



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

**Implementación optimizada sobre
sistemas heterogéneos de algoritmos de
Deep Learning para clasificación de
imágenes**

Autor
David Sánchez Pérez

Directores
José Miguel Mantas Ruiz



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, mes de Febrero 2024



**Implementación optimizada sobre
sistemas heterogéneos de algoritmos de
Deep Learning para clasificación de
imágenes**

Autor

David Sánchez Pérez

Directores

José Miguel Mantas Ruiz

Título del Proyecto: Subtítulo del proyecto

Nombre Apellido1 Apellido2 (alumno)

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3,

Resumen

Poner aquí el resumen.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3,

Abstract

Write here the abstract in English.

Yo, **Nombre Apellido1 Apellido2**, alumno de la titulación **TITULACIÓN de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Nombre Apellido1 Apellido2

Granada a X de mes de 201 .

D. **Nombre Apellido1 Apellido2 (tutor1)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

D. **Nombre Apellido1 Apellido2 (tutor2)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Título del proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Nombre Apellido1 Apellido2 (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

Los directores:

Nombre Apellido1 Apellido2 (tutor1) **Nombre Apellido1 Apellido2 (tutor2)**

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción	1
1.1. Resumen	1
1.2. Objetivos	2
1.2.1. Objetivos de aprendizaje	2
1.2.2. Objetivos de diseño y desarrollo	3
2. Conceptos previos	5
2.1. Machine Learning	5
2.2. Componentes necesarios para el aprendizaje supervisado . . .	5
2.3. División de datos en entrenamiento y test	6
2.4. Tipos de aprendizaje	6
2.4.1. Aprendizaje Supervisado	6
2.4.2. Aprendizaje No Supervisado	6
2.4.3. Aprendizaje Por Refuerzo	6
2.5. Redes Neuronales Totalmente Conectadas	7
2.5.1. Neurona	7
2.5.2. Estructura por capas	7
2.5.3. Funciones de activación	8
2.5.4. Capa SoftMax	9
2.5.5. Función de error o pérdida	10
2.5.6. Descenso del gradiente	10
2.5.7. Inicialización de pesos y sesgos	12
2.5.8. Tipos de codificaciones	12
2.5.9. Propagación hacia delante con softmax	12
2.6. Redes Neuronales Convolucionales	13
2.6.1. Introducción a CNNs	13
2.6.2. Capa convolucional	13
2.6.3. Capa de agrupación máxima	19
2.6.4. Capa de aplanado	22
2.7. cuDNN (Deep Neural Network)	23
2.7.1. Manejador	24
2.7.2. Tensores	24
2.7.3. Principales funciones	26

3. Aportaciones	33
3.1. Redes Neuronales Totalmente Conectadas	33
3.1.1. Gradiente de la función de pérdida respecto a Soft-Max, [1] [2]	33
3.1.2. BackPropagation con 1 capa oculta [3] [4]	36
3.1.3. Retropropagación con 2 capas ocultas	42
3.1.4. Conclusiones	49
3.2. Paralelización en entrenamiento	51
3.3. Redes Neuronales Convolucionales	53
3.3.1. Propagación hacia detrás	53
3.3.2. Propagación hacia detrás con relleno	61
4. Adaptación GPU	71
4.1. Convolución como GEMM	72
4.1.1. Memoria requerida al emplear GEMM	73
4.2. Retropropagación GEMM en capa convolucional	74
4.3. Capa totalmente conectada como GEMM [5]	77
4.3.1. Propagación hacia delante	77
4.3.2. Retropropagación	78
4.4. CUDA	80
4.4.1. Multiplicación de matrices en CUDA	80
5. Comparación con distintas plataformas	83
5.1. cuDNN	83

Índice de figuras

2.1. Imagen de una neurona	7
2.2. Imagen de una capa de neuronas	7
2.3. Imagen de la función de activación ReLU	8
2.4. Imagen de la función de activación Sigmoide	9
2.5. Imagen de la función de activación SoftMax	9
2.6. Ejemplo de funcionamiento del descenso del gradiente	10
2.7. Componentes en una capa convolucional	13
2.8. Propagación hacia delante en una capa convolucional	14
2.9. Propagación hacia delante en una capa convolucional con va- rios canales de profundidad	15
2.10. Propagación hacia delante en una capa convolucional con va- rios filtros	16
2.11. Relleno sobre un volumen de entrada X	17
2.12. Relleno sobre un volumen de entrada X	17
2.12. Relleno sobre un volumen de entrada X	18
2.13. Componentes en una capa de agrupación máxima	19
2.14. Propagación hacia delante en una capa de agrupación máxima	19
2.15. Propagación hacia delante en una capa de agrupación máxima	20
2.16. Propagación hacia delante en una capa de agrupación máxima	20
2.17. Propagación hacia delante en una capa de agrupación máxima	22
2.18. Propagación hacia delante en una capa de agrupación máxima	23
2.19. Ejemplo de tensor 4D con dimensiones: N=1, C=1, H=5, y W=6	25
2.20. Ejemplo de tensor 4D NCHW con dimensiones: N=1, C=1, H=5, y W=6	26
3.1. Estructura de una red con softmax	33
3.2. Red Neuronal totalmente conectada con 1 capa oculta	36
3.3. Imagen de backpropagation en la capa softmax	37
3.4. Imagen de backpropagation en los pesos entre la capa oculta y la capa SoftMax	37
3.5. Imagen de los 'caminos' desde la capa softmax hasta n_0^1	38
3.6. Imagen de backpropagation en la capa oculta h1	39

3.7. Imagen de backpropagation en los pesos entre la capa input y la capa oculta h1	40
3.8. Imagen de los 'caminos' desde la capa oculta h1 hasta n_0^0	41
3.9. Imagen de backpropagation en la capa input	41
3.10. Red Neuronal totalmente conectada con 2 capas ocultas	42
3.11. Imagen de backpropagation en la capa softmax	42
3.12. Imagen de backpropagation en los pesos entre la capa oculta h2 y la capa SoftMax	43
3.13. Imagen de los 'caminos' desde la capa softmax hasta n_0^2	44
3.14. Imagen de backpropagation en la capa oculta h2	44
3.15. Imagen de backpropagation en los pesos entre las capas ocul- tas h1 y h2	45
3.16. Imagen de los 'caminos' desde la capa softmax hasta n_0^1	46
3.17. Imagen de backpropagation en la capa oculta h1	47
3.18. Imagen de backpropagation en los pesos entre la capa input y la capa oculta h1	47
3.19. Imagen de los 'caminos' desde la capa oculta h1 hasta n_0^0	48
3.20. Imagen de backpropagation en la capa input	49
3.21. Imagen de backpropagation en la capa l	50
3.22. Imagen de backpropagation en los pesos entre la capa l-1 y l .	50
3.23. Ejemplo de propagación hacia delante en una capa convolu- cional	53
3.24. Ejemplo de propagación hacia delante en una capa convolu- cional	54
3.25. Ejemplo de propagación hacia delante en una capa convolu- cional	55
3.26. Ejemplo de propagación hacia delante en una capa convolu- cional	56
3.27. Cálculo del gradiente de la pérdida respecto a cada filtro como convolución entre X e Y	58
3.28. Invertir pesos en K tanto horizontal como verticalmente	59
3.29. Cálculo del gradiente de la pérdida respecto a cada filtro como convolución entre X e Y	60
3.30. Ejemplo de propagación hacia detrás en una capa convolucio- nal con relleno	61
3.31. Retropropagación de Y_{11}^c	61
3.32. Retropropagación de Y_{12}^c	62
3.33. Retropropagación de Y_{13}^c	63
3.34. Retropropagación de Y_{21}^c	63
3.35. Retropropagación de Y_{22}^c	64
3.36. Retropropagación de Y_{23}^c	65
3.37. Retropropagación de Y_{31}^c	66
3.38. Retropropagación de Y_{32}^c	66
3.39. Retropropagación de Y_{33}^c	67

3.40. Cálculo del gradiente de la pérdida respecto a cada filtro como convolución entre X e Y	68
3.41. Cálculo del gradiente de la pérdida respecto de la entrada como convolución	69
3.42. Cálculo del gradiente de la pérdida respecto a la entrada X con uno o dos niveles de relleno	70
4.1. Imagen de una convolución estándar frente a una convolución empleando GEMM	72
4.2. Retropropagación en una capa convolucional de forma estándar frente a GEMM respecto a la entrada	75
4.3. Retropropagación en una capa convolucional de forma estándar frente a GEMM respecto a los pesos	76
4.4. Propagación GEMM hacia delante en una capa totalmente conectada	77
4.5. Propagación GEMM de un minibatch entero hacia delante en una capa totalmente conectada	77
4.6. Cálculo del gradiente respecto a la entrada en una capa totalmente conectada	78
4.7. Cálculo del gradiente respecto a la entrada de todo un minibatch en una capa totalmente conectada	78
4.8. Cálculo del gradiente respecto a los pesos en una capa totalmente conectada	79
4.9. Cálculo del gradiente respecto a los pesos de todo un minibatch en una capa totalmente conectada	79
4.10. Tercera implementación de multiplicación matricial con CUDA	81
4.11. Cuarta implementación de multiplicación matricial con CUDA	82

Índice de cuadros

Capítulo 1

Introducción

1.1. Resumen

En los inicios, las computadoras empleaban exclusivamente CPUs para llevar a cabo tareas de programación generales. Sin embargo, desde la última década empezaron a surgir otros elementos de procesamiento como las GPUs, las cuales se desarrollaron inicialmente para realizar cálculos gráficos paralelos especializados. Con el tiempo, han ido evolucionando tanto en prestaciones como en versatilidad, permitiendo a día de hoy su uso en tareas de cómputo paralelo de propósito general de alto rendimiento.

Gracias a ello se logró el cambio de sistemas homogéneos a heterogéneos, el cual destaca por ser un logro de gran importancia y considerable magnitud en toda la historia de la computación de alto rendimiento.

La computación homogénea emplea uno o más procesadores de la misma arquitectura para ejecutar una aplicación. Por otro lado, la computación heterogénea no se rige por esas reglas y rompe dicha limitación, empleando con ello un conjunto de arquitecturas distintas para ejecutar una misma aplicación, de tal forma que cada arquitectura se encargue de aquellas tareas para las que se encuentra mejor preparada, obteniendo por ello una mejora notable en cuanto a rendimiento.

En el campo de computación heterogénea destaca el caso de CPU y GPU, pues su conjunto forma una excelente complementación. Mientras que la CPU se encuentra optimizada para tareas dinámicas de ráfagas de cómputo cortas y un flujo de control impredecible, la GPU se especializa justamente en el caso contrario: ráfagas de cómputo altamente demandantes pero con un flujo de control simple.

De esta forma, si una tarea presenta un número reducido de datos, una lógica de control sofisticada y un bajo nivel de paralelismo, se asignará a la CPU. Si por el contrario esta presenta una cantidad exuberante de datos, así como un alto grado de paralelismo en ellos, se asignará a la GPU pues presenta un gran número de núcleos y puede dar soporte a una cantidad de

hebras mucho mayor que la posible mediante CPU. [6]

Tal y como se explicará en detalle en secciones posteriores, el patrón de entrenamiento en redes neuronales convolucionales es computacionalmente intensivo y altamente paralelo [7]. Por ello, se adoptará un enfoque de computación heterogénea, con el propósito de acortar los tiempos de ejecución requeridos en dichos entrenamientos.

1.2. Objetivos

El principal objetivo de este proyecto es diseñar y desarrollar redes neuronales convolucionales (CNNs) desde sus cimientos, a un nivel muy básico. Esto permite una profunda comprensión de sus fundamentos y funcionamiento, comunes a bibliotecas especializadas en el campo. Para ello, se desarrollan distintas implementaciones sobre el mismo software, cada una con mejores prestaciones que la anterior.

La principal razón de este proyecto es aprender los fundamentos del machine learning aplicados a redes neuronales convolucionales, así como el diseño y desarrollo de sistemas heterogéneos de altas prestaciones, y el uso de librerías de bajo nivel del ámbito como cudnn.

A continuación se desglosan en dos categorías los objetivos específicos que permiten alcanzar el objetivo principal. Los objetivos de aprendizaje se centran en la adquisición de los conocimientos teóricos requeridos para la concepción de este proyecto, mientras que los objetivos de diseño y desarrollo buscan llevar a la práctica dicho conocimiento teórico adquirido anteriormente, aportando con ello una experiencia de aprendizaje de mayor categoría.

1.2.1. Objetivos de aprendizaje

1. **OA.1** Estudiar los fundamentos del machine learning y como se aplican a CNNs.
2. **OA.2** Estudiar los distintos componentes de una CNN y la conexión entre los mismos.
3. **OA.3** Estudiar implementaciones similares a las planteadas en este proyecto para comprender y analizar las funcionalidades y propiedades que se requieren.
4. **OA.4** Estudiar como diseñar e implementar CNNs empleando distintas tecnologías de programación como opencv o C++, entre otras.
5. **OA.5** Estudiar como diseñar e implementar CNNs empleando paralelización a nivel de cpu mediante OpenMP y C++.
6. **OA.6** Estudiar como diseñar e implementar sistemas heterogéneos empleando CUDA.

7. **OA.7** Estudiar como diseñar e implementar CNNs mediante sistemas heterogéneos.
8. **OA.8** Estudiar como diseñar e implementar CNNs mediante librerías de bajo nivel como cuDNN.

1.2.2. Objetivos de diseño y desarrollo

1. **ODD.1** Diseñar e implementar CNNs a bajo nivel mediante C++.
2. **ODD.2** Diseñar e implementar CNNs a bajo nivel mediante C++ y paralelización a nivel de cpu mediante openmp.
3. **ODD.3** Diseñar e implementar CNNs a bajo nivel como sistema heterogéneo mediante C++ y CUDA.
4. **ODD.4** Diseñar e implementar CNNs mediante la librería de bajo nivel cuDNN.

Capítulo 2

Conceptos previos

2.1. Machine Learning

Se entiende como el campo de las ciencias de computación que en vez de enfocarse en el diseño de algoritmos explícitos, optan por el estudio de técnicas de aprendizaje. Este enfoque tiene un gran éxito en tareas computacionales donde no es factible diseñar un algoritmo de forma explícita. [7] En vez de averiguar las distintas reglas a seguir para llegar a una solución, esta alternativa permite simplemente suministrar ejemplos de lo que debería pasar en distintas situaciones, y dejar que la máquina aprenda y extraiga ella misma sus propias conclusiones. De esta forma, el procedimiento en aprendizaje supervisado consiste en 'entrenar' con una muestra de N ejemplos, extraer información de ellos, y posteriormente poder evaluar de forma 'correcta' (bajo un margen de error controlado) otra muestra de M ejemplos, siendo $M > N$. [8]

Este enfoque ha contribuido en el avance de áreas como reconocimiento de voz, visión por ordenador, procesamiento de lenguaje natural, etc.

