \ell}{\partial \hat{y}_k}(\mathbf{a}_L) \left[\sum_{r=1}^{d_L} \frac{\partial F_k^{(L)}}{\partial z_r}(\mathbf{z}_L) \frac{\partial}{\partial w_{ij}^{(L-1)}} \left(\sum_{s=1}^{d_{L-1}} w_{rs}^{(L)} a_s^{(L-1)} \right) \right] \\
&= \sum_{k=1}^{d_L} \frac{\partial \ell}{\partial \hat{y}_k}(\mathbf{a}_L) \left[\sum_{r=1}^{d_L} \frac{\partial F_k^{(L)}}{\partial z_r}(\mathbf{z}_L) \left(\sum_{s=1}^{d_{L-1}} w_{rs}^{(L)} \frac{\partial a_s^{(L-1)}}{\partial w_{ij}^{(L-1)}} \right) \right] \\
&= \sum_{k=1}^{d_L} \frac{\partial \ell}{\partial \hat{y}_k}(\mathbf{a}_L) \left[\sum_{r=1}^{d_L} \frac{\partial F_k^{(L)}}{\partial z_r}(\mathbf{z}_L) \left(\sum_{s=1}^{d_{L-1}} w_{rs}^{(L)} \frac{\partial F_s^{(L-1)}}{\partial z_i}(\mathbf{z}_{L-1})\, a_j^{(L-2)} \right) \right]
\end{aligned}$$

Usando la notación vectorial obtenemos

$$\nabla_{W_{L-1}} \ell(\mathbf{y}, \hat{\mathbf{y}}) = \left(\left(\left(\nabla_{\hat{\mathbf{y}}^t} \ell(\mathbf{a}_L) \cdot \frac{\partial \mathbf{F}_L}{\partial \mathbf{z}_L}(\mathbf{z}_L) \right) \cdot W_L \right) \cdot \frac{\partial \mathbf{F}_{L-1}}{\partial \mathbf{z}_{L-1}}(\mathbf{z}_{L-1}) \right)^t \cdot \mathbf{a}_{L-2}^t \tag{2.6}$$

Si denominamos $\boldsymbol{\delta}_L = \nabla_{\hat{\mathbf{y}}^t} \ell(\mathbf{a}_L) \cdot \frac{\partial \mathbf{F}_L}{\partial \mathbf{z}_L}(\mathbf{z}_L)$ y $\boldsymbol{\delta}_{L-1} = (\boldsymbol{\delta}_L \cdot W_L) \cdot \frac{\partial \mathbf{F}_{L-1}}{\partial \mathbf{z}_{L-1}}(\mathbf{z}_{L-1})$ podemos escribir las derivadas (2.5) y (2.6) como

$$\begin{aligned}
\nabla_{W_L} \ell(\mathbf{y}, \hat{\mathbf{y}}) &= \boldsymbol{\delta}_L^t \cdot \mathbf{a}_{L-1}^t \\
\nabla_{W_{L-1}} \ell(\mathbf{y}, \hat{\mathbf{y}}) &= \boldsymbol{\delta}_{L-1}^t \cdot \mathbf{a}_{L-2}
\end{aligned}$$

la siguiente derivada $\nabla_{W_{L-2}}\ell(\mathbf{y},\hat{\mathbf{y}})$ la podemos calcular como

$$\nabla_{W_{L-2}}\ell(\mathbf{y},\hat{\mathbf{y}}) = \Big(\big(\boldsymbol{\delta}_{L-1}\cdot W_{L-1}\big)\cdot\frac{\partial \mathbf{F}_{L-2}}{\partial \mathbf{z}_{L-2}}(\mathbf{z}_{L-2})\Big)^t\cdot\mathbf{a}_{L-2}^t = \boldsymbol{\delta}_{L-2}^t\odot\mathbf{a}_{L-3}^t$$

donde hemos definido $\boldsymbol{\delta}_{L-2} = \big(\boldsymbol{\delta}_{L-1}\cdot W_{L-1}\big)\cdot\dfrac{\partial \mathbf{F}_{L-2}}{\partial \mathbf{z}_{L-2}}(\mathbf{z}_{L-2})$.

Aplicando recursividad vemos que, definiendo la sucesión de vectores fila (que avanza en orden inverso) $\{\boldsymbol{\delta}_l\}_{l=L}^{l=0}$ como

$$\boldsymbol{\delta}_{l-1} = \big(\boldsymbol{\delta}_l\cdot W_l\big)\cdot\frac{\partial \mathbf{F}_{l-1}}{\partial \mathbf{z}_{l-1}}(\mathbf{z}_{l-1}) \text{ para cada } l = L, L-1,\dots,1;\quad \boldsymbol{\delta}_L = \nabla_{\hat{\mathbf{y}}^t}\ell(\mathbf{y},\hat{\mathbf{y}})\cdot\frac{\partial \mathbf{F}_L}{\partial \mathbf{z}_L}(\mathbf{z}_L)$$

cualquiera de las derivadas $\nabla W_l\ell(\mathbf{y},\hat{\mathbf{y}})$ se calcula como

$$\nabla W_l\ell(\mathbf{y},\hat{\mathbf{y}}) = \boldsymbol{\delta}_l^t\cdot\mathbf{a}_l^t \text{ para cada } l = L, L-1,\dots,0. \tag{2.7}$$

Podemos evaluar las derivadas $\nabla_{W_l}\ell(\mathbf{y},\hat{\mathbf{y}})$ mediante la aplicación sucesiva de dos algoritmos

1. Propagación directa: calcular $\mathbf{z}_l$ y $\mathbf{a}_l$ (algoritmo 2.1)
2. Propagación inversa (retropropagación): calcular $\boldsymbol{\delta}_l$ y $\nabla_{W_l}\ell(\mathbf{y},\hat{\mathbf{y}})$(algoritmo 2.2)

Observación 2.31. En algunos contexto se incluye la propagación directa como parte de la retropropagación y se hace referencia al algoritmo de retropropagación como un todo.

Algoritmo 2.1: Propagación directa

Datos:
$\mathbf{x}$,
$W_1, W_2, \dots, W_L$.
Resultado:
$\mathbf{a}_0, \mathbf{a}_1 \dots, \mathbf{a}_L$;
$\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_L$.

```
a_0 ← x;
para l = 1, 2, ..., L hacer
    z_l ← W_l a_{l-1};
    a_l ← F_l(z_l);
fin
```

Observación 2.32. En el caso más habitual en que la función no lineal $\mathbf{F}(\mathbf{z})$ toma la forma

$$\mathbf{F}(\mathbf{z}) = \begin{pmatrix} f(z_1)\\ \vdots\\ f(z_{d_l})\end{pmatrix}$$

Algoritmo 2.2: Propagación inversa (retropropagación)

Datos:
$\mathbf{a}_0, \mathbf{a}_1 \ldots, \mathbf{a}_L$;
$\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_L$;
$W_0, W_1, \ldots, W_L$.
Resultado:
$\nabla_{W_L}\ell(\mathbf{y}, \hat{\mathbf{y}}), \nabla_{W_{L-1}}\ell(\mathbf{y}, \hat{\mathbf{y}}), \ldots, \nabla_{W_1}\ell(\mathbf{y}, \hat{\mathbf{y}})$

$\boldsymbol{\delta}_L \leftarrow \nabla_{\hat{\mathbf{y}}^t}\ell(\mathbf{y}, \hat{\mathbf{y}}) \cdot \dfrac{\partial \mathbf{F}_L}{\partial \mathbf{z}_L}(\mathbf{z}_L)$;
$\nabla_{W_L}\ell(\mathbf{y}, \hat{\mathbf{y}}) \leftarrow \boldsymbol{\delta}_L^t \cdot \mathbf{a}_L^t$;
para $l = L-1, L-2, \ldots, 1$ **hacer**
  $\boldsymbol{\delta}_{l-1} \leftarrow \left(\boldsymbol{\delta}_l \cdot W_l\right) \cdot \dfrac{\partial \mathbf{F}_{l-1}}{\partial \mathbf{z}_{l-1}}(\mathbf{z}_{l-1})$;
  $\nabla_{W_l}\ell(\mathbf{y}, \hat{\mathbf{y}}) \leftarrow \boldsymbol{\delta}_l^t \cdot \mathbf{a}_l^t$;
fin

la matriz de derivadas

$$\frac{\partial \mathbf{F}_l}{\partial \mathbf{z}} = \begin{pmatrix} f'(z_1) & 0 & \cdots & 0 \\ 0 & f'(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'(z_{d_l}) \end{pmatrix}$$

se vuelve diagonal y el número de operaciones a realizar se reduce, en el algoritmo 2.2 (retropropagación) puede sustituirse el paso $\boldsymbol{\delta}_{l-1} \leftarrow \left(\boldsymbol{\delta}_l \cdot W_l\right) \cdot \dfrac{\partial \mathbf{F}_{l-1}}{\partial \mathbf{z}_{l-1}}(\mathbf{z}_{l-1})$ por el paso

$$\boldsymbol{\delta}_{l-1} \leftarrow \left[\boldsymbol{\delta}_l \odot \operatorname{diag}\left(\frac{\partial \mathbf{F}_{l-1}}{\partial \mathbf{z}_{l-1}}(\mathbf{z}_{l-1})\right)^t\right] \cdot W_l$$

que reduce la multiplicación de dos matrices ($(d_l)^2$ multiplicaciones de vectores elemento a elemento) a multiplicar dos vectores elemento a elemento.

El número de operaciones es elevado dado que contempla la multiplicación de un elevado número de matrices hasta para las redes neuronales más pequeñas. Para ilustrar el número de operaciones necesarias vamos a calcular el número de operaciones `fila` $\times$ `columna` y productos elemento a elemento de vectores. Empleamos la versión más habitual con la diagonal.

- Para la propagación directa
  - $d_0 + d_1 + \cdots + d_{L-1}$ productos `fila` $\times$ `columna`
- Para la retropropagación
  - $L + 1$ productos de vectores
  - $d_0 + d_1 + \cdots + d_{L-1}$ productos `fila` $\times$ `columna`

Vemos como el número de productos elemento a elemento crece linealmente con el número de capas $\mathcal{O}(L)$ y cuadráticamente con la dimensión de las capas $\mathcal{O}\big((d_l)^2\big)$.

Si pensamos en que una red pequeña con 5 capas para procesar imágenes a baja resolución (200 $\times$ 200 píxeles) tiene una capa de entrada con dimensión $d_0 = 40\,000$, nos hacemos a la idea del altísimo coste computacional de este entrenamiento. Para poder realizarse en tiempos de computación razonables es necesaria la paralización masiva de estas operaciones en hardware dedicado.