2.2. Componentes necesarios para el aprendizaje supervisado

Datos de entrada X y de salida Y que el modelo empleará para aprender y tomar decisiones. Ambos se unen para formar un dataset de entradas-salidas $D=\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$. Para que el aprendizaje sea posible, debe existir una función $F: X \rightarrow Y$ tal que $y_i = F(x_i)$ para $i \in \{1 \dots N\}$. De esta forma, en función del dataset D, el modelo tratará de encontrar una función G que aproxime F para dicho conjunto. Además, se suelen aplicar técnicas que permitan una mejor generalización del modelo, expandiendo las capacidades del mismo y permitiendo que su conocimiento pueda ser útil incluso fuera de la muestra de datos inicial. [8]

2.3. División de datos en entrenamiento y test

Para visualizar la generalización del modelo, el conjunto de datos D se suele dividir en 2 subconjuntos, (entrenamiento y test) de forma que se pueda estimar si realmente 'aprende' o solo memoriza.

Una vez realizada la división, se entrena el modelo con los datos del conjunto de entrenamiento. Cuando se termina el entrenamiento, se accede al conjunto test y se visualiza el rendimiento del modelo sobre el mismo. Como los datos de test no se emplearon en ningún momento, aportan una estimación sobre la generalización del modelo fuera de la muestra con la que se entrenó.

2.4. Tipos de aprendizaje

2.4.1. Aprendizaje Supervisado

Es el que se empleará en este proyecto. Se caracteriza por la presencia de una etiqueta 'correcta' y_i asociada a cada dato de entrada x_i . Posteriormente, la red empleará ambos valores para, a partir de x_i , tratar de deducir y_i . [8]

Aunque se trate de impedir, la existencia de ruido en los datos es inevitable, implicando que algunas etiquetas de $Y=\{y_1, y_2, \dots, y_N\}$ puedan ser erróneas. Este tipo de aprendizaje se divide a su vez en problemas de clasificación y regresión, centrándose en predecir etiquetas o valores numéricos, respectivamente.

2.4.2. Aprendizaje No Supervisado

En este tipo de aprendizaje los datos no contienen ninguna información respecto a Y. De esta forma, el conjunto de datos D se compone exclusivamente de valores $X=\{x_1, x_2, \dots, x_N\}$. [8]

2.4.3. Aprendizaje Por Refuerzo

En este caso tampoco existe un y_i 'correcto' asociado a cada x_i . En su lugar, se asocia a cada x_i una etiqueta con un valor posible de y_i , además de una medida que indica como de bueno es el mismo. [8]

2.5. Redes Neuronales Totalmente Conectadas

2.5.1. Neurona

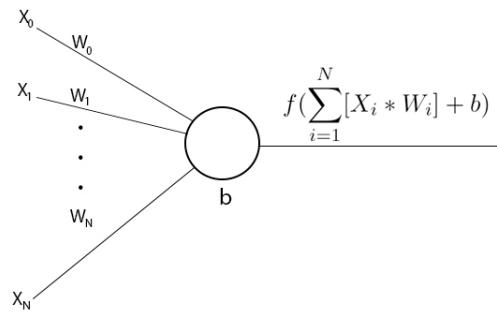


Figura 2.1: Imagen de una neurona

Una neurona parte de una serie de datos de entrada $X = \{x_1, x_2, \dots, x_N\}$ tal que cada $x_i \in X$ se encuentra asociado a un peso $w_i \in W$.

Esta los emplea para realizar una suma ponderada y posteriormente añadir un sesgo b , además de aplicar una función de activación f sobre el resultado obtenido.

2.5.2. Estructura por capas

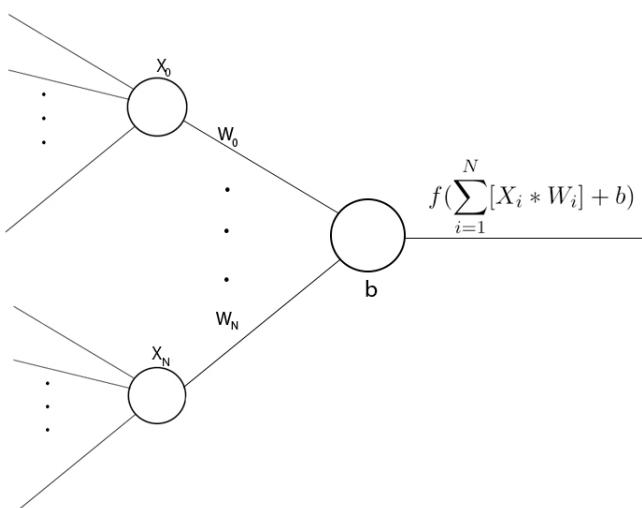


Figura 2.2: Imagen de una capa de neuronas

Las neuronas se suelen agrupar por capas, de tal forma que la salida de una compone la entrada de la siguiente, formando así modelos más sofisticados.

2.5.3. Funciones de activación

ReLU

$$\text{ReLU}(x) = \max(0, x) \quad (2.1)$$

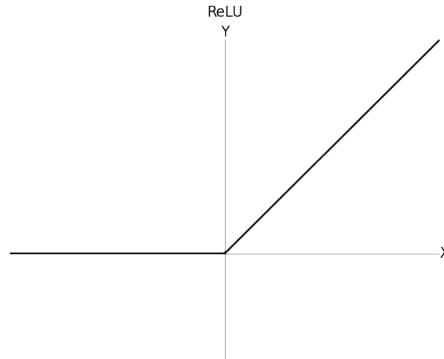


Figura 2.3: Imagen de la función de activación ReLU

A cambio de un bajo coste computacional, aporta no linealidad a la neurona, permitiendo a esta aprender funciones de mayor complejidad. Como su gradiente es 0 o 1, evita una reducción excesiva del mismo para valores positivos, mitigando así el problema del desvanecimiento del gradiente, caracterizado por la presencia de gradientes muy pequeños en backpropagation y provocar un aprendizaje lento. [9]

Sigmoide

$$\text{sigmoide}(x) = \frac{1}{1 + e^{-x}} \quad (2.2)$$

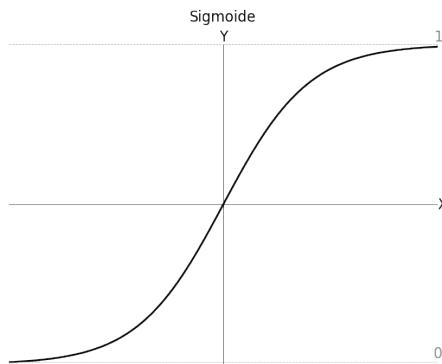


Figura 2.4: Imagen de la función de activación Síntesis

Se trata de una función interesante en el ámbito de la clasificación binaria, pues se caracteriza por transformar un valor de entrada en una salida comprendida en el rango [0-1].

Aunque sea monótona creciente y diferenciable en todos los puntos, tiende a saturarse con valores extremos (positivos o negativos). Por tanto, su aplicación dependerá del caso concreto a tratar. [10]

2.5.4. Capa SoftMax

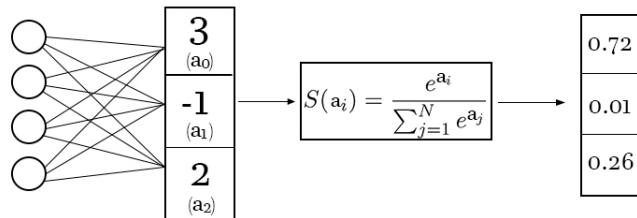


Figura 2.5: Imagen de la función de activación SoftMax

Para n entradas, produce n salidas con valores en el rango [0-1] que mantienen la proporción de entrada y cuya suma es 1. Por tanto, cada salida i se puede interpretar como la probabilidad de pertenencia a la clase i, siendo especialmente útil en clasificación multiclas. [11]

De esta forma, en la Figura 2.5 se muestra un ejemplo de clasificación multiclas con 3 clases distintas. Para una entrada dada, el modelo estima que esta pertenece a la clase 0, pues este según este, dicha entrada presenta un 0.72 % de probabilidades de pertenecer a la clase 0, un 0.01 % a la clase 1, y un 0.26 % a la clase 2.

2.5.5. Función de error o pérdida

Entropía Cruzada

$$E(y, \hat{y}) = - \sum_{i=1}^H [y_i * \log(\hat{y}_i)] \quad (2.3)$$

Es una métrica empleada en aprendizaje automático para medir qué tan bien se desempeña un modelo de clasificación. La pérdida o error se mide como un valor en el rango [0-1], siendo 0 un modelo perfecto y 1 otro totalmente erróneo. [12]

En la fórmula 2.3 se muestra el cálculo de la misma, donde H es el número de clases al que puede pertenecer cada dato de entrada $x_i \in X$, y_i representa la etiqueta real de la entrada x_i empleada, y \hat{y}_i representa la etiqueta que el modelo predijo para la entrada x_i .

2.5.6. Descenso del gradiente

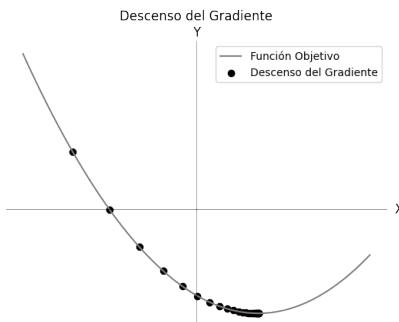


Figura 2.6: Ejemplo de funcionamiento del descenso del gradiente

Es un método de optimización iterativo que busca el mínimo local en una función diferenciable. En la figura 2.6 se muestra un ejemplo del mismo, donde cada punto representa una iteración del algoritmo.

Su nombre viene del término 'gradiente', siendo este una generalización multivariante de la derivada y denominado por el símbolo ∇ . Para una función f y un punto p , este indica la dirección del máximo incremento en la misma. El descenso del gradiente usa esta información para, una vez obtenido el gradiente, desplazarse en dirección contraria, es decir, en dirección del mínimo. Además, la distancia que se recorre en cada iteración viene dada por un hiperparámetro denominado "learning rate" o α [13] [14] [15].

A diferencia de los parámetros de un modelo (pesos y sesgos), sus hiperparámetros son establecidos por el usuario y no se "aprenden" durante el entrenamiento del modelo.

Entrenamiento

De esta forma, el procedimiento para entrenar una red neuronal consiste en, para una entrada x_i y una etiqueta asociada y_i , emplear x_i para producir una predicción \hat{y}_i (**ForwardPropagation** o Propagación hacia delante) que posteriormente se podrá comparar con y_i mediante una función de error $H(x)$ y obtener una medida de lo buena o mala que fue la misma. Una vez obtenido dicho “error”, se aplica el algoritmo del descenso del gradiente para cada parámetro de la red (**BackPropagation** o retropropagación) [12].

$$W_{t+1} = W_t - \alpha * \frac{\partial H(x)}{\partial W_t} \quad (2.4)$$

$$b_{t+1} = b_t - \alpha * \frac{\partial H(x)}{\partial b_t} \quad (2.5)$$

Así, se actualizarán los parámetros de la red neuronal según las fórmulas 2.4 y 2.5. En ellas, W_t y b_t indican los valores del peso W y bias b en el instante o iteración t, de la misma forma que W_{t+1} y b_{t+1} representan los valores de los mismos en el instante t+1 [16].

Descenso del gradiente estocástico

Algorithm 1 Descenso del gradiente estocástico [17]

```

Datos de entrenamiento  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ .
for época  $p \in \{0, \dots, P - 1\}$  do
    Desordenar vector de datos D.
    for  $i \in [m_{\text{inicio}}, \dots, m_{\text{fin}}]$  do
        Obtener pareja de datos  $(x_i, y_i) \in D$ 
        Realizar propagación hacia delante de  $x_i$  y obtener predicción  $\hat{y}_i$ .
        Evaluar  $\hat{y}_i$  con la función de pérdida y calcular el error  $e_i$ .
        Realizar propagación hacia detrás y obtener gradientes
        de cada parámetro del modelo.
        Actualizar parámetros.
    end for
end for

```

Es una variante que sustituye el gradiente real por una estimación del mismo, logrando reducir la carga computacional y tiempo de entrenamiento a cambio de una menor tasa de convergencia [18] [19].

Se caracterizada por, en cada época, dividir el conjunto de entrenamiento

en varios subconjuntos aleatorios y disjuntos entre ellos (mini-batch), de tal forma que se calcule el gradiente y actualicen los parámetros del modelo en cada uno de ellos [18].

2.5.7. Inicialización de pesos y sesgos

Inicialización de pesos

Siguiendo la bibliografía [20] [21], se empleará ReLU como función de activación en las capas intermedias del modelo. Por tanto, tal y como se indica en la bibliografía, se inicializan los pesos mediante la “inicialización He” o “inicialización Kaiming He”. Esta consiste en, para un peso w , inicializarlo según una distribución gaussiana con una media de 0.0 y una desviación típica de $\sqrt{\frac{2}{N_in}}$, siendo N_in el número de neuronas en la capa de entrada [22] [23] [24].

Inicialización de sesgos

De la misma forma, se vuelve a seguir la bibliografía, indicando esta vez una inicialización de sesgos a 0.0 [25] [26].

2.5.8. Tipos de codificaciones

En el campo de machine learning existen varios tipos de codificaciones. De esta forma, para codificar 3 clases distintas se podrían codificar o bien mediante $\{1, 2, 3\}$ (codificación de etiquetas), o mediante $\{100, 010, 001\}$ (codificación one-hot), por ejemplo. En este proyecto se empleará la codificación one-hot, pues aporta unas ventajas que se contemplarán con detalle en secciones posteriores.

2.5.9. Propagación hacia delante con softmax

Suponemos que para un x_i dado, se obtiene $S(\hat{y}) = [S(\hat{y}_1), S(\hat{y}_2), S(\hat{y}_3)] = [0.04, 0.7, 0.26]$

Para dicho x_i , $y_i = [0, 0, 1]$

En este caso, el modelo cree que x_i pertenece a la clase 2 (0.7 es el mayor número del vector $S(\hat{y})$). Sin embargo, y_i indica que x_i pertenece a la clase 3.

Se calcula el error de entropía cruzada:

$$E(y, S(\hat{y})) = -(0 * \log(0.04) + 0 * \log(0.7) + 1 * \log(0.26)) \quad (2.6)$$

$$E(y, S(\hat{y})) = -\log(0.26) = 0.585 \quad (2.7)$$

[1]

2.6. Redes Neuronales Convolucionales

2.6.1. Introducción a CNNs

Las redes neuronales convolucionales (CNN, Convolutional Neural Network) se caracterizan por trabajar con volúmenes de datos 3D. En este proyecto, se emplearán imágenes RGB como entrada para cada CNN. Por tanto, por simplificar la nomenclatura, cada capa 2D de un volumen 3D se denominará como imagen, pues una imagen RGB se trata de 3 imágenes 2D, una por color. Además, la entrada de cada capa se define como X o $X(C^H^W)$ y en ambos casos cuenta con unas dimensiones de C^H^W , siendo C el número de canales de profundidad, H el número de filas y W el número de columnas de dicho volumen 3D. De la misma forma, se define el volumen de salida como $Y(M * H_{out} * W_{out})$. En cuanto a los pesos de la capa convolucional, cada filtro o kernel se denominará mediante un símbolo/s diferente/s (K, K_1, G , etc), pues cada uno puede presentar unas dimensiones diferentes. Aun así, se podrá definir a un filtro de pesos como $K_1(K^K)$, indicando que posee K filas y columnas. Además, también se trata de una estructura con 3 dimensiones. Sin embargo, no se requiere especificar el número de canales de profundidad de un kernel de pesos pues siempre será C (depende de X). De la misma forma, el número de filtros a aplicar sobre X recibe el nombre de M , y el tamaño de Y depende de este, pues tal y como se verá posteriormente, cada filtro genera una canal de profundidad distinto de la salida Y .

2.6.2. Capa convolucional

Componentes

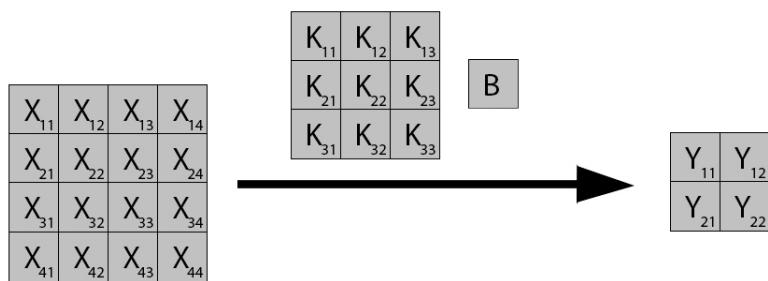


Figura 2.7: Componentes en una capa convolucional

Una capa convolucional parte de un volumen de entrada X , un kernel de filtros K , un sesgo B y una función de activación para, mediante una

convolución, obtener un volumen de salida Y [27] [28].

Propagación hacia delante

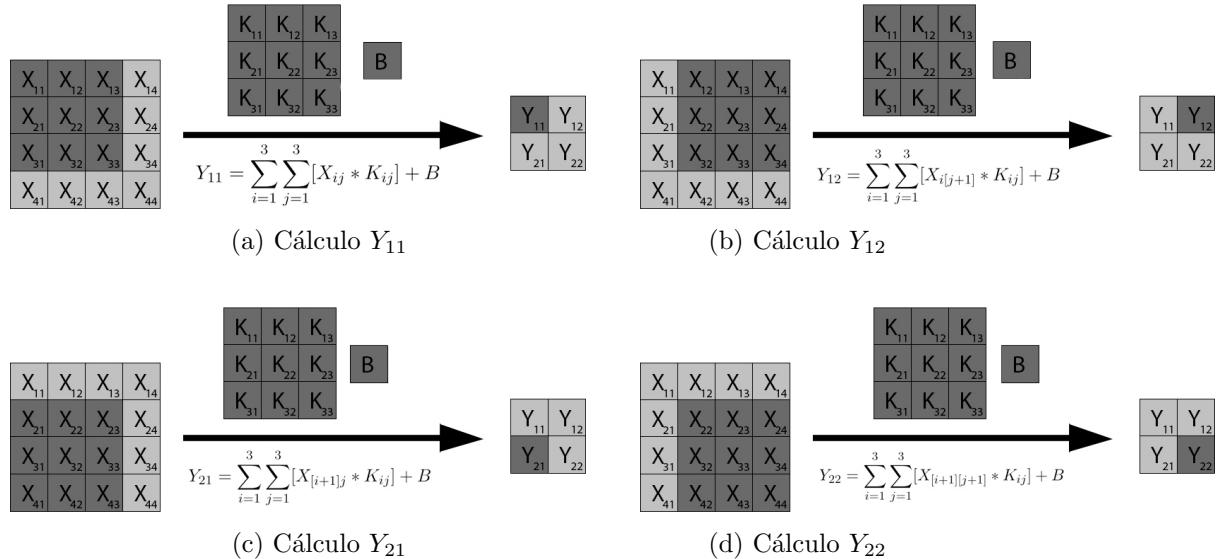


Figura 2.8: Propagación hacia delante en una capa convolucional

Una convolución entre 2 volúmenes de datos X y K , consiste en “deslizar K sobre X ” tal y como se muestra en la Figura 2.8. De esta forma, en cada “paso” se recorren ambos volúmenes, multiplicando los elementos de X y K que se encuentren en la misma posición. Posteriormente, se suma cada resultado obtenido, además de un sesgo B y finalmente aplicar una función de activación [27].

En la Figura 2.8 se emplea un volumen X con un solo canal de profundidad. Sin embargo, este no es el caso común. Por tanto, se denominará como X_{ij}^c al elemento de X que se encuentre en la posición (i,j) del canal de profundidad c [28].

Propagación hacia delante, canales de profundidad

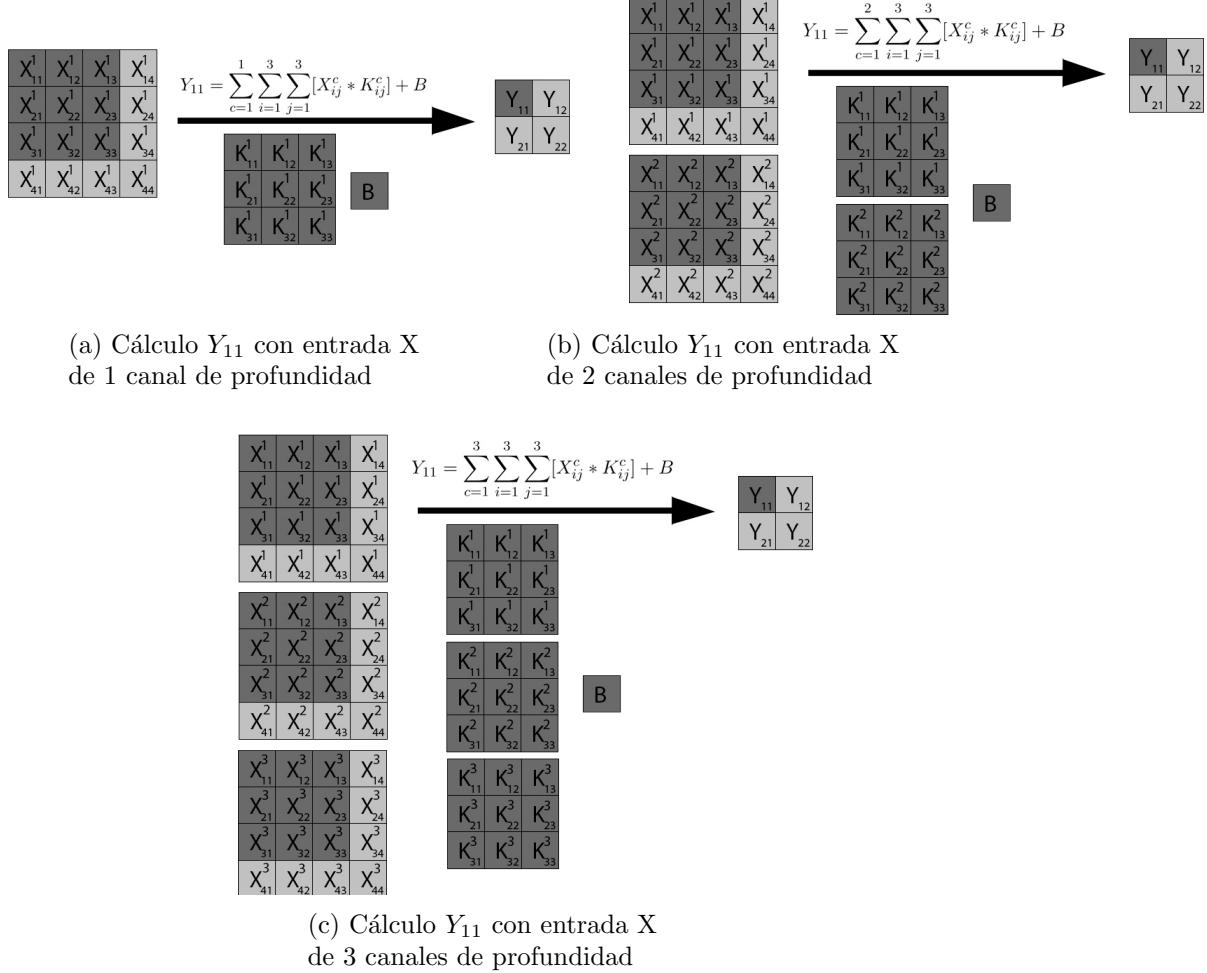


Figura 2.9: Propagación hacia delante en una capa convolucional con varios canales de profundidad

De esta forma, en la Figura 2.9 se muestra como una convolución con C canales de profundidad se descompone en la suma de C convoluciones con un canal de profundidad.

Para una entrada X con C canales de profundidad, convolución(X, K) = convolución(X^1, K^1) + convolución(X^2, K^2) + ... + convolución(X^C, K^C). Por último, en cada “paso” del “deslizamiento” se suma un solo sesgo y se aplica una sola vez la función de activación, independientemente del número de canales de profundidad que presente la entrada X .

Propagación hacia delante, número de filtros

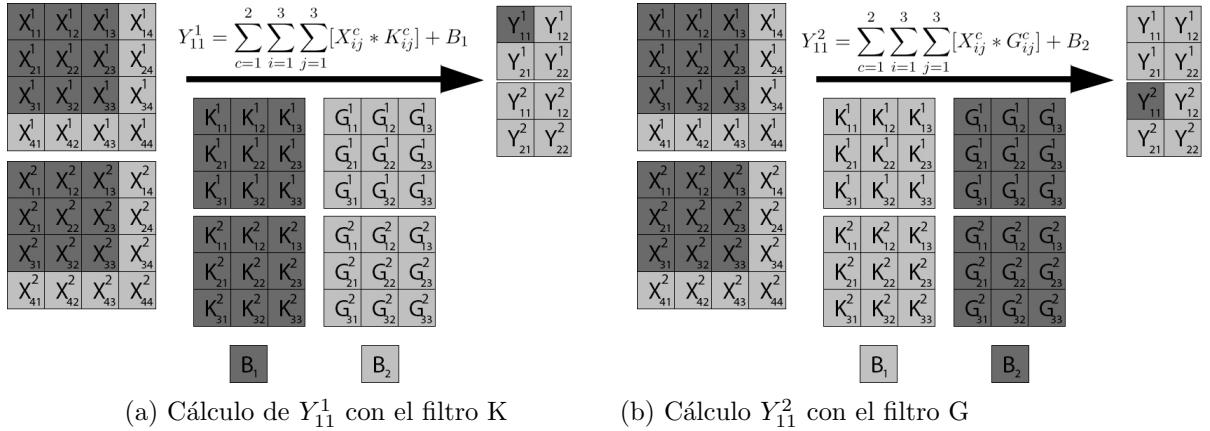


Figura 2.10: Propagación hacia delante en una capa convolucional con varios filtros

Cada convolución entre dos volúmenes 3D produce un volumen de salida 2D. Por tanto, al aplicar N convoluciones entre un volumen de entrada X y una serie de filtros $K=\{K_1, K_2, \dots, K_N\}$, se obtendrá un volumen 3D de salida con tantas capas de profundidad como convoluciones se aplicaron (N).

En la Figura 2.10 se observa como al aplicar $N=2$ convoluciones sobre la misma entrada X (una con el filtro K y otra con el filtro G) se obtiene un volumen de salida con 2 capas de profundidad [28].

Relleno o “Padding”

En la figura 2.8 se visualiza como al realizar una convolución entre un volumen X con dimensiones $1 \times 4 \times 4$ y un kernel de pesos K con dimensiones $1 \times 3 \times 3$, el resultado obtenido es un volumen Y de dimensiones $1 \times 2 \times 2$. La reducción de dimensionalidad es un problema pues afecta directamente al número de convoluciones que se pueden aplicar sobre un volumen. Por tanto, el “relleno” o “padding” se aplica antes de realizar una convolución y es una técnica empleada para conservar las dimensiones espaciales de un volumen de entrada X , expandiendo cada canal del mismo tal y como se muestra en la figura 2.11 [29].

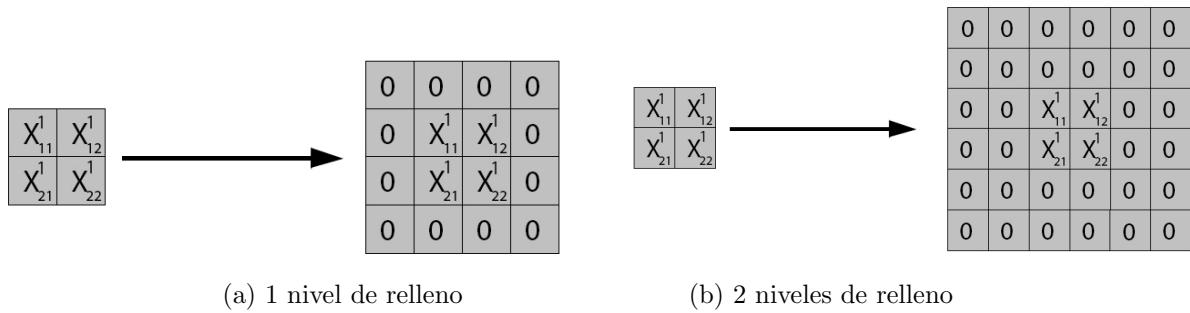


Figura 2.11: Relleno sobre un volumen de entrada X

Figura 2.11. Reñido sobre un volumen de entrada X

De esta forma, un relleno a un nivel sobre un volumen añadirá sobre el mismo dos filas y dos columnas con valores igual a 0. Un relleno a dos niveles añadirá cuatro filas y columnas con valores igual a cero, ..., un relleno a n niveles añadirá 2^n filas y columnas con valores igual a 0.