2.3.2 DESCENSO DE GRADIENTE (GD)

Ahora que sabemos calcular el gradiente con respecto a todos los parámetros, podemos emplear el gradiente para buscar mínimos globales en la función objetivo $\mathcal{R}_S$. Recordemos que dado un conjunto de entrenamiento $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ el riesgo empírico se define como

$$\mathcal{R}_S = \sum_{i=1}^{n} \ell\big(\mathbf{y}_i, h_{\boldsymbol{\theta}}(\mathbf{x}_i)\big).$$

donde ℓ es la función de pérdida y $h_{\boldsymbol{\theta}}$ una red neuronal. Podemos tratar $\mathcal{R}_S$ como una función de los parámetros $\mathbf{w} = (w_{11}^{(1)}, \ldots, w_{d_L d_L}^{(L)})$ de la red.

Aunque en la sección precedente no hemos visto como calcular el gradiente cuando tratemos con capas convolucionales, este gradiente puede calcularse de una manera similar y vamos a tomar el gradiente $\nabla_{\mathbf{w}} \mathcal{R}_S(\mathbf{w})$ como conocido.

En condiciones de suficiente regularidad, como el gradiente de $\mathcal{R}_S(\mathbf{w})$ es la dirección de máximo crecimiento, si avanzamos una determinada cantidad γ en la dirección opuesta al gradiente, nos acercaremos a un mínimo local de la función. Así, nace un proceso iterativo

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \nabla \mathcal{R}_S(\mathbf{w}_t), \qquad t \geq 1.$$

La amplitud de paso $\gamma_t \in \mathbb{R}$ se denomina *ratio de aprendizaje* y cada una de las iteraciones *época de entrenamiento* o *paso de entrenamiento*[1]. El gradiente puede calcularse mediante la aplicación sucesiva de los algoritmos 2.1 (propagación directa) y 2.2 (retropropagación) tal y como se explica en la sección 2.3.1, pues $\nabla \mathcal{R}_S(\mathbf{w}) = \sum_{i=1}^{n} \nabla \ell\big(\mathbf{y}_i, h_{\boldsymbol{\theta}}(\mathbf{x}_i)\big)$.

Podemos resumir el algoritmo como:

Algoritmo 2.3: Descenso del gradiente (GD)

Datos:
$\gamma_0, \ldots, \gamma_{\tau-1}$;
$\mathbf{w}_0$ `// Inicialización aleatoria`;
Resultado:
$\mathbf{w}$

para $0 \geq t \geq \tau - 1$ **hacer**
  `Calcular` $\nabla \mathcal{R}_S(\mathbf{w}_t)$;
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \gamma_t \nabla \mathcal{R}_S(\mathbf{w}_t)$;
fin

[1] Estas dos denominaciones que coinciden en este algortimo, dejarán de significar lo mismo en los algoritmos siguientes.

Observación 2.33. En lo que precede hemos asumido que $h_{\boldsymbol{\theta}} : \mathbb{R}^{d_0} \longrightarrow \mathbb{R}^{d_L}$ es una función diferenciable en todo su dominio. Sin embargo, de manera general emplearemos funciones de activación que no son diferenciables en algún punto y, por tanto, $h_{\boldsymbol{\theta}}$ no será diferenciable en todo su dominio. Esta dificultad se soluciona si sustituimos los gradientes por *subgradientes*. El método de los subgradientes no se estudia en este trabajo. Puede encontrar más información en Shalev-Shwartz *et al.* (2014, p. 188–191).

Condiciones de convergencia

Un estudio detallado de las condiciones de convergencia excede el alcance de este trabajo. Se presentan algunas condiciones para problemas de aprendizaje convexos. Nótese, que estas garantías teóricas no son de aplicación a las redes neuronales expuestas en este capítulo. Sin embargo, se comprueba experimentalmente que las técnicas de descenso del gradiente aquí explicadas sí suelen converger en la práctica.

▶ **Definición 2.34.** Sea $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ una función continua y diferenciable, decimos que es β–*Lipschitz* si verfica

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leq \beta \|\mathbf{x} - \mathbf{y}\| \text{ para todo } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n.$$

▶ **Definición 2.35.** Sea $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ una función continua y diferenciable, decimos que es β–*suave* si su gradiente ∇f es β–Lipschitz, esto es,

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq \beta \|\mathbf{x} - \mathbf{y}\| \text{ para todo } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n.$$

Teorema 2.36. Sea $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ una función convexa y β–suave con mínimo absoluto $\mathbf{w}^*$. Si partimos de un punto $\mathbf{w}_0 \in R^n$ y aplicamos τ iteraciones del algoritmo de descenso del gradiente con ratio de aprendizaje $\gamma = \frac{1}{\beta}$ para obtener $\mathbf{w}_\tau \in \mathbb{R}^n$, entonces se cumple que

$$f(\mathbf{w}_\tau) - f(\mathbf{w}^*) \leq \frac{2\beta \|\mathbf{w}_\tau - \mathbf{w}^*\|}{\tau}.$$

Demostración. La demostración se omite y puede encontrarse en Bubeck (2015, Th 3.3, p.267) □

Veamos ahora como podemos mejorar este resultado bajo hipótesis más restrictivas.

▶ **Definición 2.37.** Se dice que una función $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ es α–*fuertemente convexa* si la función $g : \mathbb{R}^n \longrightarrow \mathbb{R}$ definida como

$$g(\mathbf{x}) = f(\mathbf{x}) - \frac{\alpha}{2} \|\mathbf{x}\|^2$$

es convexa.

Proposición 2.38. Sea $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ una función α–fuertemente convexa, entonces f es convexa.

Demostración. Por ser f α–fuertemente convexa, la función $g(\mathbf{x}) = f(\mathbf{x}) - \frac{\alpha}{2} \|\mathbf{x}\|^2$ es convexa. Por tanto, $f(\mathbf{x}) = g(\mathbf{x}) + \frac{\alpha}{2} \|\mathbf{x}\|^2$ es convexa por ser suma de funciones convexas. □

Teorema 2.39. Sea $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ una función α–fuertemente convexa, β–suave y con mínimo absoluto en $\mathbf{w}^*$; entonces el algoritmo de descenso del gradiente con ratio de aprendizaje $\eta = \frac{2}{\alpha+\beta}$, valor inicial $\mathbf{x}_0$ y valor final $\mathbf{x}_\tau$ (tras τ interacciones) verifica que

$$f(\mathbf{w}_\tau) - f(\mathbf{w}^*) \leq \frac{\beta}{2} \exp\left(\frac{-4\tau}{\frac{\beta}{\alpha}+1}\right) \|\mathbf{w}_0 - \mathbf{w}^*\|^2$$

Demostración. La demostración se omite y puede consultarse en Bubeck (2015, Th 3.12, p.279) □

En este caso, vemos como la adicción de la fuerte convexidad como hipótesis hace que el teorema 2.39 pueda convertir en exponencial la convergencia del teorema 2.36. Con estas condiciones más restrictivas es dónde más desarrollada se encuentra la teoría.

2.3.3 DESCENSO ESTOCÁSTICO DEL GRADIENTE (SGD)

El algoritmo de descenso del gradiente supone evaluar el gradiente $\nabla_{\mathbf{w}}\ell(\mathbf{y}, \hat{\mathbf{y}})$ en cada iteración para todos los ejemplos $(\mathbf{x}_i, \mathbf{y}_i)$ del conjunto de entrenamiento S. Una única evaluación de este gradiente es computacionalmente muy costosa. Si el tamaño del conjunto de entrenamiento es del orden de los miles de ejemplos[2], el cálculo de $\nabla_{\mathbf{w}}\ell(\mathbf{y}, \hat{\mathbf{y}})$ es demasiado costoso para ser llevado acabo. Estaríamos calculando derivadas con respecto a millones de parámetros para miles de imágenes y queremos repetir este procedimiento miles de veces. Es esencial aproximar este gradiente sin incurrir en un coste computacional tan elevado y sin recurrir a reducir el número total de ejemplos sobre el que entrenamos.

Se puede demostrar (véase por ejemplo Bubeck (2015, p.329–338)) que bajo ciertas condiciones es suficiente con que el gradiente calculado sea el correcto en promedio. Es decir, basta obtener un estimador insesgado de $\nabla\mathcal{R}_S$. Para ello, en lugar de calcular $\nabla\mathcal{R}_S$ usando todo el conjunto de entrenamiento, en cada iteración del algoritmo muestrearemos un pequeño *lote*[3] $S^* \subset S$ de forma aleatoria sin repetición ni sustitución y calcularemos el gradiente usando solo los ejemplos de este lote. Cuando hayamos agotado el muestreo de todo S, lo volveremos a muestrear. El tamaño de estos lotes suele rondar la decena de ejemplos.

Como la diferencia entre el tamaño de S y S^* es grande, la diferencia entre muestrear S sin repetición (como lo hacemos) y con repetición (lo que daría una *i.i.d.* sobre $\mathcal{D}$ es despreciable). Seguimos obteniendo un estimador insesgado de $\nabla\mathcal{R}_D$ a la vez que usamos todo el conjunto de entrenamiento.

El algoritmo modificado se resume como:

[2]Los tamaños habituales suelen rondar 10^4–10^6.
[3]Referidos como *batches* en la literatura inglesa

Algoritmo 2.4: Descenso Estocástico de Gradiente (SDG)

Datos:
$\gamma_0, \ldots, \gamma_{\tau-1}$;
$\mathbf{w}_0$ `// Inicialización aleatoria`;
Resultado:
$\mathbf{w}$

para $0 \geq t \geq \tau - 1$ **hacer**
  `Elige muestra aleatoria` $S^* \subset S$;
  `Calcula` $\nabla \mathcal{R}_{S^*}(\mathbf{w}_t)$;
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \gamma_t \nabla \mathcal{R}_{S^*}(\mathbf{w}_t)$;
fin

▶ **Definición 2.40.** A cada iteración del algoritmo 2.4 la llamaremos *paso de entrenamiento*. Cuando hayamos muestreado todas las imágenes de S, diremos que hemos completado una *época de entrenamiento* y volveremos a muestrear S de nuevo.

La definición dada de época de entrenamiento ahora y la dada para el descenso de gradiente coinciden. La diferencia es que en el descenso del gradiente observamos todo S en un único paso de entrenamiento y en el descenso del gradiente damos muchos pasos hasta completar S.

Condiciones de convergencia

El siguiente teorema que presentamos es el equivalente del teorema 2.39 para el algoritmo que nos ocupa.

Teorema 2.41. Sea $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ una función α–fuertemente convexa con mínimo absoluto $\mathbf{w}^* \in \mathbb{R}^n$. Si $\mathbb{E} \|\nabla \mathcal{R}_{S^*}(\mathbf{w})\| \leq B$, entonces el algoritmo de descenso del gradiente estocástico con ratio de aprendizaje $\eta_t = \frac{2}{\alpha(t+1)}$ verifica

$$f\left(\sum_{i=1}^{\tau} \frac{2i}{\tau(\tau+1)} \mathbf{w}_i\right) - f(\mathbf{w}^*) \leq \frac{2B^2}{\alpha(\tau+1)}.$$

Demostración. La demostración se omite y puede consultarse en Bubeck (2015, Th 6.2, p.331). □

Vemos que la convergencia exponencial que garantizaba el teorema 2.39 se reduce a una convergencia $\sim \frac{1}{\tau}$ en este caso. Solo podemos garantizar la convergencia en promedio.

2.3.4 DESCENSO ESTOCÁSTICO DEL GRADIENTE CON MOMENTO

El descenso del gradiente estocástico produce una ratio de aprendizaje lento. Para acelerar este ratio de aprendizaje Polyak (1964) introduce el método del momento[4].

La actualización consiste en introducir un hiperparámetro $\alpha \in [0, 1)$ que tiene en cuenta la contribución de los gradientes anteriores. Nos referimos por hiperparámetro a aquellos

[4]El nombre se obtiene por analogía con el momento lineal en física.

valores que fijamos arbitrariamente y no forman parte de los pesos de la red. De esta forma, la actualización recursiva de los pesos $\mathbf{w}$ se sustituye por

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \nabla_{\mathbf{w}} \mathcal{R}_{S^*}(\mathbf{w}_t) + \alpha \mathbf{v}_t$$
$$\mathbf{v}_{t+1} = \alpha \mathbf{v}_t - \gamma_t \nabla_{\mathbf{w}} \mathcal{R}_{S^*}(\mathbf{w}_t)$$

Para valores más grandes de α respecto de γ, más afectan los gradientes previos a la dirección de avance. Valores comunes de α son 0.5, 0.9, y 0.99. Al igual que es común disminuir el ratio de aprendizaje con el número de iteraciones, también es común ir aumentando el valor de α.

El algoritmo puede entonces resumirse como:

Algoritmo 2.5: Descenso estocástico del gradiente con momento.

Datos:
$\gamma_0, \ldots, \gamma_{\tau-1}$;
$\alpha_0, \ldots, \alpha_{\tau-1}$;
$\mathbf{w}_0$ `// Inicialización aleatoria`;
Resultado:
$\mathbf{w}$

$\mathbf{v}_0 \leftarrow 0$;
para $0 \geq t \geq \tau - 1$ **hacer**
  `Elige muestra aleatoria` $S^* \subset S$;
  `Calcula` $\nabla \mathcal{R}_{S^*}(\mathbf{w}_t)$;
  $\mathbf{v}_{t+1} \leftarrow \alpha_t \mathbf{v}_t - \gamma_t \nabla_{\mathbf{w}} \mathcal{R}_{S^*}(\mathbf{w})$;
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{v}_{t+1}$;
fin

2.4 ALGUNOS EJEMPLOS DE REDES NEURONALES CONVOLUCIONALES

El precursor de las arquitecturas de redes neuronales modernas seguramente fuese Fukushima (1980) cuando introduce en la práctica algunos conceptos usados hoy en día como el extractor de atributos (*feauture extractor*) o la idea de conexiones compartidas (en la línea de la convolución). Fukushima trabajó en el reconocimiento de caracteres y su trabajo está claramente inspirado en los sistemas nerviosos de distintos animales. Estas ideas las introdujo para conseguir que la respuesta de la red fuera invariante bajo desplazamientos de los caracteres de entrada sobre su versión anterior de la red (Fukushima, 1975).

El primer caso rotundamente exitoso en el empleo de Redes Neuronales Convolucionales como las conocemos hoy en día y entrenadas mediante algoritmos de descenso del gradiente fue el de Lecun *et al.* (1998) con su modelo LeNet-5, ilustrado en la figura 2.8.

Lecun, al igual que Fukushima, trabajó en el reconocimiento óptico de caracteres consiguiendo tasas de error de 0.95 %. Además, introdujo técnicas de aumentación de datos mediante translaciones, rotaciones y deformaciones; como se muestra en la figura 2.9.

En el año 2010, se crea el primer desafío ImageNet: Desafío para la clasificación de imágenes a gran escala (Russakovsky *et al.*, 2015). Esto supuso un gran impulso a la

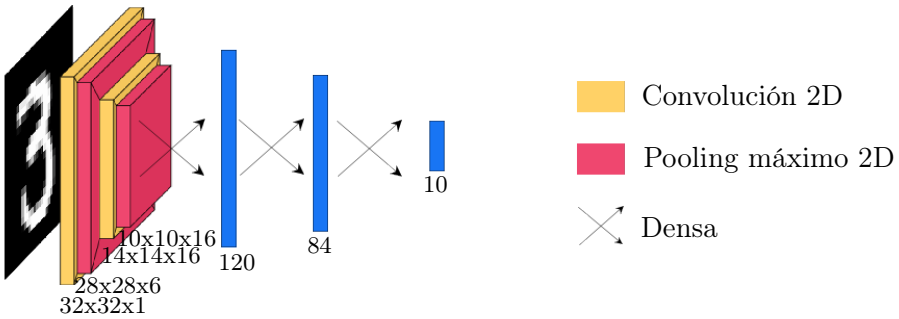


Figura 2.8: Arquitectura de la red LeNet-5.

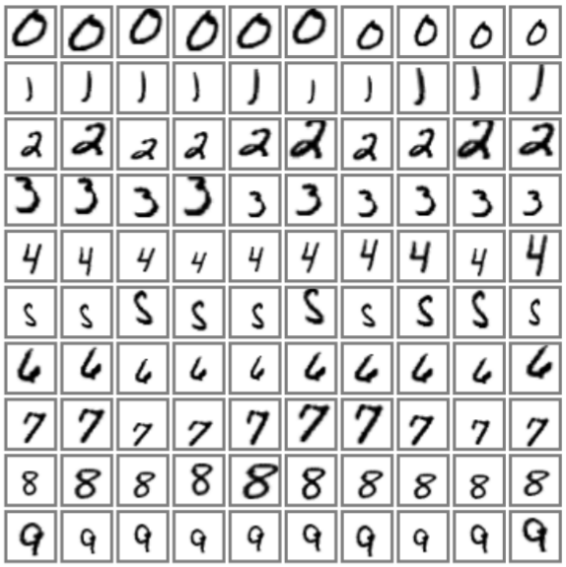


Figura 2.9: Algunos ejemplos de aumentación de datos mediante translaciones, rotaciones y deformaciones llevada a cabo por Lecun *et al.* (1998).

investigación en Visión Artificial. De la mano de este dataset, en 2012 llegó otro gran salto con AlexNet (Krizhevsky *et al.*, 2012), ilustrado en la figura figura 2.10, que ganó la competición ImageNet de 2010 con una tasa de error del 37 %.

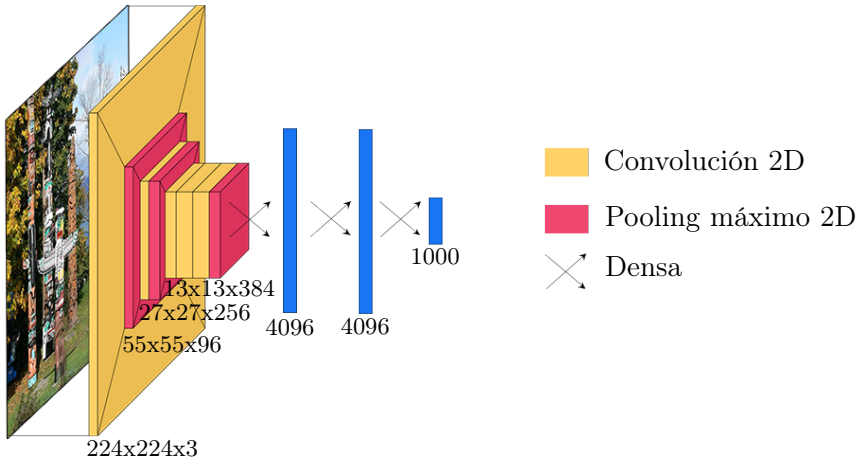


Figura 2.10: Arquitectura de la red AlexNet.

En el año 2014, la arquitectura VGG16[5] (Simonyan *et al.*, 2014), ilustrada en la figura 2.11, demostró que se puede aumentar notablemente la exactitud de la red si se aumenta el número de capas convoluciones. Entrenaron su red sobre el dataset del desafío ImageNet de 2014. El modelo cuenta con un total de 16 capas convolucionales y 3 capas densas al final. Simonyan *et al.* (2014) también presentaron el modelo VGG19, con 19 capas convolucionales, comprobando que, si bien obtuvieron mejores resultados, la exactitud del modelo comenzaba a saturar.

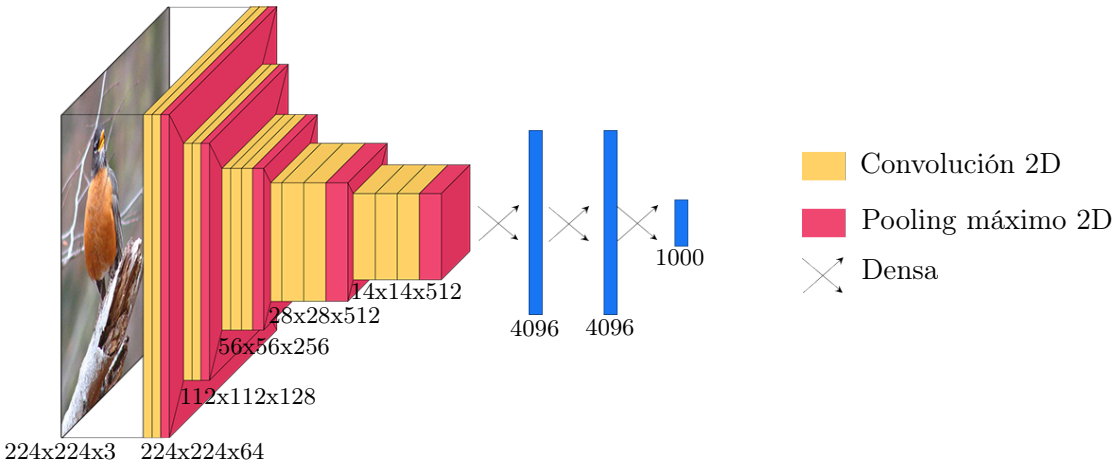


Figura 2.11: Arquitectura de la red VGG16.

La tabla 2.1 muestra una comparación de los modelos que hemos visto.

[5]VGG significa *Visual Geometry Group*, un grupo de investigación de la universidad de Oxford

	LeNet-5	**AlexNet**	**VGG16**
	1998	2012	2014
Tamaño entrada	28×28	224×224	224×224
Nº capas conv.	2	5	16
Nº capas densas	2	3	3
Nº parámetros	431k	61M	138M

Tabla 2.1: Comparativa de diferentes modelos de Redes Neuronales Convolucionales para clasificación de imágenes. Crédito: Alom *et al.* (2018).

Capítulo 3

DETECCIÓN DE OBJETOS CON REDES NEURONALES

Hasta ahora hemos tratado con redes orientadas a resolver el problema de clasificación. En 2012, durante el workshop de ImageNet, ya se planteaba la pregunta de hasta qué punto se pueden generalizar estos modelos para resolver el problema de detección (Girshick *et al.*, 2014, p.1). Enfocado en esta pregunta, en 2014 llega el modelo R-CNN Girshick *et al.* (2014). Este modelo aborda el problema en dos etapas diferentes: una primera etapa propone regiones candidatas a contener algún elemento a identificar y una segunda etapa clasifica cada una de las regiones propuestas. Así, se emplean dos redes neuronales, una en cada etapa, como ilustra la figura 3.1. Esto resulta muy costoso computacionalmente.