Relleno completo

Se denomina como relleno completo aquel que asegura que cada valor o casilla de X sea visitada el mismo número de veces que el resto [30].

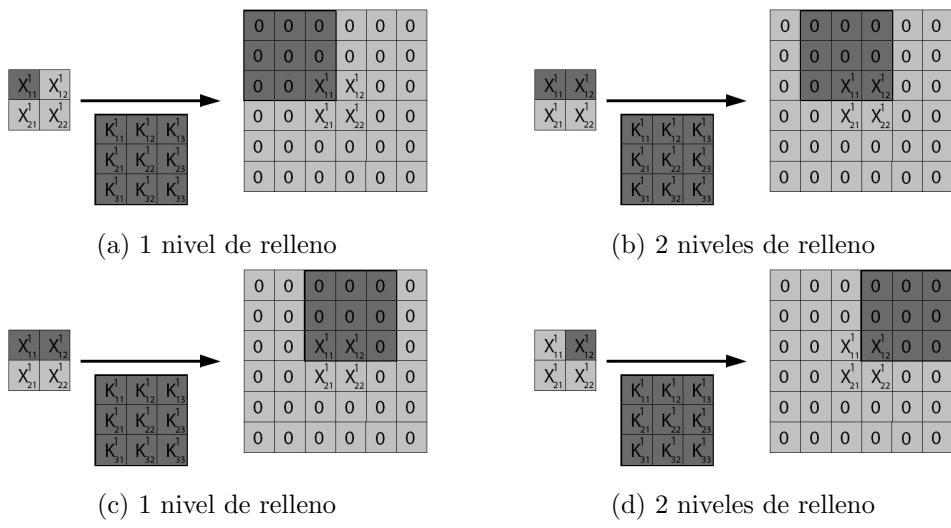


Figura 2.12: Relleno sobre un volumen de entrada X

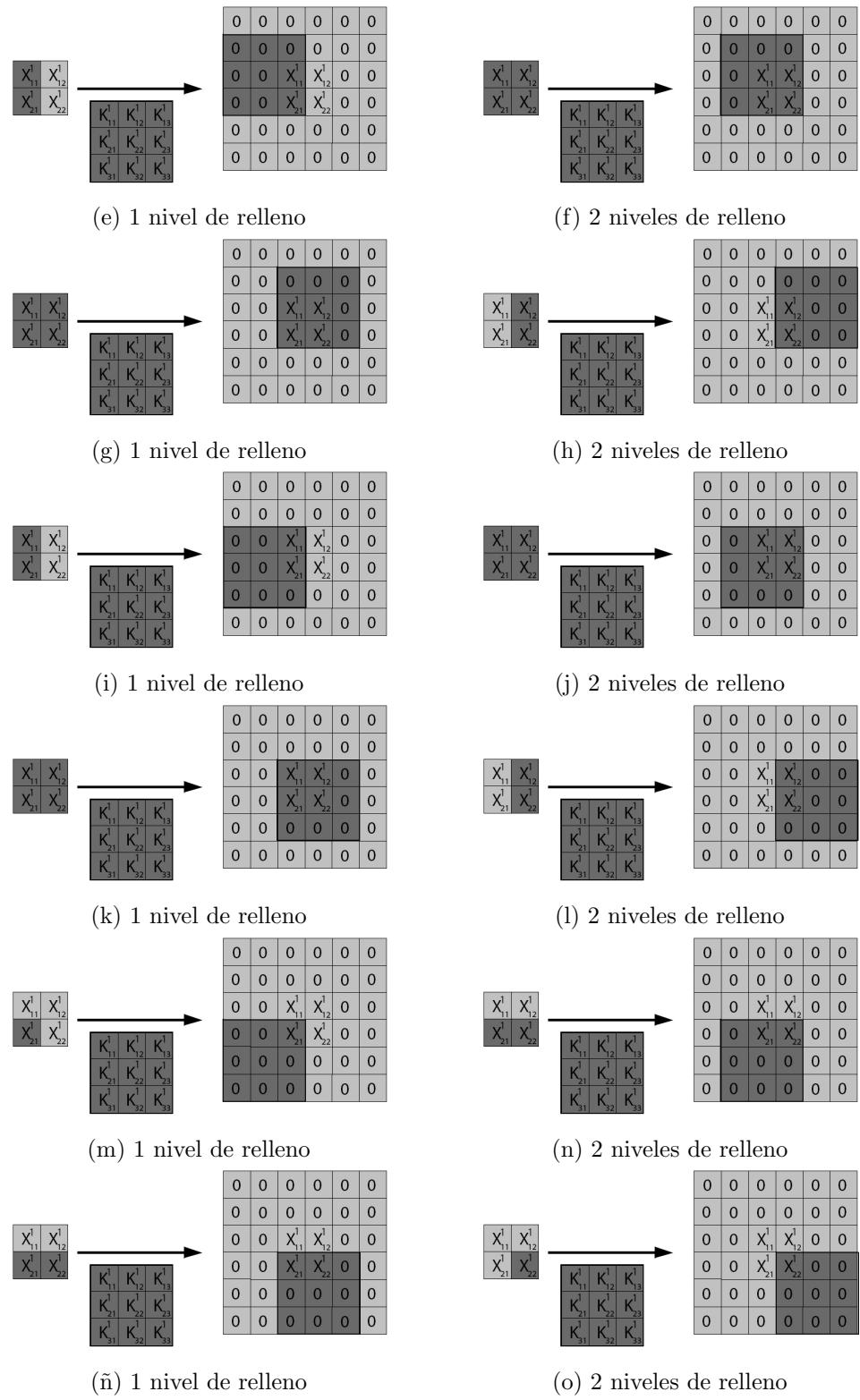


Figura 2.12: Relleno sobre un volumen de entrada X

Tal y como se muestra en la figura 2.12, se realiza un relleno completo pues en la convolución entre X y K se accede a cada valor de X el mismo número de veces (9).

2.6.3. Capa de agrupación máxima

Componentes

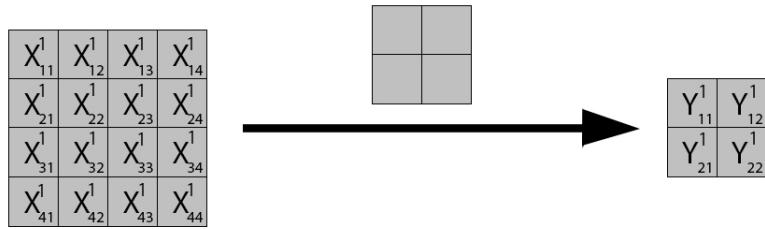


Figura 2.13: Componentes en una capa de agrupación máxima

Al igual que las capas convolucionales, las capas de agrupación máxima también presentan una “ventana” que se irá deslizando por el volumen de entrada. Sin embargo, el resultado en cada iteración viene dado por el valor máximo contenido en ella. Por tanto, no presenta parámetros asociados.

Propagación hacia delante

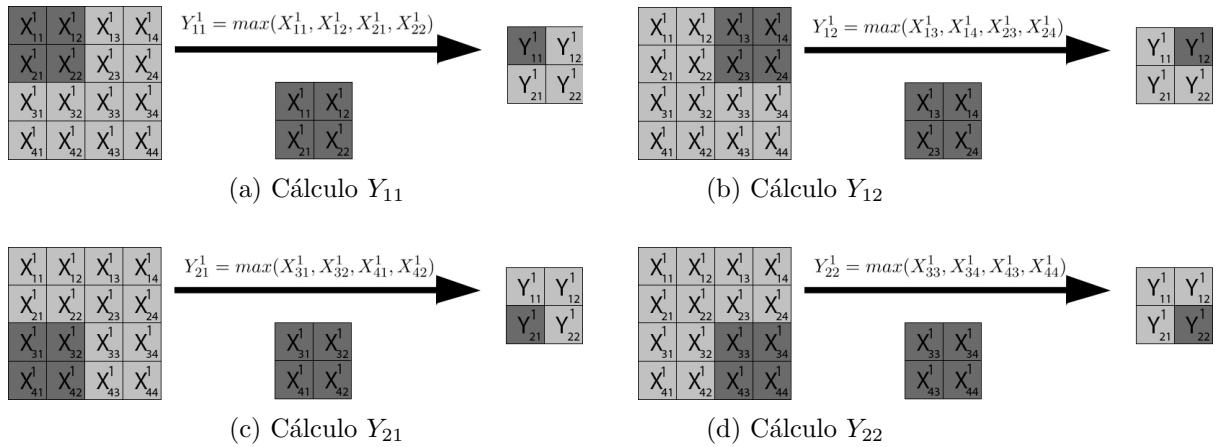


Figura 2.14: Propagación hacia delante en una capa de agrupación máxima

A diferencia de las capas convolucionales, las capas de agrupación máxima no comparten regiones del volumen de entrada entre distintas iteraciones. Esto se muestra en las figuras 2.14 y 2.8.

Propagación hacia delante, canales de profundidad

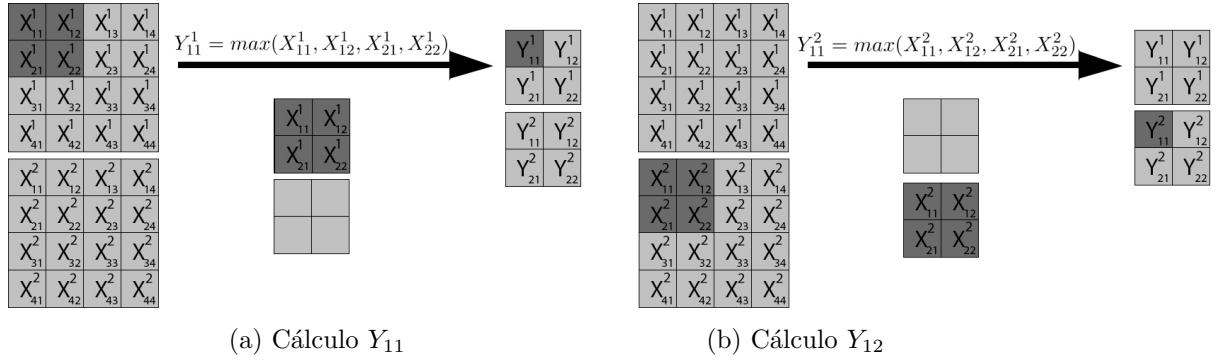


Figura 2.15: Propagación hacia delante en una capa de agrupación máxima

Tal y como se muestra en la figura 2.15, cada “ventana” se desliza sobre el volumen de entrada en un canal de profundidad distinto.

Propagación hacia detrás

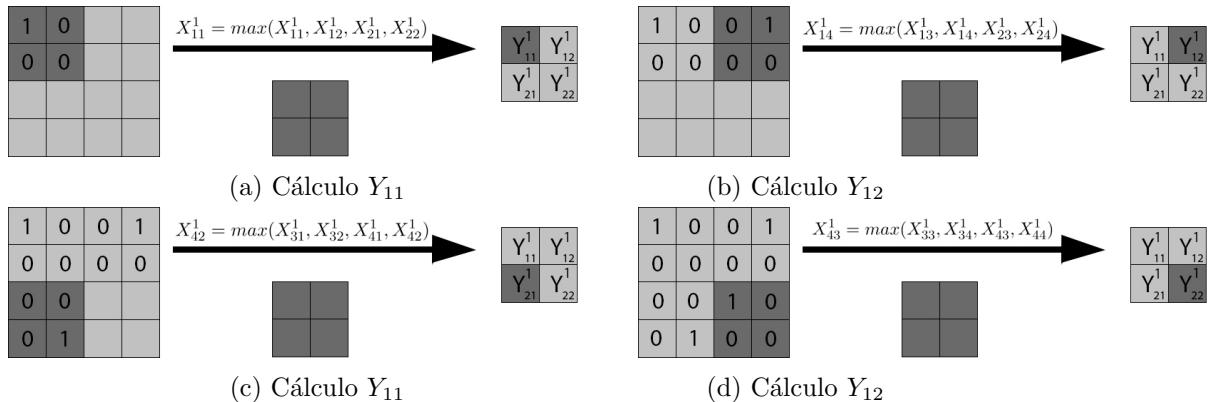


Figura 2.16: Propagación hacia delante en una capa de agrupación máxima

Para la primera iteración de la figura 2.16, en la propagación hacia detrás se calcula el gradiente respecto al volumen de entrada de la siguiente forma:

$$\frac{\partial E}{\partial X_{11}^1} = \frac{\partial E}{\partial Y} * \frac{\partial Y}{\partial X_{11}^1} \quad (2.8)$$

$$\frac{\partial E}{\partial X_{12}^1} = \frac{\partial E}{\partial Y} * \frac{\partial Y}{\partial X_{12}^1} \quad (2.9)$$

$$\frac{\partial E}{\partial X_{21}^1} = \frac{\partial E}{\partial Y} * \frac{\partial Y}{\partial X_{21}^1} \quad (2.10)$$

$$\frac{\partial E}{\partial X_{22}^1} = \frac{\partial E}{\partial Y} * \frac{\partial Y}{\partial X_{22}^1} \quad (2.11)$$

Tomando el ejemplo de la figura 2.16 (a), se obtiene que $Y_{11}^1 = X_{11}^1$. Por tanto, las fórmulas anteriores se convierten en:

$$\frac{\partial E}{\partial X_{11}^1} = \frac{\partial E}{\partial Y} * \frac{\partial X_{11}^1}{\partial X_{11}^1} = \frac{\partial E}{\partial Y} * 1 = \frac{\partial E}{\partial Y} \quad (2.12)$$

$$\frac{\partial E}{\partial X_{12}^1} = \frac{\partial E}{\partial Y} * \frac{\partial X_{12}^1}{\partial X_{12}^1} = \frac{\partial E}{\partial X_{11}^1} * 0 \quad (2.13)$$

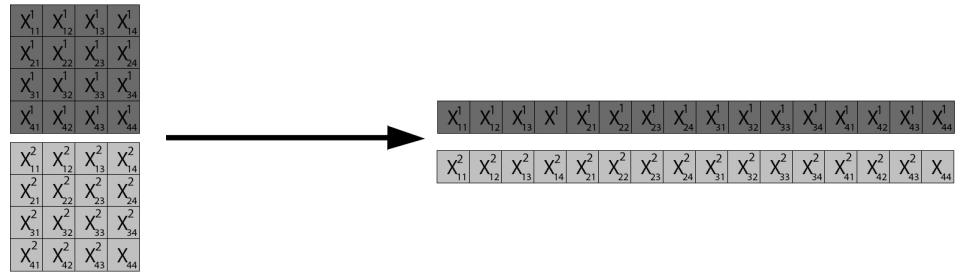
$$\frac{\partial E}{\partial X_{21}^1} = \frac{\partial E}{\partial Y} * \frac{\partial X_{21}^1}{\partial X_{21}^1} = \frac{\partial E}{\partial X_{11}^1} * 0 \quad (2.14)$$

$$\frac{\partial E}{\partial X_{22}^1} = \frac{\partial E}{\partial Y} * \frac{\partial X_{22}^1}{\partial X_{22}^1} = \frac{\partial E}{\partial X_{11}^1} * 0 \quad (2.15)$$

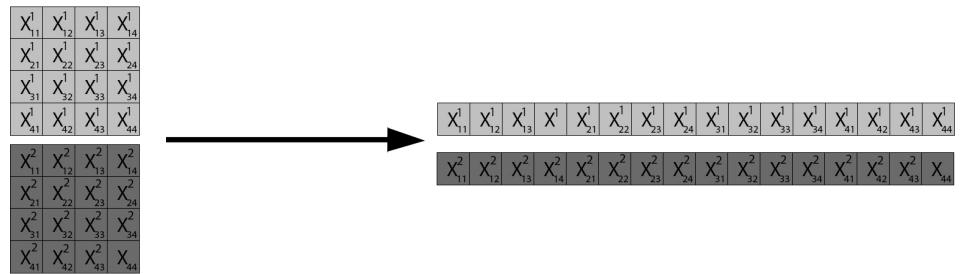
Como el resultado obtenido en cada iteración solo depende del valor máximo de la ventana, tiene sentido que la derivada de la salida Y respecto a la entrada X sea igual a 1 solo en dicho caso y 0 en el resto [31] [32].