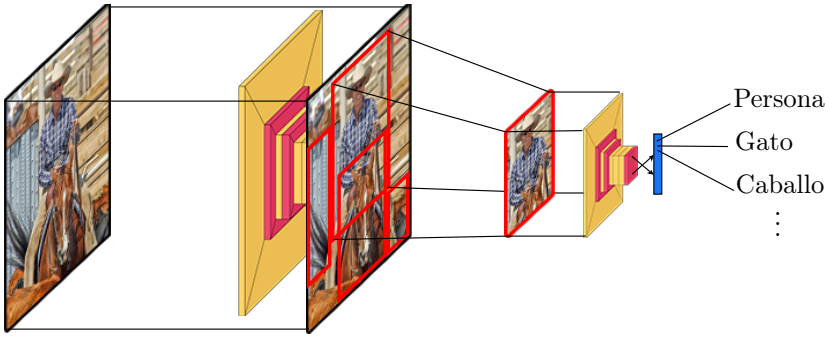


Figura 3.1: Esquema de funcionamiento de R-CNN.

En los años posteriores, aparecieron mejoras sobre este tipo de modelos como Fast R-CNN (Girshick, 2015), enfocados en ser capaces de hacer predicciones más rápidas pero sin ser capaces de llegar al tiempo real. Estas propuestas seguían procesando la imagen con dos redes neuronales, manteniendo un alto coste computacional.

Para reducir el tiempo de inferencia de los modelos, en 2015 surge YOLO (You Only Look Once) (Redmon; Divvala *et al.*, 2016)[1]. La ventaja clave de este algoritmo consiste en procesar la imagen con una única red neuronal, de ahí su nombre.

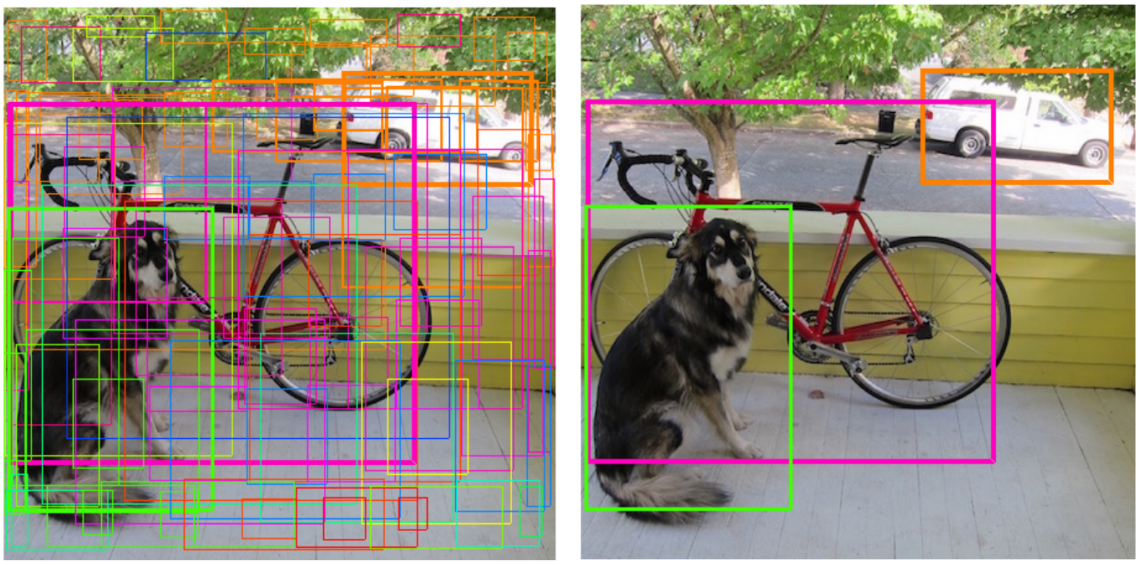
[1]Como curiosidad, de acuerdo al propio autor, el artículo fue rechazado para su presentación en

3.1 YOLO (YOU ONLY LOOK ONCE)

La bibliografía principal de esta sección son el artículo original (Redmon; Divvala *et al.*, 2016) y la presentación del artículo (Redmon, 2016) en CVPR 2016. Una primera pregunta fundamental es cómo vamos a hacer que el algoritmo prediga un número variable de objetos en una imagen cuando la salida de una red neuronal es siempre la misma. Por ello, la predicción va a constar de dos partes diferenciadas:

1. Generador de candidatos a detección: Red Neuronal Convolucional (figura 3.2a), y
2. Algoritmo para filtrar candidatos: Non Maximum Suppresion (NMS) (figura 3.2b).

Con este procedimiento, solo la primera parte debe procesar la imagen con una red neuronal, que es la parte más costosa computacionalmente. Así, generaremos una cantidad fija de candidatos mucho mayor del número de objetos que la imagen pueda contener y, después, nos quedaremos únicamente con aquellos candidatos que representen un objeto en la imagen.



(a) Todas las cajas predichas por la red. El color representa la clase y el grosor la puntuación.

(b) Resultado de la inferencia después de filtrar las predicciones y aplicar NMS.

Figura 3.2: Fases de la inferencia mediante el algortimo YOLO. (a) Generación de candidatos a detecciones con una CNN. (b) Filtrado de candidatos con el algoritmo NMS. Crédito Redmon (2016).

Observación 3.1. A lo largo de esta sección, vamos a hablar de *probabilidades inferidas por la red* o, simplemente, *probabilidades*. No hay ningún modelo probabilístico detrás de estos valores. Se trata de un abuso de lenguaje que pretende hacer más intuitivo el funcionamiento del algoritmo.

NeurIPS 2015 (Conferencia sobre sistemas de procesamiento de Información Neuronal) siendo calificado como «*borderline paper*» (Redmon, [s.f.]). Fue aceptado más tarde en el CVPR 2016 (Congreso sobre el reconocimiento de patrones y la visión por ordenador) y el público le otorgo el premio *People's Award* de la OpenCV University, que no es una universidad sino una plataforma de cursos online.

▶ **Definición 3.2.** Un *objeto* es un elemento reconocible en una imagen. Está definido por un *rectángulo delimitador* o *caja* y una categoría c_i de un conjunto de categorías $\mathscr{C} = \{c_1, \ldots, c_C\}$. Un *rectángulo delimitador* es una 4-upla $(a, b, w, h) \in \mathbb{R}^4$ donde a y b representan las coordenadas del vértice superior izquierdo[2] y w y h son el ancho y alto absolutos respectivamente del rectángulo.

▶ **Definición 3.3.** Dados dos rectángulos delimitadores R_1 y R_2 definimos su *intersección sobre la unión* que se abrevia $\mathrm{IoU}(R_1, R_2) \in [0, 1]$ como el cociente del área de su intersección entre el área de su unión. Es decir,

$$\mathrm{IoU}(R_1, R_2) = \frac{\text{Área } (R_1 \cap R_2)}{\text{Área } (R_1 \cup R_2)}.$$

El conjunto de atributos para nuestro modelo son las imágenes RGB de 8 bits con una resolución de 448×448 píxeles, es decir, $\mathcal{X} = \{0, 1, \ldots, 2^8 - 1\}^{448 \times 448 \times 3}$. La etiqueta de una imagen es un conjunto de objetos. Todos los posibles objetos están formados por $\mathcal{O} = \mathbb{R}^2 \times [0, \infty)^2 \times \mathscr{C}$. El conjunto de etiquetas es $\mathcal{Y} = \mathcal{P}\big(\mathcal{O}\big)$, es decir, el conjunto de partes de $\mathcal{O}$.

3.1.1 RED NEURONAL CONVOLUCIONAL

La red neuronal constituye un modelo de aprendizaje supervisado en sí misma. Primero describiremos el conjunto de atributos y de etiquetas de la red para entender los conceptos fundamentales detrás del funcionamiento del algoritmo. Después, veremos la arquitectura del modelo. Finalmente, veremos cómo se realiza el entrenamiento, junto con la función de perdida que nos permitirá obtener las etiquetas descritas.

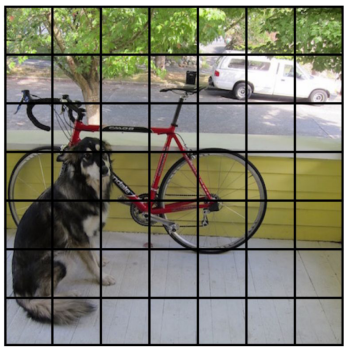
Atributos y etiquetas

La entrada de la red serán las imágenes RGB directamente. Por lo tanto, el conjunto de atributos $\mathcal{X}^*$ que tomará la red es $\mathcal{X}^* = \mathcal{X} = \{0, 1, \ldots, 2^8 - 1\}^{448 \times 448 \times 3}$. En cuanto al conjunto de etiquetas $\mathcal{Y}^*$, necesitamos codificar la información de los objetos de manera que podamos entrenar a la red para generar esta representación. Como la salida de la red es fija, esta representación debe incluir más objetos que los que pueda haber en una imagen y debe incluir algún tipo de información que nos permita filtrar los objetos que no son de interés. Vamos a describir el significado que va a tener cada uno de los términos de estas etiquetas $\mathbf{y}^* \in \mathcal{Y}^*$ sin importarnos cómo se relacionan con las etiquetas del modelo global $\mathbf{y} \in \mathcal{Y}$, de las que realmente disponemos. Trataremos esta relación cuando hablemos del entrenamiento.

La idea fundamental del artículo consiste en dividir la imagen de entrada en una rejilla regular $S \times S$, $S \in \mathbb{N}$ (véase figura 3.3a). Esta rejilla hace referencia a la salida y no a que vayamos a alimentar el modelo con porciones de la imagen. Cada una de las celdas de la rejilla se encargará de:

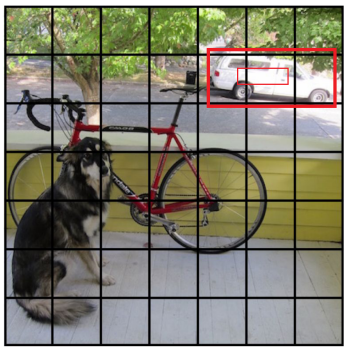
- Clasificar un objeto en caso de existir. La probabilidad de que el objeto pertenezca a una clase $\{c_1, \ldots, c_C\}$ determinada condicionado a la probabilidad de existir un objeto en la celda $\mathbb{P}(c_k|\mathrm{Obj})$ para cada $k = 1, \ldots, C$ (véase figura 3.3b).

[2] Como es habitual en informática, se entiende que la esquina superior izquierda de la imagen representa la coordenada $(0, 0)$, el eje x crece hacia la derecha y el eje y hacia abajo (siempre son positivas).



(a) División de la imagen en una rejilla de $S \times S$, $S = 7$ celdas.