2.6.4. Capa de aplanado

Propagación hacia delante



(a) Propagación hacia delante de la capa de aplanado con la primera capa de profundidad



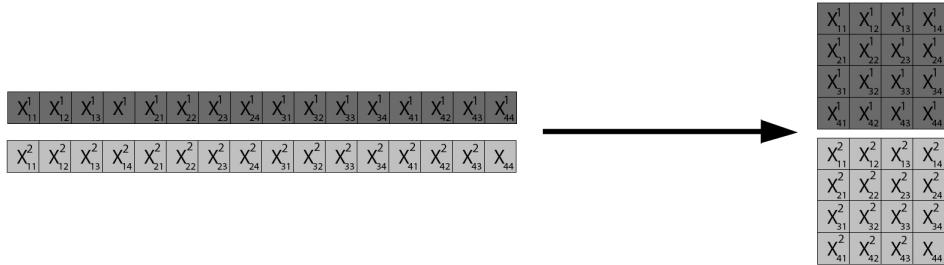
(b) Propagación hacia delante de la capa de aplanado con la segunda capa de profundidad

Figura 2.17: Propagación hacia delante en una capa de agrupación máxima

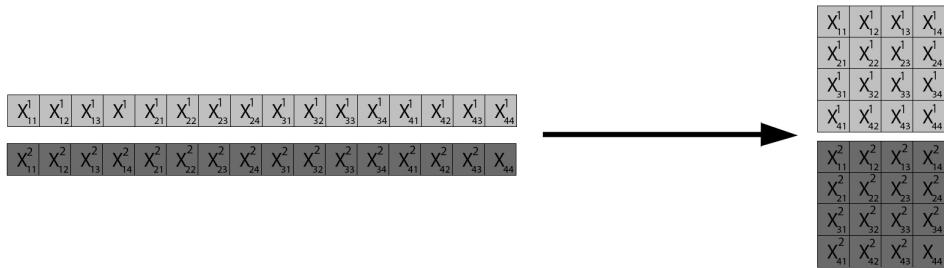
La capa de aplanado tiene como objetivo la creación de un “enlace” entre las capas de convolución y agrupación con la capas totalmente conectadas. En la propagación hacia delante, parte de un volumen de entrada 3D para, mediante un “aplanado”, convertirlo en un vector 1D que pueda ser usado como entrada para una red totalmente conectada.

Como solo se modifica la forma en la que se agrupan los datos, no requiere la presencia de parámetros [33].

Propagación hacia detrás



(a) Propagación hacia detrás de la capa de aplanado con la primera capa de profundidad



(b) Propagación hacia detrás de la capa de aplanado con la segunda capa de profundidad

Figura 2.18: Propagación hacia delante en una capa de agrupación máxima

En la propagación hacia detrás, se parte de un array 1D y se convierte en un volumen 3D que pueda ser usado como entrada para una capa convolucional o de agrupación [33].

2.7. cuDNN (Deep Neural Network)

Es una librería de primitivas acelerada por GPU para redes neuronales profundas. Proporciona implementaciones altamente optimizadas para rutinas estándar como la propagación hacia delante y hacia detrás en capas convolucionales, de agrupación máxima, e incluso con funciones de activación como ReLU o sigmoide, entre otras [34].

Su meta es obtener el mejor rendimiento posible en GPUs de NVIDIA para casos importantes de aprendizaje profundo. Dados sus buenos resultados, se emplea en gran cantidad de frameworks de aprendizaje profundo, siendo algunos de ellos Caffe2, Keras, MATLAB, Pytorch, o TensorFlow [35].

Está diseñada para ser utilizada en aplicaciones de aprendizaje profundo

que requieran un poder computacional intensivo, permitiendo a desarrolladores e investigadores aprovechar al máximo las prestaciones de las GPUs de NVIDIA. Entre ellas, destaca la compatibilidad con múltiples arquitecturas de GPU, su optimización de la memoria, y su flexibilidad y facilidad de integración. Por todo esto y más, se ha convertido en un componente esencial en el desarrollo de soluciones de inteligencia artificial, facilitando la investigación y la innovación en el campo del aprendizaje profundo.

2.7.1. Manejador

cuDNN asume que los datos necesarios se encuentran ya en GPU y accesibles desde device. Se hablará de esto con más detalle en la sección sobre cuda y GPU.

Una aplicación que use cuDNN requiere de la inicialización de un manejador o handle. Dicho manejador será requerido por cuDNN en cada operación que se quiera realizar con dicha librería, permitiendo al usuario un control explícito sobre el funcionamiento de la misma aunque este emplee múltiples hebras o GPUs.

Por ejemplo, en el caso de múltiples GPUs, se pueden asociar diferentes dispositivos con diferentes hebras del host, de forma que cada una tenga un manejador de cuDNN distinto. Así, las llamadas a cuDNN con distinto manejador se ejecutarán en distinta GPU [36].

2.7.2. Tensores

Las operaciones en cuDNN reciben tensores como entrada y producen tensores como salida [36].

Tensor 3D

Un tensor 3D se suele emplear para representar un volumen 3D como podría ser una imagen RGB. Sus dimensiones se describen mediante 3 letras: B, M y N.

1. **B:** Tamaño del batch
2. **M:** Filas por imagen 2D
3. **N:** Columnas por imagen 2D

Tensor 4D

Suele representar conjuntos de imágenes 2D. Sus dimensiones son N, C, H, W

1. **N:** Tamaño del batch

2. **C:** Número de imágenes 2D
3. **H:** Filas por imagen 2D
4. **W:** Columnas por imagen 2D

cuDNN permite varios formatos pero usaremos NCHW por compatibilidad con el resto de implementaciones desarrolladas en este proyecto.

Representación de un tensor 4D

Dimensiones

N = 1
C = 2
H = 5
W = 6

C = 1

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

C = 2

31	32	33	34	35	36
37	38	39	40	41	42
43	44	45	46	47	48
49	50	51	52	53	54
55	56	57	58	59	60

Figura 2.19: Ejemplo de tensor 4D con dimensiones: N=1, C=1, H=5, y W=6

En la figura 2.19 se muestra un conjunto de imágenes 3D con las siguientes dimensiones:

1. **N:** Tamaño del batch, 1
2. **C:** Número de imágenes 2D o canales por imagen 3D, 2
3. **H:** Filas por imagen 2D, 5
4. **W:** Columnas por imagen 2D, 6

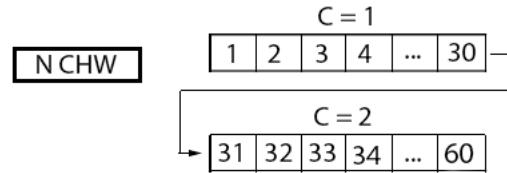


Figura 2.20: Ejemplo de tensor 4D NCHW con dimensiones: N=1, C=1, H=5, y W=6

En la figura 2.20 se muestra como se almacena en memoria el volumen de datos de la figura 2.19 según el formato NCHW. Esto es, por cada imagen 3D $n \in \{0, \dots, N\}$, almacenar cada canal $c \in \{0, \dots, C\}$ según una ordenación por filas.

2.7.3. Principales funciones

A continuación se mencionarán las principales funciones de cuDNN que se han empleado en este proyecto, siendo las responsables tanto de la propagación hacia delante como de la retropropagación en capas convolucionales y de agrupación máxima, entre otras.

cudnnConvolutionForward

Realiza la propagación hacia delante en una capa convolucional.

```
cudnnStatus_t cudnnConvolutionForward(
    cudnnHandle_t                  handle,
    const void                      *alpha,
    const cudnnTensorDescriptor_t   xDesc,
    const void                      *x,
    const cudnnFilterDescriptor_t   wDesc,
    const void                      *w,
    const cudnnConvolutionDescriptor_t convDesc,
    cudnnConvolutionFwdAlgo_t       algo,
    void                            *workSpace,
    size_t                           workSpaceSizeInBytes,
    const void                      *beta,
    const cudnnTensorDescriptor_t   yDesc,
    void                            *y)
```

1. **handle:** Manejador.
2. **alpha, beta:** Punteros a escalares empleados para combinar los resultados con valores anteriores tal que $\text{valor_final} = \text{alpha} * \text{result} + \text{beta} * \text{valor_anterior}$.

3. **xDesc:** Descriptor asociado al tensor de entrada.
4. **x:** Puntero a los datos de entrada en GPU asociados con el descriptor de tensor xDesc.
5. **wDesc:** Descriptor asociado al tensor de pesos.
6. **w:** Puntero a los pesos en GPU asociados con el descriptor de tensor wDesc.
7. **convDesc:** Descriptor de convolución.
8. **algo:** Especifica qué algoritmo de convolución aplicar.
9. **workSpace:** Puntero a un espacio de trabajo en memoria de GPU.
10. **workSpaceSizeInBytes:** Especifica el tamaño en bytes de workSpace.
11. **yDesc:** Descriptor asociado al tensor de salida.
12. **y:** Puntero a los datos de salida en GPU asociados con el descriptor de tensor yDesc.

[37]

cudnnPoolingForward

Se encarga de la propagación hacia delante en una capa de agrupación máxima.

```
cudnnStatus_t cudnnPoolingForward(
    cudnnHandle_t             handle,
    const cudnnPoolingDescriptor_t  poolingDesc,
    const void                 *alpha,
    const cudnnTensorDescriptor_t xDesc,
    const void                 *x,
    const void                 *beta,
    const cudnnTensorDescriptor_t yDesc,
    void                      *y)
```

1. **handle:** Manejador.
2. **poolingDesc:** Descriptor de la operación de agrupación.
3. **alpha, beta:** Punteros a escalares empleados para combinar los resultados con valores anteriores tal que valor_final = alpha*result + beta*valor_anterior.

4. **xDesc**: Descriptor asociado al tensor de entrada.
5. **x**: Puntero a los datos de entrada en GPU asociados con el descriptor de tensor xDesc.
6. **yDesc**: Descriptor asociado al tensor de salida.
7. **y**: Puntero a los datos de salida en GPU asociados con el descriptor de tensor yDesc.

[38]

cudnnPoolingBackward

Realiza la retropropagación en una capa de agrupación máxima.

```
cudnnStatus_t cudnnPoolingBackward(
    cudnnHandle_t                  handle,
    const cudnnPoolingDescriptor_t  poolingDesc,
    const void                      *alpha,
    const cudnnTensorDescriptor_t   yDesc,
    const void                      *y,
    const cudnnTensorDescriptor_t   dyDesc,
    const void                      *dy,
    const cudnnTensorDescriptor_t   xDesc,
    const void                      *xData,
    const void                      *beta,
    const cudnnTensorDescriptor_t   dxDesc,
    void                           *dx)
```

1. **handle**: Manejador.
2. **poolingDesc**: Descriptor de la operación de agrupación.
3. **alpha, beta**: Punteros a escalares empleados para combinar los resultados con valores anteriores tal que $\text{valor_final} = \text{alpha} * \text{result} + \text{beta} * \text{valor_anterior}$.
4. **yDesc**: Descriptor asociado al tensor de salida.
5. **y**: Puntero a los datos de salida en GPU asociados con el descriptor de tensor yDesc.
6. **dyDesc**: Descriptor asociado al tensor que almacena el gradiente de la pérdida respecto a los datos de salida.
7. **dy**: Puntero al gradiente de la pérdida respecto a los datos de salida en GPU asociados con el descriptor de tensor dyDesc.

8. **xDesc:** Descriptor asociado al tensor de entrada.
9. **x:** Puntero a los datos de entrada en GPU asociados con el descriptor de tensor xDesc.
10. **dxDesc:** Descriptor asociado al tensor que almacena el gradiente de la pérdida respecto a los datos de entrada.
11. **dx:** Puntero al gradiente de la pérdida respecto a los datos de entrada en GPU asociados con el descriptor de tensor dxDesc.

[38]

cudnnConvolutionBackwardFilter

Realiza la retropropagación respecto a los pesos en una capa convolucional.

```
cudnnStatus_t cudnnConvolutionBackwardFilter(
    cudnnHandle_t                      handle,
    const void*                         *alpha,
    const cudnnTensorDescriptor_t       xDesc,
    const void*                         *x,
    const cudnnTensorDescriptor_t       dyDesc,
    const void*                         *dy,
    const cudnnConvolutionDescriptor_t convDesc,
    cudnnConvolutionBwdFilterAlgo_t     algo,
    void*                               *workSpace,
    size_t                             workSpaceSizeInBytes,
    const void*                         *beta,
    const cudnnFilterDescriptor_t      dwDesc,
    void*                               *dw)
```

1. **handle:** Manejador.
2. **alpha, beta:** Punteros a escalares empleados para combinar los resultados con valores anteriores tal que $\text{valor_final} = \text{alpha} * \text{result} + \text{beta} * \text{valor_anterior}$.
3. **xDesc:** Descriptor asociado al tensor de entrada.
4. **x:** Puntero a los datos de entrada en GPU asociados con el descriptor de tensor xDesc.
5. **dyDesc:** Descriptor asociado al tensor que almacena el gradiente de la pérdida respecto a los datos de salida.