(b) Clasificación de cada celda en una categoría.



(c) $B = 2$ cajas generadas por una celda para la predicción. El grosor de la línea representa la puntuación P_i para esa caja.

Figura 3.3: Funcionamiento de la red YOLO. Crédito Redmon (2016).

- Generar B rectángulos delimitadores (a, b, w, h) donde $a, b \in [0, 1]$ son las coordenadas relativas a la celda del centro de la caja, $w, h \in (0, \infty)$ son las coordenadas relativas al ancho de la celda para el ancho y alto de la caja[3]. Además, cada caja obtiene una *puntuación* definida como $P = \mathbb{P}(\text{obj}) \cdot \text{IoU}(\text{pred.}, \text{real})$. (véase figura 3.3c). Si una celda no tiene un objeto, aún así predecimos cajas a partir de ellas y lo importante es que $P \approx 0$.

De esta forma, para cada celda queremos obtener un vector

$$\Big(P_1, a_1, b_1, w_1, h_1; \ldots; P_B, a_B, b_B, w_B, h_B; \mathbb{P}(c_1|\text{Obj}), \ldots, \mathbb{P}(c_C|\text{Obj})\Big) \tag{3.1}$$

donde se ha enfatizado con punto y coma cada uno de los bloques de la predicción. Así la salida de la red tendrá un tamaño $(S, S, 5B + C)$. Redmon; Divvala *et al.* (2016) usan un total de $C = 20$ categorías, $B = 2$ cajas y $S = 7$ celdas, la salida de nuestra red será un tensor de dimensiones $(7, 7, 30)$. La figura 3.4 muestra un esquema de cómo es esta salida junto con las 98 cajas inferidas. De esta forma, las etiquetas a predecir por la red son $\mathbf{y}^* \in \mathcal{Y}^* = \mathbb{R}^{S \times S \times (5B+C)}$.

La limitación reside en que en cada imagen solo puede haber un máximo de $B \cdot S^2$ predicciones (en este caso 98) y que en cada celda solo puede haber B objetos y estos deben de pertenecer a la misma categoría. Así, hemos cumplido el requisito de generar más objetos de los que hay en una imagen y podremos, más adelante, usar el valor $P = \mathbb{P}(\text{obj}) \cdot \text{IoU}(\text{pred.}, \text{real})$ como referencia de que un objeto inferido corresponda o no a una predicción de interés.

[3] Abusando de notación hemos denotado estas coordenadas relativas igual que las coordenadas absolutas antes descritas. Es sencillo cambiar entre un y otro sistema de referencia. No haremos más menciones a este respecto.

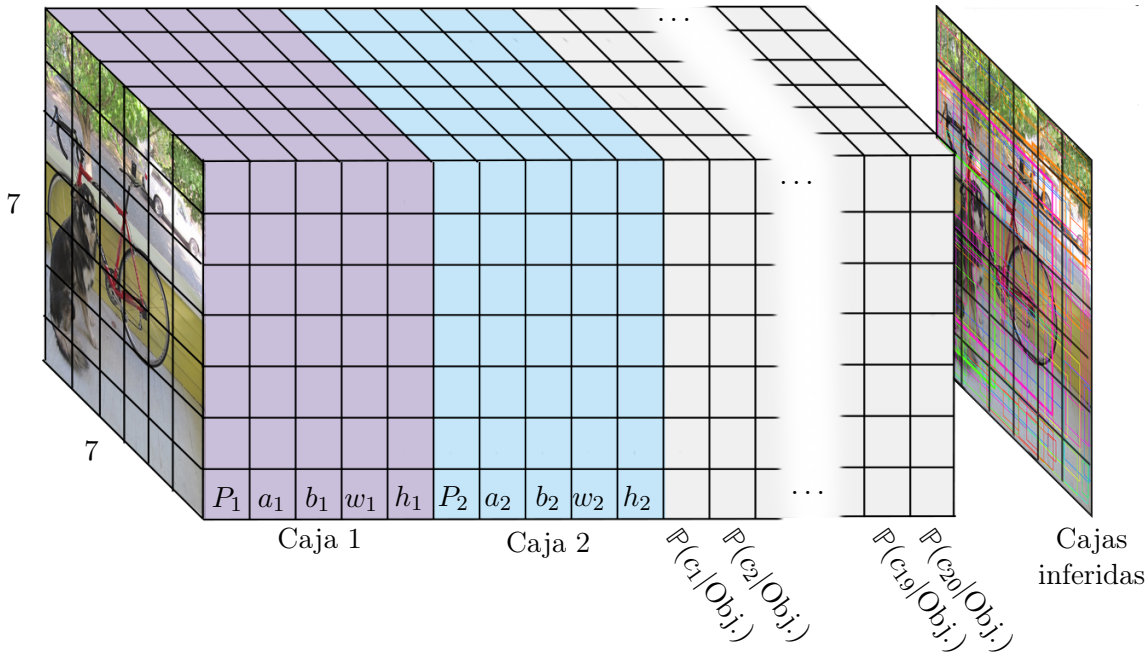


Figura 3.4: Formato de la salida de la red $(S, S, 5B + C)$ con los elementos de (3.1) para cada celda.

Arquitectura de la red

En la jerga de la visión artificial, la red consiste en una estructura de tipo *encoder* + *classifier*, es decir, una primera parte encargada de extraer información abstracta de la imagen y una segunda parte encargada de producir los vectores con las cajas y probabilidades condicionadas. La figura 3.5 muestra un diagrama de la arquitectura de la red que se encuentra explicitada en la tabla 3.1.

La red consiste en la aglutinación de capas convolucionales y de pooling máximo con distintos rellenos y pasos. Al final de la red, tras 24 capas convolucionales, se colocan dos capas densas. A la salida de cada capa se aplica la función *Leaky ReLU* en cada componente, es decir,

$$\mathbf{F}_l(\mathbf{z}) = \begin{cases} f(z_1) \\ \vdots \\ f(z_{d_l}) \end{cases} \qquad \text{donde} \qquad f(x) = \begin{cases} 0.1x & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases}$$

Como excepción, en la última capa, se aplica la función identidad.

Los autores podrían haber optado por emplear algunas otras funciones para imponer el rango de algunos parámtros de salida. Como $a_{ij}, b_{ij}, P_{ij} \in [0, 1]$ podemos usar la función logística (2.4d) para imponer esta condición. También, se podría usar ReLU (2.4a) para imponer $w_{ij}, h_{ij} \in [0, \infty)$. Con la función SoftMax (2.4g) se podría imponer que $\mathbb{P}(c_k|\text{Obj}) \in [0, 1]$ para todo $k = 1, \ldots, C$ y, además, que $\mathbb{P}(c_1|\text{Obj}) + \cdots + \mathbb{P}(c_C|\text{Obj}) = 1$.

Esta arquitectura, que denotaremos $h_{\boldsymbol{\theta}}$, tiene un total de 268M parámetros que ocupan 1.02 Gb en memoria. El 79 % de los parámetros corresponde las dos últimas capas densas, cuando la mayor parte del coste computacional corresponde a las capas convolucionales.

Capa	**Forma Entrada** (ancho, alto, ch)	**Tamaño** ch×(m_1, m_2)	**Paso** (s_1, s_2)	**Relleno** (p_1, p_2)	**Nº Params.**
Conv. 2D	$(448, 448, 3)$	$64\times(7,7)$	$(2,2)$	$(3,3)$	9 472
Poolmax	$(224, 224, 64)$	$(2,2)$	$(2,2)$	$(1,1)$	0
Conv. 2D	$(112, 112, 64)$	$192\times(3,3)$	$(1,1)$	$(1,1)$	110 784
Poolmax	$(112, 112, 192)$	$(2,2)$	$(2,2)$	$(1,1)$	0
Conv. 2D	$(56, 56, 192)$	$128\times(1,1)$	$(1,1)$	$(0,0)$	24 704
Conv. 2D	$(56, 56, 128)$	$256\times(3,3)$	$(1,1)$	$(1,1)$	295 168
Conv. 2D	$(56, 56, 256)$	$256\times(1,1)$	$(1,1)$	$(1,1)$	65 792
Conv. 2D	$(56, 56, 256)$	$512\times(3,3)$	$(1,1)$	$(1,1)$	131 584
Poolmax	$(56, 56, 512)$	$(2,2)$	$(2,2)$	$(1,1)$	0
Conv. 2D	$(28, 28, 512)$	$256\times(1,1)$	$(1,1)$	$(0,0)$	131 328
Conv. 2D	$(28, 28, 256)$	$512\times(3,3)$	$(1,1)$	$(1,1)$	1 180 160
Conv. 2D	$(28, 28, 512)$	$256\times(1,1)$	$(1,1)$	$(0,0)$	131 328
Conv. 2D	$(28, 28, 256)$	$512\times(3,3)$	$(1,1)$	$(1,1)$	1 180 160
Conv. 2D	$(28, 28, 512)$	$256\times(1,1)$	$(1,1)$	$(0,0)$	131 328
Conv. 2D	$(28, 28, 256)$	$512\times(3,3)$	$(1,1)$	$(1,1)$	1 180 160
Conv. 2D	$(28, 28, 512)$	$256\times(1,1)$	$(1,1)$	$(0,0)$	131 328
Conv. 2D	$(28, 28, 256)$	$512\times(3,3)$	$(1,1)$	$(1,1)$	1 180 160
Conv. 2D	$(28, 28, 512)$	$512\times(1,1)$	$(1,1)$	$(0,0)$	131 328
Conv. 2D	$(28, 28, 256)$	$1024\times(3,3)$	$(1,1)$	$(2,2)$	2 360 320
Poolmax	$(28, 28, 1024)$	$(2,2)$	$(2,2)$	$(1,1)$	0
Conv. 2D	$(14, 14, 1024)$	$512\times(1,1)$	$(1,1)$	$(0,0)$	524 800
Conv. 2D	$(14, 14, 512)$	$1024\times(3,3)$	$(1,1)$	$(1,1)$	4 719 616
Conv. 2D	$(14, 14, 1024)$	$512\times(1,1)$	$(1,1)$	$(0,0)$	524 800
Conv. 2D	$(14, 14, 512)$	$1024\times(3,3)$	$(1,1)$	$(1,1)$	4 719 616
Conv. 2D	$(14, 14, 1024)$	$1024\times(3,3)$	$(1,1)$	$(1,1)$	9 438 208
Conv. 2D	$(14, 14, 1024)$	$1024\times(3,3)$	$(2,2)$	$(1,1)$	9 438 208
Conv. 2D	$(7, 14, 1024)$	$1024\times(3,3)$	$(1,1)$	$(1,1)$	9 438 208
Conv. 2D	$(7, 14, 1024)$	$1024\times(3,3)$	$(1,1)$	$(1,1)$	9 438 208
Densa	$(7, 14, 1024)$	–	–	–	205 524 992
Densa	$(1, 1, 4096)$	–	–	–	6 022 590
Salida	$(7, 7, 30)$				

Tabla 3.1: Detalle de las capas que componen la red YOLO. Cada bloque de la tabla representa un bloque de la figura 3.5. La abreviatura *ch* hace referencia al número de canales de la salida. El número total de parámetros es 268 164 350.

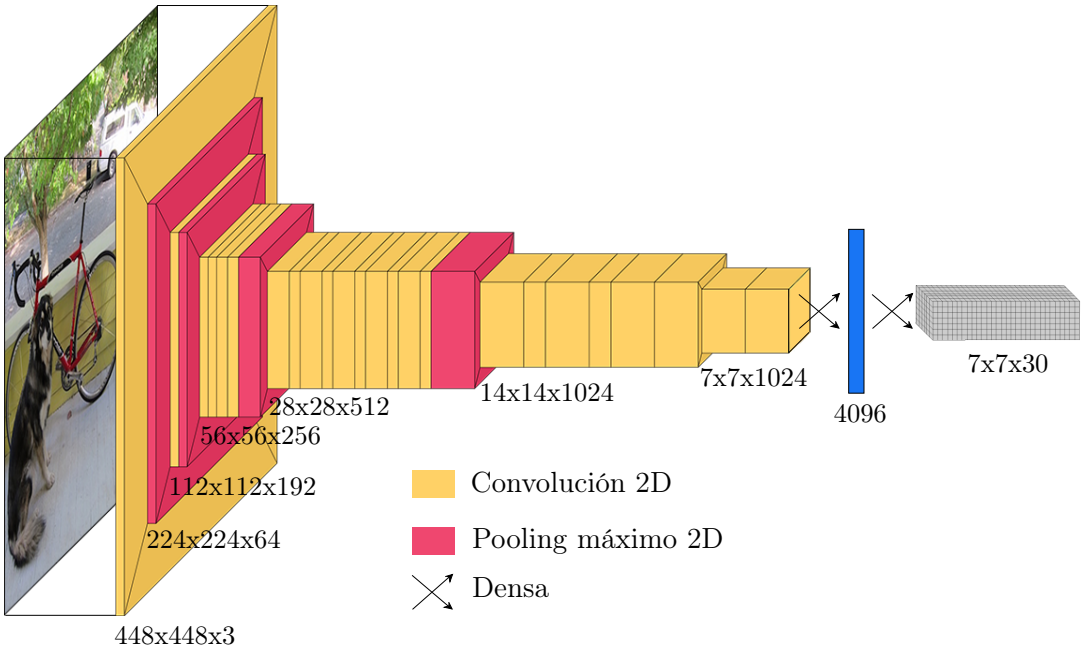


Figura 3.5: Arquitectura de la red YOLO. El detalle de cada capa se encuentra en la tabla 3.1.

Entrenamiento

Es muy difícil encontrar el óptimo para la red con una técnica de tipo descenso del gradiente sin partir desde cerca del mínimo. Por ello, el entrenamiento se realiza en dos partes:

1. Preentrenamiento primeras 20 capas convolucionales para clasificación.
2. Entrenamiento del modelo $h'_{\boldsymbol{\theta}'}$ para detección.

El preentrenamiento se realiza para un problema más sencillo como es la clasificación, para el cual será más fácil encontrar el óptimo. Como la arquitectura necesaria para un problema de clasificación no es la misma que la que necesitamos para un problema de detección deberemos adaptar la arquitectura de detección. Esto constituye, de nuevo, un modelo de aprendizaje automático en sí mismo.

Se toman las 20 primeras capas convolucionales seguidas de una capa de pooling promedio y una capa densa. Además, se cambia el paso de la primera capa convolucional a 1, obteniendo una capa con el mismo número de parámetros pero con una entrada a la mitad de resolución (véase figura 3.6). La entrada para esta red es $\mathcal{X}' = \{0, 1, \ldots, 2^8 - 1\}^{224 \times 224 \times 3}$. Llamaremos a esta nueva red $h'_{\boldsymbol{\theta}'}$. La salida de la última capa es un vector con mil elementos $\hat{\mathbf{y}}' \in \mathcal{Y}' = \mathbb{R}^{1000}$, cada uno corresponde a la puntuación $\hat{y}'_k$ que otorga la red al clasificar esa imagen en una categoría k-ésisma en $\mathscr{C}'$. Por supuesto, puede haber más de una categoría en la imagen.

La red se entrena durante aproximadamente una semana en el dataset ImageNet 2012 de clasificación en 1000 categorías (Russakovsky *et al.*, 2015). Téngase en cuenta que este conjunto de categorías es distinto del de la red original y por eso lo denotamos $\mathscr{C}'$. Redmon;

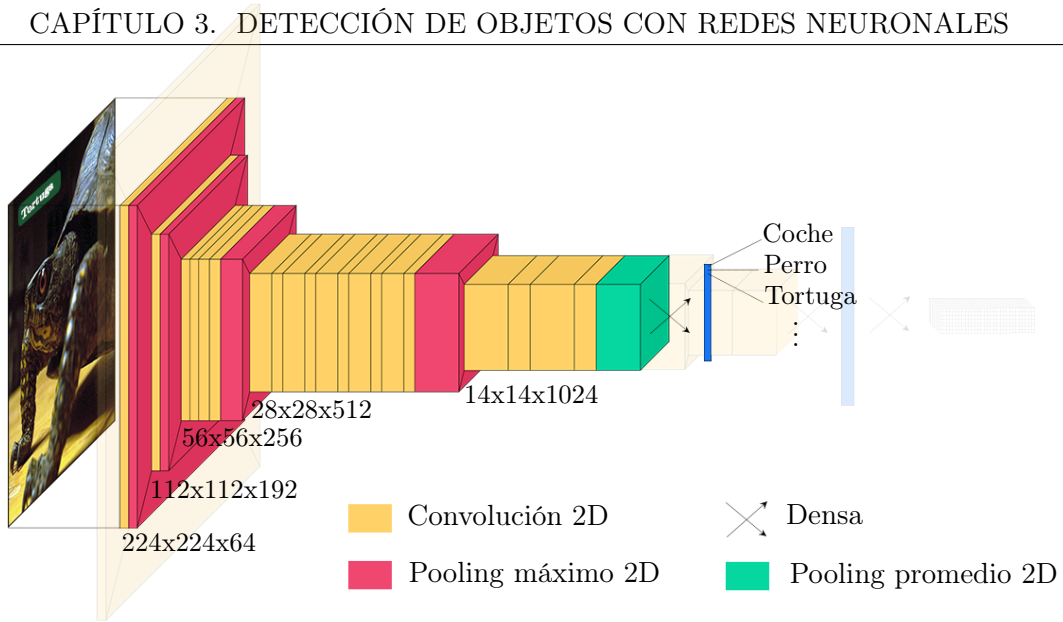


Figura 3.6: Preentrenamiento del modelo para un problema de clasificación. Translúcido puede verse el modelo original.

Divvala *et al.* (2016) no especifican la función de pérdida de este preentrenamiento ni la función de activación de la última capa. Una opción común es usar la función de activación logística (2.4d) en cada componente y la función de pérdida ℓ' *entropía cruzada binaria*:

$$\ell'(\mathbf{y}', \hat{\mathbf{y}}') = \frac{1}{|\mathscr{C}'|} \sum_{k=1}^{|\mathscr{C}'|} \left(\mathbb{1}_k^{\text{obj.}}(\mathbf{y}) \log(\hat{y}_k) + \mathbb{1}_k^{\text{no obj.}}(\mathbf{y}) \log(1 - \hat{y}_k) \right)$$

donde la función indicatriz $\mathbb{1}_k^{\text{obj.}}(\mathbf{y}) = y'_k$ es 1 si la imagen pertenece a la categoría k-ésima y $\mathbb{1}_k^{\text{no obj.}}(\mathbf{y}) = 1 - \mathbb{1}_k^{\text{obj.}}(\mathbf{y})$.

Con las primeras 20 capas convolucionales preentrenadas, se forma el modelo completo (figura 3.5) y se entrena sobre los datasets Pascal VOC (Everingham *et al.*, 2007) y (Everingham *et al.*, 2012) de detección de 20 categorías. La experiencia dice que los pesos calculados con la red $h'_{\boldsymbol{\theta}'}$ serán un buen punto de partida para la red $h_{\boldsymbol{\theta}}$ que queremos entrenar.

Ahora vamos a ver cómo se generan las etiquetas de referencia de la red neuronal $\mathbf{y}^*$ a partir de las etiquetas de referencia de las que disponemos $\mathbf{y}$. A estas etiquetas de referencia también nos referiremos como etiquetas reales. Recordemos que $\mathbf{y}$ es un conjunto que contiene un número de objetos. Por ejemplo, una etiqueta con tres objetos sería $\mathbf{y} = \{o_1, o_2, o_3\}$ donde cada objeto está formado por una caja y una categoría $o_i = \big((a_i, b_i, w_i, h_i), c_i\big)$. Como la red está prediciendo $S^2 \cdot B$ cajas, muchas más de las que hay en la etiqueta $\mathbf{y}$, está claro que no podemos calcular todas las que aparecen en $\mathbf{y}^*$. También, predecimos una categoría para cada una de las S^2 y solo podemos saber la categoría de algunas celdas que tengan objeto. Esta dificultad se salva haciendo que en la función de pérdida (3.2), que veremos un poco más adelante, no aparezcan esos términos que no podemos calcular. Para cada caja en $\mathbf{y}$ buscaremos una caja $\mathbf{y}^*$ responsable de representar este objeto. Esto lo vamos a hacer de la siguiente forma:

1. Para cada objeto real en $\mathbf{y}$, se toma el centro de la caja y se calcula la celda i-ésima en la que se encuentra (véase figura 3.7). Después, se calcula la caja (a, b, w, h) relativa a esa celda. Además, se establecen la probabilidad $\mathbb{P}(\text{Obj}) = 1$, que servirá

para calcular P más adelante, y, si el objeto real es de la clase c_k, $\mathbb{P}_i(c_k|\text{Obj}) = 1$ y $\mathbb{P}_i(c_r|\text{Obj}) = 0$ para todo $r \neq k$ en la celda.

2. Para las celdas que no tienen ningún objeto real asociado, fijamos $P = 0$ y no nos importan el resto de valores, pues no aparecerán en la función de pérdida (3.2) más adelante.