6. **dy**: Puntero al gradiente de la pérdida respecto a los datos de salida en GPU asociados con el descriptor de tensor dyDesc.
7. **convDesc**: Descriptor de convolución.
8. **algo**: Especifica qué algoritmo de convolución aplicar.
9. **workSpace**: Puntero a un espacio de trabajo en memoria de GPU.
10. **workSpaceSizeInBytes**: Especifica el tamaño en bytes de workSpace.
11. **dwDesc**: Descriptor del tensor asociado al gradiente de la pérdida respecto a los pesos.
12. **dw**: Puntero al gradiente de los pesos en GPU asociados con el descriptor de tensor dwDesc.

[39]

cudnnConvolutionBackwardData

Realiza la retropropagación respecto a los datos de entrada en una capa convolucional.

```
cudnnStatus_t cudnnConvolutionBackwardData(
    cudnnHandle_t                  handle,
    const void                     *alpha,
    const cudnnFilterDescriptor_t   wDesc,
    const void                     *w,
    const cudnnTensorDescriptor_t   dyDesc,
    const void                     *dy,
    const cudnnConvolutionDescriptor_t convDesc,
    cudnnConvolutionBwdDataAlgo_t   algo,
    void                           *workSpace,
    size_t                          workSpaceSizeInBytes,
    const void                     *beta,
    const cudnnTensorDescriptor_t   dxDesc,
    void                           *dx)
```

1. **handle**: Manejador.
2. **alpha, beta**: Punteros a escalares empleados para combinar los resultados con valores anteriores tal que $\text{valor_final} = \text{alpha} * \text{result} + \text{beta} * \text{valor_anterior}$.
3. **wDesc**: Descriptor asociado al tensor de pesos.

4. **w**: Puntero a los pesos en GPU asociados con el descriptor de tensor wDesc.
5. **dyDesc**: Descriptor asociado al tensor que almacena el gradiente de la pérdida respecto a los datos de salida.
6. **dy**: Puntero al gradiente de la pérdida respecto a los datos de salida en GPU asociados con el descriptor de tensor dyDesc.
7. **convDesc**: Descriptor de convolución.
8. **algo**: Especifica qué algoritmo de convolución aplicar.
9. **workSpace**: Puntero a un espacio de trabajo en memoria de GPU.
10. **workSpaceSizeInBytes**: Especifica el tamaño en bytes de workSpace.
11. **dxDesc**: Descriptor asociado al tensor que almacena el gradiente de la pérdida respecto a los datos de entrada.
12. **dx**: Puntero al gradiente de la pérdida respecto a los datos de entrada en GPU asociados con el descriptor de tensor dxDesc.

[40]

Capítulo 3

Aportaciones

3.1. Redes Neuronales Totalmente Conectadas

3.1.1. Gradiante de la función de pérdida respecto a Soft-Max, [1] [2]

Siendo Z_i la entrada i de la última capa, se denotará mediante O_i una vez se le aplique SoftMax.

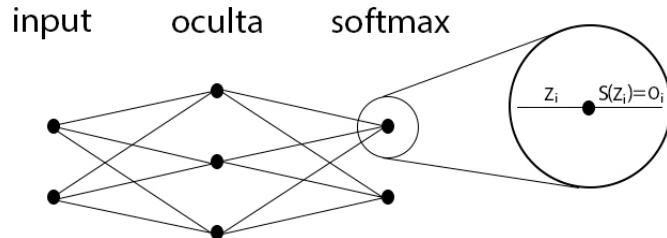


Figura 3.1: Estructura de una red con softmax

$$E = - \sum_{i=1}^H [y_i * \log(O_i)] \quad (3.1)$$

Según esta notación, la función de error ?? se convierte en la fórmula 3.1.

Gradiente de la función de error

$$\frac{\partial E}{\partial Z_k} = \frac{\partial (-\sum_{i=1}^H [y_i * \log(O_i)])}{\partial Z_k} = - \sum_{i=1}^H \left[\frac{\partial (y_i * \log(O_i))}{\partial Z_k} \right] \quad (3.2)$$

Como y_i es independiente respecto a Z_k , se trata como una constante.

$$\frac{\partial E}{\partial Z_k} = - \sum_{i=1}^H [y_i * \frac{\partial(\log(O_i))}{\partial Z_k}] \quad (3.3)$$

Se aplica la regla de la cadena, pues O_i no depende directamente de Z_k ,

$$\frac{\partial E}{\partial Z_k} = - \sum_{i=1}^H [y_i * \frac{\partial(\log(O_i))}{\partial O_i} * \frac{\partial O_i}{\partial Z_k}] \quad (3.4)$$

$$\frac{\partial E}{\partial Z_k} = - \sum_{i=1}^H [\frac{y_i}{O_i} * \frac{\partial O_i}{\partial Z_k}] \quad (3.5)$$

Derivada de softmax respecto de su entrada, $\frac{\partial O_i}{\partial Z_k}$

Hay 2 casos posibles, $\frac{S(Z_i)}{Z_i}$ o $\frac{S(Z_i)}{Z_j}$, donde $i \neq j$.

Caso $\frac{S(Z_i)}{Z_i}$

$$\frac{\partial f(x)}{\partial g(x)} = \frac{f'(x) * g(x) - g'(x) * f(x)}{g(x)^2} \quad (3.6)$$

$$S(z_i) = \frac{e^{Z_i}}{e^{Z_1} + \dots + e^{Z_H}} \quad (3.7)$$

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{[\frac{\partial e^{Z_1}}{\partial Z_1} * (e^{Z_1} + \dots + e^{Z_H})] - [\frac{\partial(e^{Z_1} + \dots + e^{Z_H})}{\partial Z_1} * e^{Z_1}]}{(e^{Z_1} + \dots + e^{Z_H})^2} \quad (3.8)$$

Se aplica $\frac{\partial e^{Z_1}}{\partial Z_1} = e^{Z_1}$

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{[e^{Z_1} * \sum_{i=1}^H e^{Z_i}] - [e^{Z_1} * e^{Z_1}]}{(\sum_{i=1}^H e^{Z_i})^2} \quad (3.9)$$

(3.10)

Se saca factor común e^{Z_1}

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{e^{Z_1}([\sum_{i=1}^H e^{Z_i}] - e^{Z_1})}{(\sum_{i=1}^H e^{Z_i})^2} \quad (3.11)$$

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{e^{Z_1}}{\sum_{i=1}^H e^{Z_i}} * \frac{[\sum_{i=1}^H e^{Z_i}] - e^{Z_1}}{\sum_{i=1}^H e^{Z_i}} \quad (3.12)$$

Se recuerda que $\frac{\sum_{i=1}^H e^{Z_i}}{\sum_{i=1}^H e^{Z_i}} = 1$ y que $S(Z_1) = \frac{e^{Z_1}}{\sum_{i=1}^H e^{Z_i}}$

$$\frac{\partial S(Z_1)}{\partial Z_1} = S(Z_1) * (1 - S(Z_1)) \quad (3.13)$$

Caso $\frac{S(Z_i)}{Z_j}$, con $i \neq j$

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{[\frac{\partial e^{Z_2}}{\partial Z_1} * (e^{Z_1} + \dots + e^{Z_H})] - [\frac{\partial(e^{Z_1} + \dots + e^{Z_H})}{\partial Z_1} * e^{Z_2}]}{(e^{Z_1} + \dots + e^{Z_H})^2} \quad (3.14)$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{[0 * [\sum_{i=1}^H e^{Z_i}]] - [e^{Z_1} * e^{Z_2}]}{(\sum_{i=1}^H e^{Z_i})^2} \quad (3.15)$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{-e^{Z_1} * e^{Z_2}}{(\sum_{i=1}^H e^{Z_i})^2} \quad (3.16)$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{-e^{Z_1}}{\sum_{i=1}^H e^{Z_i}} * \frac{e^{Z_2}}{\sum_{i=1}^H e^{Z_i}} \quad (3.17)$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = -S(Z_1) * S(Z_2) \quad (3.18)$$

Combinación de casos

De esta forma, tendremos que dividir el proceso en 2 partes, cuando i sea igual a j , y cuando $i \neq j$, perteneciendo a la primera todos los casos menos uno.

Parte izquierda cuando $i \neq k$, parte derecha cuando $i = k$.

Retomamos la fórmula 3.5, aplicando 3.18 en la parte izquierda y 3.13 en la derecha.

$$\frac{\partial E}{\partial Z_k} = -[\sum_{i=k}^H [\frac{y_i}{O_i} * -O_i * O_k] + \frac{y_k}{O_k} * O_k * (1 - O_k)] \quad (3.19)$$

Se simplifica O_i en la parte izquierda y O_k en la derecha.

$$\frac{\partial E}{\partial Z_k} = -[\sum_{i=k}^H [-y_i * O_k] + [y_k * (1 - O_k)]] \quad (3.20)$$

Se extrae O_k de la suma, pues es independiente respecto al índice i

$$\frac{\partial E}{\partial Z_k} = -[-O_k \sum_{i=k}^H -y_i + [y_k * (1 - O_k)]] \quad (3.21)$$

Simplificación One-Hot

Al emplear la codificación one-hot en Y , se sabe que la suma de sus elementos es igual a 1.

$$\sum_{i=1}^H y_i = 1 \quad (3.22)$$

$$\sum_{i=k}^H y_i = \sum_{i=1}^H y_i - y_k = 1 - y_k \quad (3.23)$$

Se emplea 3.23 para simplificar la suma en 3.21.

$$\frac{\partial E}{\partial Z_k} = [O_k * (1 - y_k)] - [y_k * (1 - O_k)] \quad (3.24)$$

$$\frac{\partial E}{\partial Z_k} = O_k - O_k * y_k - y_k + O_k * y_k \quad (3.25)$$

Se simplifica $O_k * y_k$.

$$\frac{\partial E}{\partial Z_k} = O_k - y_k = \text{gradiente}_Z k \quad (3.26)$$

3.1.2. BackPropagation con 1 capa oculta [3] [4]

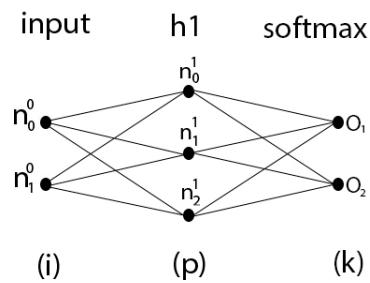


Figura 3.2: Red Neuronal totalmente conectada con 1 capa oculta

La Figura 3.2 se compone de 'puntos' interconectados mediante líneas, representando neuronas y pesos que las conectan respectivamente. Cada punto corresponde a una neurona, y cada línea a un peso.

La Figura 3.2 presenta 3 capas (input, h_1 , softmax) que corresponden a capa de entrada, capa oculta h_1 , y capa de salida respectivamente. El superíndice indica la capa a la que pertenece una neurona o peso, mientras que el

subíndice indica el número del mismo en su respectiva capa. En el caso de los pesos, se requieren 2 subíndices para identificar a cada uno (pues un peso une 2 neuronas).

La capa de entrada se compone de 2 neuronas (n_0^0 y n_1^0).

La capa oculta h_1 tiene 3 neuronas (n_0^1 , n_1^1 , y n_2^1)

El peso W_{jk}^i referencia al peso que une las neuronas n_j^i y n_k^{i+1} .

De forma adicional, se recuerda que Z_i representa la entrada i de la capa SoftMax, y O_i su salida.

Capa SoftMax

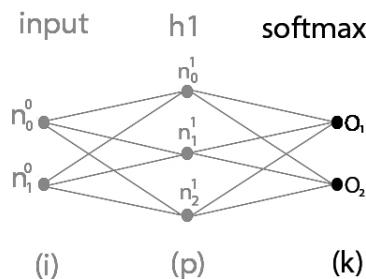


Figura 3.3: Imagen de backpropagation en la capa softmax

Sea la neurona n_j^i , se define como a_j^i el valor de dicha neurona antes de aplicar sobre ella su función de activación asociada, y z_j^i el obtenido tras aplicarla.

Tal y como se calculó previamente, el gradiente de la función de pérdida respecto a cada Z_i viene dado por la fórmula 3.26.

Pesos capas h1-SoftMax

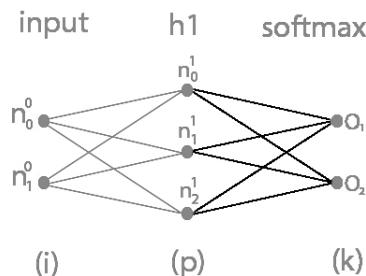


Figura 3.4: Imagen de backpropagation en los pesos entre la capa oculta y la capa SoftMax

Una vez calculado el gradiente hasta la capa softmax, se puede calcular el gradiente respecto a cada peso W_{pk}^1 que se encuentra conectado a esta

desde la capa anterior. Es decir, para cada $h_p^1 \in h_1$, se calcula $\frac{dE(x)}{dW_{pk}^1}$. Usando la regla de la cadena, equivale a realizar lo siguiente:

$$\frac{\partial Z_k}{\partial W_{pk}^1} = \frac{\partial(z_p^1 * W_{pk}^1 + b_k^2)}{\partial W_{pk}^1} = z_p^1 \quad (3.27)$$

$$\frac{\partial E(x)}{\partial W_{pk}^1} = \text{gradiente_}Z_k * \frac{\partial Z_k}{\partial W_{pk}^1} = \text{gradiente_}Z_k * z_p^1 \quad (3.28)$$

Sesgos capa softmax

$$\frac{\partial E}{\partial b_k^2} = \frac{\partial E}{\partial Z_k} * \frac{\partial Z_k}{\partial b_k^2} \quad (3.29)$$

$$\frac{\partial Z_k}{\partial b_k^2} = \frac{\partial([\sum_{c=1}^P z_c^1 * W_{pk}^1] + b_k^2)}{\partial b_k^2} = 1 \quad (3.30)$$

$$\frac{\partial E}{\partial b_k^2} = \text{gradiente_}Z_k \quad (3.31)$$

Capa oculta h1

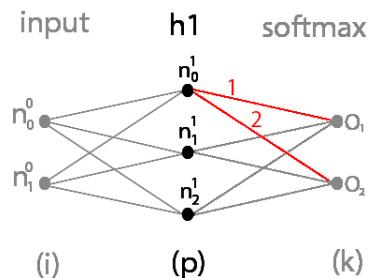


Figura 3.5: Imagen de los 'caminos' desde la capa softmax hasta n_p^1

En la figura 3.5 se muestra como hay más de un 'camino' desde la capa softmax hasta n_p^1 . Por tanto, para obtener el gradiente de la pérdida respecto a n_p^1 , habría que calcular la suma de todos los 'caminos' hacia este.

$$\frac{\partial E_{total}}{\partial a_p^1} = \sum_{k=1}^K \frac{\partial E_k}{\partial a_p^1} = \sum_{k=1}^K \text{gradiente_}Z_k * \frac{\partial Z_k}{\partial z_p^1} * \frac{\partial z_p^1}{\partial a_p^1} \quad (3.32)$$

$$\frac{\partial Z_k}{\partial z_p^1} = \frac{\partial([\sum_{c=1}^P z_c^1 * W_{ck}^1] + b_k^2)}{\partial z_p^1} = W_{pk}^1 \quad (3.33)$$

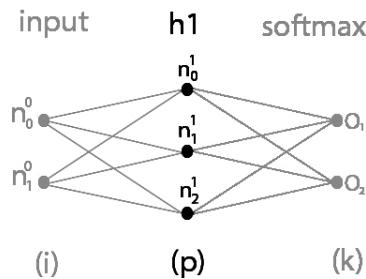


Figura 3.6: Imagen de backpropagation en la capa oculta h1

En la capa oculta h1 se emplea la función de activación sigmoide.

$$\text{sigmoide}(x) = \frac{1}{1 + e^{-x}} \quad (3.34)$$

$$\text{sigmoide}'(x) = \frac{\text{sigmoide}(x)}{1 - \text{sigmoide}(x)} \quad (3.35)$$

De esta forma,

$$\frac{\partial z_p^1}{\partial a_p^1} = \frac{\partial \text{sigmoide}(a_p^1)}{\partial a_p^1} = \text{sigmoide}(a_p^1) * (1 - \text{sigmoide}(a_p^1)) \quad (3.36)$$

Se retoma la fórmula 3.32 mediante la aplicación de 3.33 y 3.36

$$\frac{\partial E_{total}}{\partial a_p^1} = \sum_{k=1}^K \text{gradiente_}Z_k * W_{pk}^1 * \text{sigmoide}(a_p^1) * (1 - \text{sigmoide}(a_p^1)) \quad (3.37)$$

$$\frac{\partial E_{total}}{\partial a_p^1} = \text{gradiente_}h1_p \quad (3.38)$$

Pesos capas input-h1

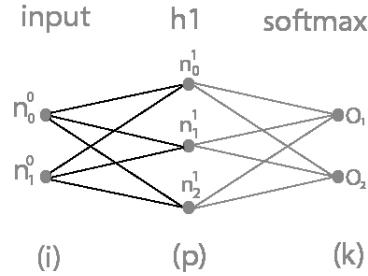


Figura 3.7: Imagen de backpropagation en los pesos entre la capa input y la capa oculta h1

$$\frac{\partial a_p^1}{\partial W_{ip}^0} = \frac{\partial [\sum_{c=1}^I z_c^0 * W_{cp}^0 + b_p^1]}{\partial W_{ip}^0} = z_i^0 \quad (3.39)$$

$$\frac{\partial E}{\partial W_{ip}^0} = \frac{\partial E_{total}}{\partial a_p^1} * \frac{\partial a_p^1}{\partial W_{ip}^0} \quad (3.40)$$

$$\frac{\partial E(x)}{\partial W_{ip}^0} = \text{gradiente_h1}_p * \frac{\partial a_p^1}{\partial W_{ip}^0} = \text{gradiente_h1}_p * z_i^0 \quad (3.41)$$

Sesgos capa h1

$$\frac{\partial E}{\partial b_p^1} = \frac{\partial E_{total}}{\partial a_p^1} * \frac{\partial a_p^1}{\partial b_p^1} \quad (3.42)$$

$$\frac{\partial a_p^1}{\partial b_p^1} = \frac{\partial ([\sum_{c=1}^I z_c^0 * W_{ip}^0] + b_p^1)}{\partial b_p^1} = 1 \quad (3.43)$$

$$\frac{\partial E}{\partial b_p^1} = \text{gradiente_h1}_p \quad (3.44)$$

Capa input

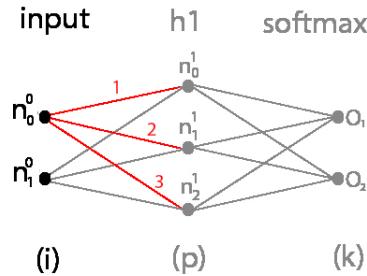


Figura 3.8: Imagen de los 'caminos' desde la capa oculta h1 hasta n_0^0

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{p=1}^P \frac{\partial E_{total}}{\partial a_p^1} * \frac{\partial a_p^1}{\partial z_i^0} * \frac{\partial z_i^0}{\partial a_i^0} \quad (3.45)$$

$$\frac{\partial a_p^1}{\partial z_i^0} = \frac{\partial([\sum_{c=1}^I z_c^0 * W_{ip}^0] + b_p^1)}{\partial z_i^0} = W_{ip}^0 \quad (3.46)$$

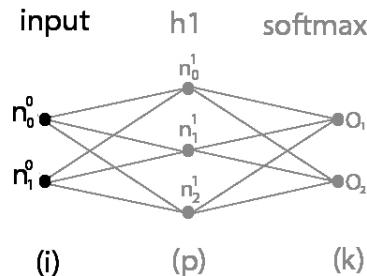


Figura 3.9: Imagen de backpropagation en la capa input

Como la capa input no presenta ninguna función de activación asociada, z_i^0 es igual a a_i^0 .

$$\frac{\partial z_i^0}{\partial a_i^0} = 1 \quad (3.47)$$

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{p=1}^P \text{gradiente_h1}_p \quad (3.48)$$

3.1.3. Retropropagación con 2 capas ocultas

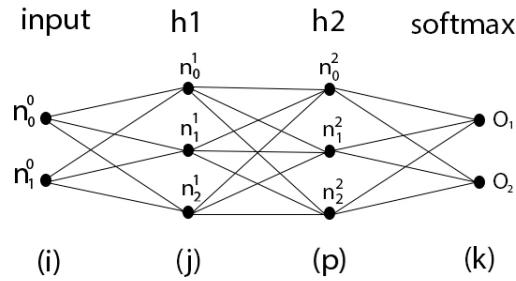


Figura 3.10: Red Neuronal totalmente conectada con 2 capas ocultas

Tal y como muestra la Figura 3.10, en este caso se emplea una red totalmente conectada con 2 capas ocultas (h1 y h2).

Capa SoftMax

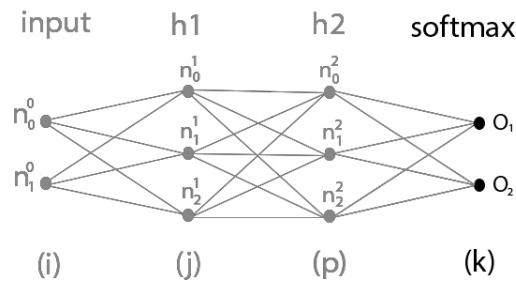


Figura 3.11: Imagen de backpropagation en la capa softmax

Al igual que en el apartado anterior (3.1.2), el gradiente de la función de pérdida respecto a cada Z_i viene dado por la fórmula 3.26.

Pesos capas h2-SoftMax

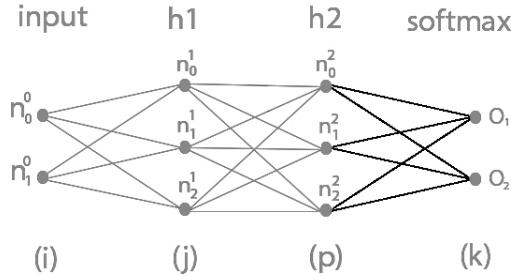


Figura 3.12: Imagen de backpropagation en los pesos entre la capa oculta h2 y la capa SoftMax

Se realiza el cálculo del gradiente de la función de pérdida respecto a cada peso W_{pk}^2 que une las neuronas de la capa oculta h2 con las de la capa de salida softmax.

$$\frac{\partial Z_k}{\partial W_{pk}^2} = \frac{\partial(z_p^2 * W_{pk}^2 + b_k^3)}{\partial W_{pk}^2} = z_p^2 \quad (3.49)$$

$$\frac{\partial E(x)}{\partial W_{pk}^2} = \text{gradiente_}Z_k * \frac{\partial Z_k}{\partial W_{pk}^2} = \text{gradiente_}Z_k * z_p^2 \quad (3.50)$$

Como es de esperar, las fórmulas 3.49 y 3.50 son casi idénticas a 3.27 y 3.28 respectivamente, salvo por el superíndice empleado ($1 \neq 2$). Esto tiene sentido pues esta parte también es común al apartado anterior.

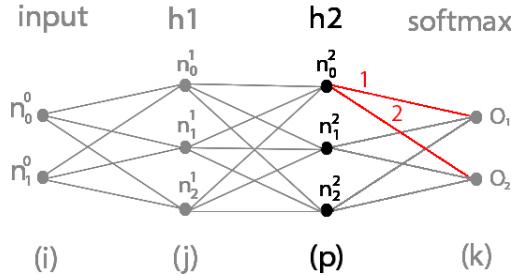
Sesgos capa softmax

$$\frac{\partial E}{\partial b_k^3} = \frac{\partial E}{\partial Z_k} * \frac{\partial Z_k}{\partial b_k^3} \quad (3.51)$$

$$\frac{\partial Z_k}{\partial b_k^3} = \frac{\partial([\sum_{c=1}^P z_c^2 * W_{pk}^2] + b_k^3)}{\partial b_k^3} = 1 \quad (3.52)$$

$$\frac{\partial E}{\partial b_k^3} = \text{gradiente_}Z_k \quad (3.53)$$

Capa oculta h2

Figura 3.13: Imagen de los 'caminos' desde la capa softmax hasta n_p^2

Tal y como se comentó anteriormente, hay más de un 'camino' desde la capa softmax hasta n_p^2 . Por tanto, para obtener el gradiente de la pérdida respecto a cada n_p^2 , habría que calcular la suma de todos los ellos.

$$\frac{\partial E_{total}}{\partial a_p^2} = \sum_{k=1}^K \frac{\partial E_k}{\partial a_p^2} = \sum_{k=1}^K \text{gradiente}_Z k * \frac{\partial Z_k}{\partial z_p^2} * \frac{\partial z_p^2}{\partial a_p^2} \quad (3.54)$$

$$\frac{\partial Z_k}{\partial z_p^2} = \frac{\partial([\sum_{c=1}^P z_c^2 * W_{ck}^2] + b_k^3)}{\partial z_p^2} = W_{pk}^2 \quad (3.55)$$

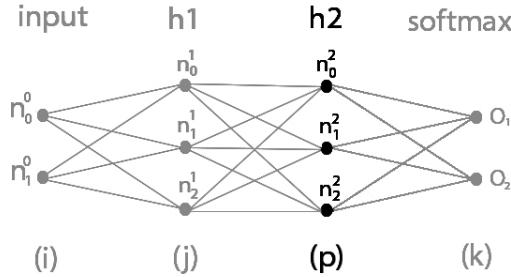


Figura 3.14: Imagen de backpropagation en la capa oculta h2

En la capa oculta h2 se emplea la función de activación sigmoide.

$$\text{sigmoide}(x) = \frac{1}{1 + e^{-x}} \quad (3.56)$$

$$\text{sigmoide}'(x) = \frac{\text{sigmoide}(x)}{1 - \text{sigmoide}(x)} \quad (3.57)$$

De esta forma,

$$\frac{\partial z_p^2}{\partial a_p^2} = \frac{\partial \text{sigmoide}(a_p^2)}{\partial a_p^2} = \text{sigmoide}(a_p^2) * (1 - \text{sigmoide}(a_p^2)) \quad (3.58)$$

Se retoma la fórmula 3.54 mediante la aplicación de 3.55 y 3.58.

$$\frac{\partial E_{total}}{\partial a_p^2} = \sum_{k=1}^K \text{gradiente_Z}_k * W_{pk}^2 * \text{sigmoide}(a_p^2) * (1 - \text{sigmoide}(a_p^2)) \quad (3.59)$$

$$\frac{\partial E_{total}}{\partial a_p^2} = \text{gradiente_h2}_p \quad (3.60)$$

Una vez más, la fórmula obtenida (3.60) coindice con la calculada previamente (3.38).

Pesos capas h1-h2

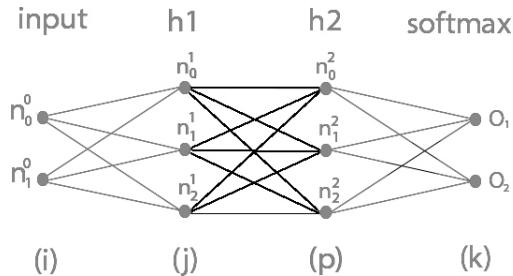


Figura 3.15: Imagen de backpropagation en los pesos entre las capas ocultas h1 y h2

$$\frac{\partial a_p^2}{\partial W_{jp}^1} = \frac{\partial [\sum_{c=1}^J z_c^1 * W_{cp}^1] + b_p^2}{\partial W_{jp}^1} = z_j^1 \quad (3.61)$$

$$\frac{\partial E}{\partial W_{jp}^1} = \frac{\partial E_{total}}{\partial a_p^2} * \frac{\partial a_p^2}{\partial W_{jp}^1} \quad (3.62)$$

$$\frac{\partial E(x)}{\partial W_{jp}^1} = \text{gradiente_h2}_p * \frac{\partial a_p^2}{\partial W_{jp}^1} = \text{gradiente_h2}_p * z_j^1 \quad (3.63)$$

La fórmula 3.63 vuelve a coincidir con 3.41

Sesgos capa h2

$$\frac{\partial E}{\partial b_p^2} = \frac{\partial E_{total}}{\partial a_p^2} * \frac{\partial a_p^2}{\partial b_p^2} \quad (3.64)$$

$$\frac{\partial a_p^2}{\partial b_p^2} = \frac{\partial([\sum_{c=1}^J z_c^1 * W_{jp}^1] + b_p^2)}{\partial b_p^2} = 1 \quad (3.65)$$

$$\frac{\partial E}{\partial b_p^2} = \text{gradiente_h2}_p \quad (3.66)$$

La fórmula 3.66 coincide con 3.44.