Para entrenar el modelo, Redmon; Divvala *et al.* (2016) usan la función de pérdida:

$$\begin{aligned}
&\ell(\mathbf{y}^*, \hat{\mathbf{y}}^*) = \\
&= \lambda_{\text{coord.}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{\text{obj.}} \Big[\big(a_{ij} - \hat{a}_{ij}\big)^2 + \big(b_{ij} - \hat{b}_{ij}\big)^2 + \big(\sqrt{w_{ij}} - \sqrt{\hat{w}_{ij}}\big)^2 + \big(\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}}\big)^2 \Big] + \\
&+ \sum_{i=1}^{S^2} \sum_{j=1}^{B} \Big[\mathbb{1}_{ij}^{\text{obj.}} + \lambda_{\text{no obj.}} \mathbb{1}_{ij}^{\text{no obj.}} \Big] \Big(P_{ij} - \hat{P}_{ij} \Big)^2 + \sum_{i=1}^{S^2} \sum_{k=1}^{C} \mathbb{1}_{i}^{\text{obj.}} \Big(\mathbb{P}_i(c_k|\text{Obj.}) - \hat{\mathbb{P}}_i(c_k|\text{Obj.}) \Big)^2
\end{aligned} \tag{3.2}$$

donde $\lambda_{\text{coord.}} = 5$ y $\lambda_{\text{no obj.}} = 0.5$ son dos constantes[4]. Las siguientes funciones indicatrices representan: $\mathbb{1}_i^{\text{obj.}}$ si hay un objeto en la celda i, $\mathbb{1}_{ij}^{\text{obj.}}$ si hay un objeto en la celda i y la caja j es responsable de representar dicho objeto y $\mathbb{1}_{ij}^{\text{no obj.}} = 1 - \mathbb{1}_{ij}^{\text{obj.}}$. Dentro de una celda, elegimos la caja j-ésima para ser responsable de la predicción con aquella que tiene el valor máximo de IoU(real, pred.) (véase figura 3.8). Abusando de notación, $P_{ij}, a_{ij}, b_{ij}, w_{ij}, h_{ij}$ es la j-ésima caja predicha por la cela i-ésima y $\mathbb{P}_i(c_k|\text{Obj.})$ la probabilidad condicionada predicha para la k-ésima e i-ésima celda. Los acentos representan los valores predichos por la red. La dependencia explícita con $\mathbf{y}^*$ e $\hat{\mathbf{y}}^*$ se omite para simplificar las escritura.

La función de pérdida (3.2) consta de tres términos que vamos a ver uno por uno. El término

$$\lambda_{\text{coord.}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{\text{obj.}} \Big[\big(a_{ij} - \hat{a}_{ij}\big)^2 + \big(b_{ij} - \hat{b}_{ij}\big)^2 + \big(\sqrt{w_{ij}} - \sqrt{\hat{w}_{ij}}\big)^2 + \big(\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}}\big)^2 \Big]$$

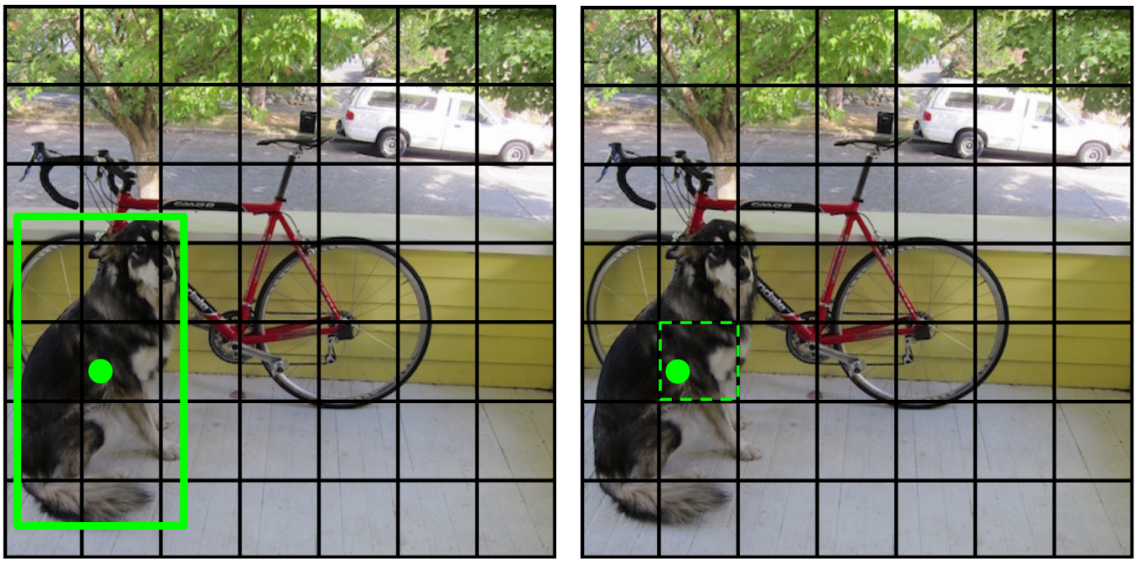
es el *error de localización* cometido por las cajas. Ponderamos más este error que el resto de términos. Debido a la función indicatriz $\mathbb{1}_{ij}^{\text{obj.}}$, solo penalizamos este error cuando hay un objeto en la celda y la caja que estamos tratando es la responsable de ese objeto. Por eso, si en una celda no hay ningún objeto real o solo hay uno, no son necesarios los valores de $a_{ij}, b_{ij}, \ldots$. Se estima $\sqrt{w_{ij}}, \sqrt{h_{ij}}$ en lugar de w_{ij}, h_{ij} para, de algún modo, penalizar más las diferencias cuando las cajas son más grandes que cuando son más pequeñas. El término

$$\sum_{i=1}^{S^2} \sum_{j=1}^{B} \Big[\mathbb{1}_{ij}^{\text{obj.}} + \lambda_{\text{no obj.}} \mathbb{1}_{ij}^{\text{no obj.}} \Big] \Big(P_{ij} - \hat{P}_{ij} \Big)^2$$

es el *error de identificación*. Damos menos peso al valor obtenido cuando no hay objeto y queremos que $\hat{P}_{ij} \approx 0$. El término

$$\sum_{i=1}^{S^2} \sum_{k=1}^{C} \mathbb{1}_{i}^{\text{obj.}} \Big(\mathbb{P}_i(c_k|\text{Obj.}) - \hat{\mathbb{P}}_i(c_k|\text{Obj.}) \Big)^2$$

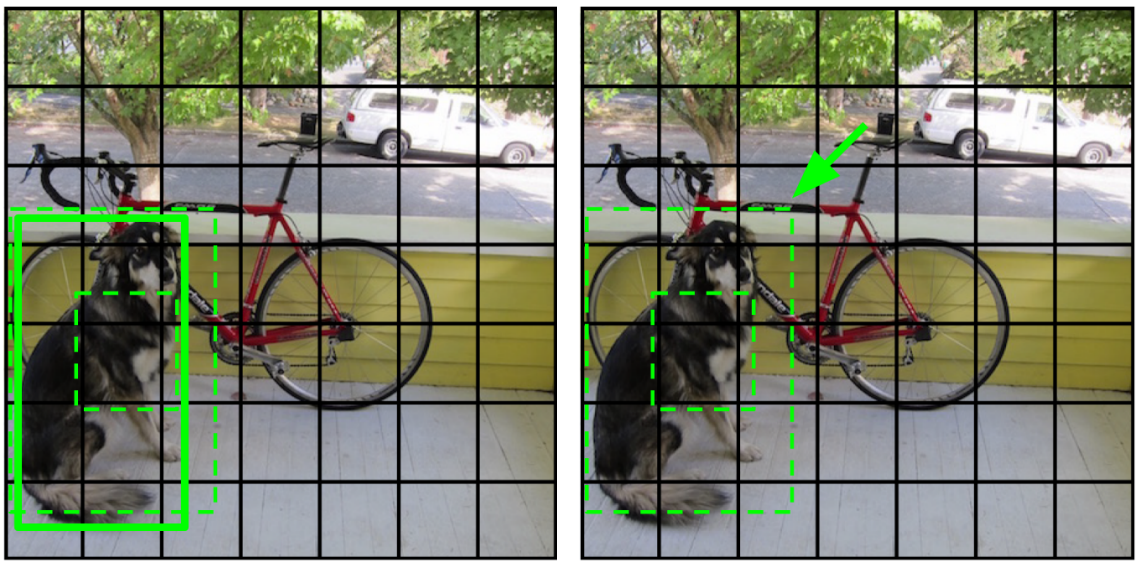
[4]En la jerga de la inteligencia artificial, las constantes arbitrareas que defines el modelo se conocen como *hiperparámetros*. También son hiperparámetros S o B.



(a) Etiquetas real con su centro.

(b) Celda elegida para responsabilizarse de esta etiqueta.

Figura 3.7: Proceso de selección de celdas para cada etiqueta. Crédito Redmon (2016).



(a) En línea continua objeto real, en línea discontinua objetos candidatos.

(b) En línea discontinua los candidatos, con una flecha el candidato seleccionado.

Figura 3.8: Elección del candidato predicho por la red que representa un objeto real. Crédito Redmon (2016).

es el *error de clasificación*. Debido a la función indicatriz $\mathbb{1}_i^{\text{obj}\cdot}$, solo penalizamos este error cuando hay un objeto en la celda.

La optimización se realiza durante ~ 135 épocas empleando el algoritmo 2.5 de Descenso Estocástico del Gradiente con Momento para minimizar el riesgo empírico $\mathcal{R}_S$. Redmon; Divvala *et al.* (2016) usan un tamaño de lote de 64, un momento constante $\alpha = 0.9$ y un ratio de aprendizaje γ_j variable. El ratio de aprendizaje se incrementa desde 10^{-3} hasta 10^{-2} durante las primeras 75 épocas después se reduce a 10^{-3} durante 30 épocas y a 10^{-4} durante otras 30 épocas.

3.1.2 NON-MAXIMUN SUPPRESION (NMS)

La red neuronal genera $B \cdot S^2$ predicciones (véase figura 3.2a) codificadas en $\hat{\mathbf{y}}^*$. Debemos eliminar las predicciones incorrectas de la red obteniendo unas pocas predicciones finales que correspondan con $\mathbf{y}$ (véase figura 3.2b). Existen dos tipos de predicciones incorrectas: aquellas que no corresponden con ningún objeto real y aquellas que se encuentran duplicadas o la caja no se adecua al objeto.

En primer lugar, se eliminan todas las predicciones cuya puntuación P_{ij} no supera un umbral P_{min} mínimo. El umbral mínimo P_{min} es un hiperparámetro del modelo, se fija en $P_{\text{min}} = 0.45$. Con esto se pretende eliminar las predicciones que no corresponden a ninguna detección.

Después, para elegir una caja de entre varios candidatos superpuestos, se aplica el algoritmo algoritmo 3.1 de *Non Maximun Suppresion*. Vamos a abusar de notación llamando $\hat{\mathbf{y}}^*$ al tensor de dimensiones $\big(S \times S \times (5B + C)\big)$ y al conjunto de S^2 elementos con los objetos equivalentes: rectángulo (a, b, w, h) y categoría c. La categoría para ese rectángulo se escoge como aquella con $\mathbb{P}(c_k|\text{Obj.})$ máxima.

Algoritmo 3.1: Non-Maximun Supression (NMS)

Datos:
Conjunto de objetos candidatos: $\hat{\mathbf{y}}^* = \{\hat{\mathbf{o}}_1, \ldots, \hat{\mathbf{o}}_r\}$;
Umbral IoU máximo: $\text{IoU}_{\text{máx}}$;
Resultado:
Lista de predicciones: $\hat{\mathbf{y}} = \{\hat{\mathbf{o}}_{i_1}, \ldots, \hat{\mathbf{o}}_{i_s}\}$;

```
ŷ = ∅;
mientras ŷ* ≠ ∅ hacer
    Elegir ô_{i_j} en ŷ* con puntuación P máxima;
    Añadir ô_{i_j} a ŷ;
    Eliminar ô_{i_j} de ŷ*;
    para cada ô_{i_k} ∈ ŷ* hacer
        si IoU(ô_{i_j}, ô_{i_k}) ≥ IoU_máx entonces
            Eliminar ô_{i_k} de ŷ*;
        fin
    fin
fin
```

El algoritmo algoritmo 3.1 elige la caja con máxima puntuación P de entre aquellas

cuyo IoU es mayor que un umbral $\text{IoU}_{\text{máx}}$. El umbral máximo es un hiperparámetro y se fija en $\text{IoU}_{\text{máx}} = 0.45$. En ocasiones, es habitual referirse a la combinación de: filtrar las predicciones cuyo P_{ij} no llega al mínimo y el algoritmo 3.1 bajo el nombre único de NMS que le hemos dado al algoritmo 3.1.

Capítulo 4

CONCLUSIONES

Las redes neuronales necesitan de una gran cantidad de áreas distintas de las matemáticas. Empezamos este trabajo hablando de probabilidad y estadística para establecer un modelo general para el aprendizaje supervisado que hemos usado varias veces.

Seguimos este trabajo introduciendo las redes neuronales prealimentadas: una familia de funciones construidas mediante la concatenación sucesiva de funciones lineales y funciones no lineales. Vimos como muchas veces podemos representar estas mediante grafos, pertenecientes a la matemática discreta.

Centramos la mirada en las redes neuronales convolucionales, donde acudimos al análisis matemático para definir la convolución discreta de matrices. Recurrimos al álgebra lineal para poder operar con las matrices usando sus coordenadas. La convolución nos permitió definir una forma muy concreta para las aplicaciones lineales de la red neuronal con muchos menas aristas entre neuronas. Esto es esencial para reducir drásticamente el tamaño de la clase de hipótesis y evitar caer en un enorme error de estimación. Terminamos el capítulo acudiendo al análisis numérico, el análisis matemático y la estadística para buscar algoritmos capaces de minimizar el riesgo empírico como: descenso del gradiente, descenso estocástico del gradiente y descenso estocástico del gradiente con momento.

Los algoritmos de optimización hicieron patentes dos realidades: la ingente cantidad de recursos computacionales necesarios para la optimización de estos modelos y el poco desarrollo de la teoría en este campo. Debimos acudir al descenso del gradiente estocástico porque no podíamos calcular derivadas con respecto de millones de parámetros para miles de imágenes en cada iteración. En lugar de calcular el gradiente del riesgo empírico, calculamos una estimación correcta en promedio. Los métodos presentados, aunque tienen excelentes garantías teóricas bajo algunas hipótesis, no dan garantías para las redes neuronales y se desconocen condiciones bajo las que estos algoritmos convergen cuando trabajamos con redes neuronales.

En el último capítulo, vimos un caso práctico de aplicación de redes neuronales: la detección a tiempo real de objetos en imágenes. Estudiamos el algoritmo YOLO (You Only Look Once) que fue el primer algoritmo a tiempo real con una precisión suficiente para ser aceptado por la comunidad. Vimos su modelo de aprendizaje automático que tomaba imágenes y devolvía conjuntos de rectángulos delimitadores. El algoritmo se componía de dos partes: una red neuronal convolucional y un algoritmo de filtrado. La red neuronal

convolucional constituye otro modelo de aprendizaje automático en sí mismo. Entrenar un modelo tan complicado es muy difícil sino partimos desde cerca del óptimo. Para el entrenamiento, configuramos una segunda red neuronal haciendo un tercer modelo de aprendizaje automático. Esta segunda red neuronal nos sirvió para una estimación inicial de los pesos de la red de interés.

Haber realizado este trabajo me ha dado una base sólida sobre las redes neuronales convolucionales y su entrenamiento. También, he alcanzado una perspectiva global sobre como las distintas áreas de las matemáticas se integran unas con otras para soportar aplicaciones diversas de las matemáticas. Mi aportaciones son dos. La primera, aunar las diferentes piezas fundamentales para que esto sea posible en un único texto. La segunda, traducir del lenguaje usando en otros campos, como la ingeniería informática, al rigor y precisión matemáticos que cabría esperar de un graduado en matemáticas. En este proceso, se han añadido tantas explicaciones como se han considerado necesarias que no se encuentran en los textos de referencia. Esto se hizo especialmente patente para completar muchas piezas ausentes del artículo de Redmon; Divvala *et al.* (2016) sobre YOLO.

Este trabajo puede continuarse bibliográficamente estudiando las mejoras introducidas en la arquitectura YOLO (Redmon; Divvala *et al.*, 2016) que han surgido en años posteriores como YOLO9000 (Redmon y Farhadi, 2017) o YOLOv3 (Redmon y Farhadi, 2018) o en la última versión de este algoritmo, YOLOv8, desarrollada por la empresa Ultralytics. También se puede adentrar en el seguimiento de objetos, es decir, en una forma de elegir los rectángulos delimitadores que tenga coherencia temporal. Como líneas de investigación, se puede ahondar en las garantías estadísticas y computacionales del desempeño y optimización de distintos modelos de redes neuronales para distintas tareas.

BIBLIOGRAFÍA

BUBECK, Sébastien, 2015. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*. Vol. 8, págs. 231-357. ISSN 19358245. Disp. desde DOI: 10.1561/2200000050.

DUMOULIN, Vincent y VISIN, Francesco, 2016. A guide to convolution arithmetic for deep learning. También disponible en: http://arxiv.org/abs/1603.07285.

EVERINGHAM, M.; VAN GOOL, L.; WILLIAMS, C. K. I.; WINN, J. y ZISSERMAN, A., 2007. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results* [en línea]. [visitado 2024-04-09]. Disp. desde: http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

EVERINGHAM, M.; VAN GOOL, L.; WILLIAMS, C. K. I.; WINN, J. y ZISSERMAN, A., 2012. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results* [en línea]. [visitado 2024-04-09]. Disp. desde: http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

GOODFELLOW, Ian; BENGIO, Yoshua y COURVILLE, Aaron, 2016. *Deep Learning*. MIT Press. ISBN 9780262337373.

REDMON, Joseph, 2016. *YOLO talk at CVPR 2016*. CVPR. También disponible en: https://www.youtube.com/watch?v=NM6lrxy0bxs.

REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross y FARHADI, Ali, 2016. You Only Look Once: Unified, Real-Time Object Detection. En: IEEE, págs. 779-788. ISBN 978-1-4673-8851-1. Disp. desde DOI: 10.1109/CVPR.2016.91.

RUSSAKOVSKY, Olga; DENG, Jia; SU, Hao; KRAUSE, Jonathan; SATHEESH, Sanjeev; MA, Sean; HUANG, Zhiheng; KARPATHY, Andrej; KHOSLA, Aditya; BERNSTEIN, Michael; BERG, Alexander C. y FEI-FEI, Li, 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*. Vol. 115, págs. 211-252. ISSN 0920-5691. Disp. desde DOI: 10.1007/s11263-015-0816-y.

SHALEV-SHWARTZ, Shai y BEN-DAVID, Shai, 2014. *Understanding Machine Learning*. Cambridge University Press. ISBN 9781107057135. Disp. desde DOI: 10.1017/CBO9781107298019.

THEVENOT, Alex, 2020. *Conv2d: Finally Understand What Happens in the Forward Pass: A visual and mathematical explanation of the 2D convolution layer and its arguments* [en línea]. [visitado 2024-03-26]. Disp. desde: https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148.

REFERENCIAS HISTÓRICAS

ALOM, Md Zahangir; TAHA, Tarek M.; YAKOPCIC, Christopher; WESTBERG, Stefan; SIDIKE, Paheding; NASRIN, Mst Shamima; ESESN, Brian C Van; AWWAL, Abdul A S. y ASARI, Vijayan K., 2018. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. También disponible en: http://arxiv.org/abs/1803.01164.

BUCHANAN, Bruce G., 2006. A (very) brief history of artificial intelligence. *AI magazine*. Vol. 26, págs. 53-60. ISSN 0738-4602. También disponible en: https://www.proquest.com/docview/208132026?accountid=14778.

FUKUSHIMA, Kunihiko, 1975. *Cognitron: A Self-organizing Multilayered Neural Network*. Vol. 20. Disp. desde DOI: doi.org/10.1007/BF00342633.

FUKUSHIMA, Kunihiko, 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. Vol. 36, págs. 193-202. ISSN 0340-1200. Disp. desde DOI: 10.1007/BF00344251.

GIRSHICK, Ross, 2015. Fast R-CNN. En: IEEE, págs. 1440-1448. ISBN 978-1-4673-8391-2. Disp. desde DOI: 10.1109/ICCV.2015.169.

GIRSHICK, Ross; DONAHUE, Jeff; DARRELL, Trevor y MALIK, Jitendra, 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. En: IEEE, págs. 580-587. ISBN 978-1-4799-5118-5. Disp. desde DOI: 10.1109/CVPR.2014.81.

HAENLEIN, Michael y KAPLAN, Andreas, 2019. A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California Management Review*. Vol. 61, págs. 5-14. ISSN 0008-1256. Disp. desde DOI: 10.1177/0008125619864925.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya e HINTON, Geoffrey E, 2012. *Advances in Neural Information Processing Systems*. ImageNet Classification with Deep Convolutional Neural Networks. Ed. por PEREIRA, F.; BURGES, C.J.; BOTTOU, L. y WEINBERGER, K.Q. Curran Associates, Inc. ISBN 9781627480031. También disponible en: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c84%5C%5C36e924a68c45b-Paper.pdf.

LECUN, Y.; BOTTOU, L.; BENGIO, Y. y HAFFNER, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. Vol. 86, págs. 2278-2324. ISSN 00189219. Disp. desde DOI: 10.1109/5.726791.

POLYAK, B.T., 1964. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*. Vol. 4, págs. 1-17. ISSN 00415553. Disp. desde DOI: 10.1016/0041-5553(64)90137-5.

REDMON, Joseph, [s.f.]. *You Only Look Once: Unified, Real-Time Object Detection. Reviewers comments from NISP* [en línea]. [visitado 2024-27]. Disp. desde: https://pjreddie.com/publications/yolo/.

REDMON, Joseph y FARHADI, Ali, 2017. YOLO9000: Better, faster, stronger. En: Institute of Electrical y Electronics Engineers Inc. Vol. 2017-January, págs. 6517-6525. ISBN 9781538604571. Disp. desde DOI: 10.1109/CVPR.2017.690.

REDMON, Joseph y FARHADI, Ali, 2018. YOLOv3: An Incremental Improvement. También disponible en: http://arxiv.org/abs/1804.02767.

SIMONYAN, Karen y ZISSERMAN, Andrew, 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. También disponible en: http://arxiv.org/abs/1409.1556.

Índice de figuras

Índice de tablas

Índice de algoritmos