Capa oculta h1

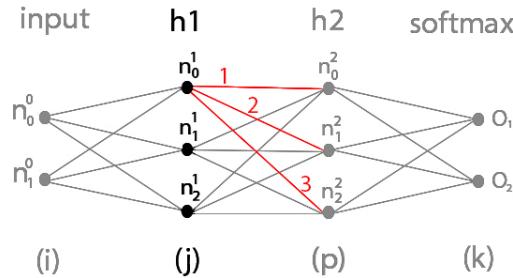


Figura 3.16: Imagen de los 'caminos' desde la capa softmax hasta n_0^1

De igual forma que se realizó en la capa h2, se calcula la suma de todos los 'caminos' hacia cada neurona n_j^1 .

$$\frac{\partial E_{total}}{\partial a_j^1} = \sum_{k=1}^K \frac{\partial E_k}{\partial a_j^1} = \sum_{p=1}^P \text{gradiente_h2}_p * \frac{\partial a_p^2}{\partial z_j^1} * \frac{\partial z_j^1}{\partial a_j^1} \quad (3.67)$$

$$\frac{\partial a_p^2}{\partial z_j^1} = \frac{\partial([\sum_{c=1}^J z_c^1 * W_{cp}^1] + b_p^2)}{\partial z_j^1} = W_{jp}^1 \quad (3.68)$$

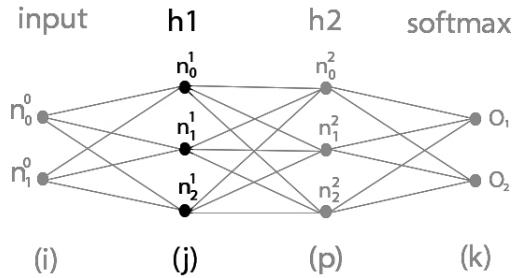


Figura 3.17: Imagen de backpropagation en la capa oculta h1

En la capa oculta h1 se emplea la función de activación ReLU.

$$\text{ReLU}(x) = \max(0, x) \quad (3.69)$$

$$\text{ReLU}'(x) = 1 \text{ si } x > 0, 0 \text{ en caso contrario} \quad (3.70)$$

De esta forma,

$$\frac{\partial z_j^1}{\partial a_j^1} = 1 \text{ si } x > 0, 0 \text{ en caso contrario} \quad (3.71)$$

$$\frac{\partial E_{total}}{\partial a_j^1} = \sum_{p=1}^P \text{gradiente_h2}_p * W_{jp}^1 * \text{ReLU}'(a_j^1) \quad (3.72)$$

$$\frac{\partial E_{total}}{\partial a_j^1} = \text{gradiente_h1}_j \quad (3.73)$$

El cálculo para obtener la fórmula 3.73 es muy parecido al realizado para 3.60, pero es “nuevo” respecto a la sección anterior, pues esta capa no existe en dicho caso.

Pesos capa input-h1

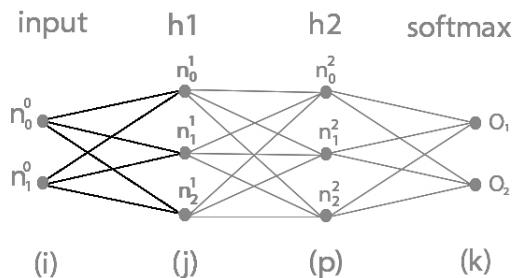


Figura 3.18: Imagen de backpropagation en los pesos entre la capa input y la capa oculta h1

$$\frac{\partial a_j^1}{\partial W_{ij}^0} = \frac{\partial([\sum_{c=1}^I z_c^0 * W_{cj}^0] + b_j^1)}{\partial W_{ij}^0} = z_i^0 \quad (3.74)$$

$$\frac{\partial E}{\partial W_{ij}^0} = \frac{\partial E_{total}}{\partial a_j^1} * \frac{\partial a_j^1}{\partial W_{ij}^0} \quad (3.75)$$

$$\frac{\partial E(x)}{\partial W_{ij}^0} = \text{gradiente_h1}_j * \frac{\partial a_j^1}{\partial W_{ij}^0} = \text{gradiente_h1}_j * z_i^0 \quad (3.76)$$

Sesgos capa h1

$$\frac{\partial E}{\partial b_j^1} = \frac{\partial E_{total}}{\partial a_j^1} * \frac{\partial a_j^1}{\partial b_j^1} \quad (3.77)$$

$$\frac{\partial a_j^1}{\partial b_j^1} = \frac{\partial([\sum_{c=1}^I z_c^0 * W_{ij}^0] + b_j^1)}{\partial b_j^1} = 1 \quad (3.78)$$

$$\frac{\partial E}{\partial b_j^1} = \text{gradiente_h1}_j \quad (3.79)$$

La fórmula 3.79 coincide con 3.44.

Capa input

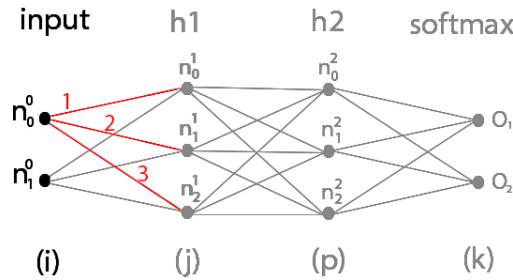


Figura 3.19: Imagen de los 'caminos' desde la capa oculta h1 hasta n_0^0

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{j=1}^J \frac{\partial E_{total}}{\partial a_j^1} * \frac{\partial a_j^1}{\partial z_i^0} * \frac{\partial z_i^0}{\partial a_i^0} \quad (3.80)$$

$$\frac{\partial a_j^1}{\partial z_i^0} = \frac{\partial([\sum_{c=1}^I z_c^0 * W_{ij}^0] + b_j^1)}{\partial z_i^0} = W_{ij}^0 \quad (3.81)$$

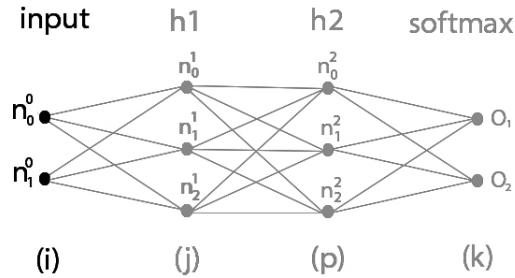


Figura 3.20: Imagen de backpropagation en la capa input

Como la capa input no presenta ninguna función de activación asociada, z_i^0 es igual a a_i^0 .

$$\frac{\partial z_i^0}{\partial a_i^0} = 1 \quad (3.82)$$

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{p=1}^P \text{gradiente_} h1_p \quad (3.83)$$

3.1.4. Conclusiones

Se definen como capas ocultas “intermedias” todas menos la última de ellas. Tal y como se ha mostrado anteriormente, comparten la mayoría del cálculo en cuanto a retropopagación. De esta forma, se puede dividir una red neuronal totalmente conectada en 4 grupos {capa input, capas ocultas intermedias, última capa oculta, capa de salida o capa softmax}.

A continuación se realiza el cálculo necesario para la retropopagación de una capa de neuronas l determinada. Suponemos que la capa l+1 tiene Q neuronas, la capa l-1 tiene K neuronas, y todas las capas ocultas intermedias usan ReLU como función de activación.

Gradiente respecto a la entrada

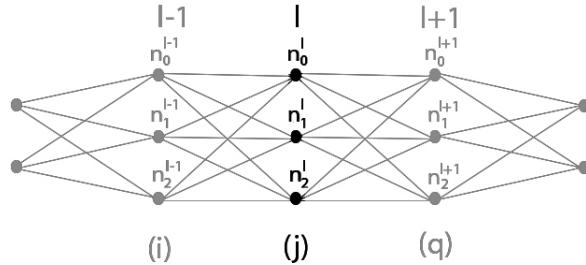


Figura 3.21: Imagen de backpropagation en la capa l

$$\frac{\partial E_{total}}{\partial a_j^l} = \sum_{q=1}^Q \frac{\partial E_{total}}{\partial a_q^{l+1}} * \frac{\partial a_q^{l+1}}{\partial z_j^l} * \frac{\partial z_j^l}{\partial a_j^l} \quad (3.84)$$

$$\frac{\partial a_j^{l+1}}{\partial z_j^l} = \frac{\partial([\sum_{c=1}^K z_c^l * W_{ij}^l] + b_j^{l+1})}{\partial z_j^l} = W_{ij}^l \quad (3.85)$$

$$\frac{\partial z_j^l}{\partial a_j^l} = \text{ReLU}'(a_j^l) \quad (3.86)$$

$$\frac{\partial E_{total}}{\partial a_j^l} = \sum_{q=1}^Q \text{gradiente_} h_{l+1_q} * W_{ij}^l * \text{ReLU}'(a_j^l) \quad (3.87)$$

$$\frac{\partial E_{total}}{\partial a_j^l} = \text{gradiente_} h_{l_j} \quad (3.88)$$

Gradiente respecto a los pesos

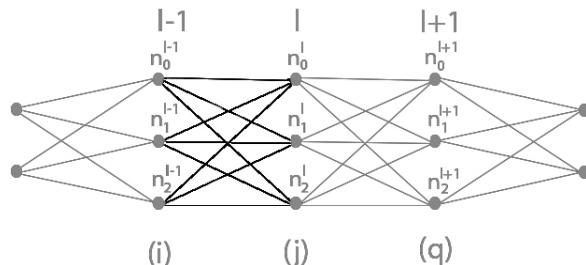


Figura 3.22: Imagen de backpropagation en los pesos entre la capa $l-1$ y l

$$\frac{\partial E}{\partial W_{ij}^{l-1}} = \frac{\partial E_{total}}{\partial a_j^l} * \frac{\partial a_j^l}{W_{ij}^{l-1}} \quad (3.89)$$

$$\frac{\partial a_j^l}{\partial W_{ij}^{l-1}} = \frac{\partial([\sum_{c=1}^K z_c^{l-1} * W_{cj}^{l-1}] + b_j^l)}{\partial W_{ij}^{l-1}} = z_i^{l-1} \quad (3.90)$$

$$\frac{\partial E(x)}{\partial W_{ij}^{l-1}} = \text{gradiente_} h_{l_j} * \frac{\partial a_j^l}{\partial W_{ij}^{l-1}} = \text{gradiente_} h_{l_j} * z_i^{l-1} \quad (3.91)$$

Gradiente respecto a sesgos

$$\frac{\partial E}{\partial b_j^l} = \frac{\partial E_{total}}{\partial a_j^l} * \frac{\partial a_j^l}{b_j^l} \quad (3.92)$$

$$\frac{\partial a_j^l}{\partial b_j^l} = \frac{\partial([\sum_{c=1}^K z_c^{l-1} * W_{ij}^{l-1}] + b_j^l)}{\partial b_j^l} = 1 \quad (3.93)$$

$$\frac{\partial E}{\partial b_j^l} = \text{gradiente_} h_{l_j} \quad (3.94)$$

3.2. Paralelización en entrenamiento

Tipos de paralelismo

El entrenamiento de una red neuronal convolucional (CNN) se puede paralelizar de distintas formas. Si el modelo se reparte entre varios ordenadores que son entrenados con los mismos datos, se denomina **paralelismo del modelo** (una capa por computador, por ejemplo). Sin embargo, si se distribuyen los datos entre múltiples nodos pero se emplea el mismo modelo para entrenar, se denomina **paralelismo de datos**.

Paralelismo en SGD

Algorithm 2 Descenso del gradiente estocástico

Datos de entrenamiento $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$.

for cada trabajador $t = 0, \dots, T - 1$ en paralelo **do**

- for** época $p \in \{0, \dots, P - 1\}$ **do**
- Trabajador 0 desordena vector de datos D.
- Sincronizar trabajadores
- for** cada mini batch $m = 0, \dots, M - 1$ **do**
- Iniciarizar gradientes^t a 0.
- $(x_i, y_i)^t \leftarrow$ Reparto de datos del batch m al trabajador t
- Realizar propagación hacia delante de x_i^t y obtener predicción \hat{y}_i^t .
- Evaluar \hat{y}_i^t con la función de pérdida y calcular el error e_i^t .
- Realizar propagación hacia detrás y obtener gradientes^t de cada parámetro del modelo.
- Acumular gradientes obtenidos por cada trabajador t.
- Actualizar parámetros.
- end for**
- end for**
- end for**

La naturaleza iterativa del algoritmo del descenso del gradiente estocástico puede parecer un obstáculo ante la paralelización del entrenamiento del modelo, pues la iteración i se basa en el resultado obtenido en la iteración i-1. Sin embargo, tal y como se indica en [41], [42], y [43], existe una forma de aplicar paralelismo en cada iteración.

En cada época se entrena al modelo con M subconjuntos de N_m datos disjuntos entre ellos de forma que, dados T “trabajadores” o procesos paralelos, se puede dividir cada mini-batch a su vez en T subconjuntos de $\frac{N_m}{T}$ datos y asignar cada uno a un trabajador distinto.

3.3. Redes Neuronales Convolucionales

3.3.1. Propagación hacia detrás

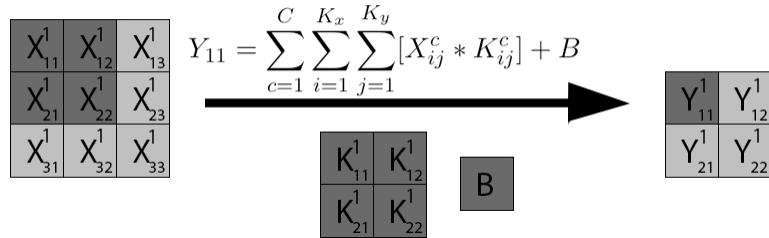


Figura 3.23: Ejemplo de propagación hacia delante en una capa convolucional

La figura 3.23 muestra el ejemplo de propagación hacia delante que se empleará en esta sección. En él, C indica el número de canales de profundidad del volumen de entrada X , mientras que K_x y K_y hacen referencia al número de filas y columnas del kernel K empleado, respectivamente.

Siguiendo la notación empleada en secciones anteriores, se denotará como A_{ij}^c al valor X_{ij}^c antes de aplicar sobre él su función de activación asociada, y Z_{ij}^c una vez esta sea aplicada.

Sumatoria de gradientes

$$\frac{\partial E}{\partial K_{11}^c} = \sum_{i=1}^I \sum_{j=1}^J \left[\frac{\partial E}{\partial Y_{ij}^c} * \frac{\partial Y_{ij}^c}{\partial K_{11}^c} \right] \quad (3.95)$$

$$\frac{\partial E}{\partial X_{11}^c} = \sum_{i=1}^I \sum_{j=1}^J \left[\frac{\partial E}{\partial Y_{ij}^c} * \frac{\partial Y_{ij}^c}{\partial Z_{11}^c} * \frac{\partial Z_{11}^c}{\partial A_{11}^c} \right] \quad (3.96)$$

Para calcular el gradiente de la función de error respecto a cada peso K_{xy} o entrada X_{xy} , se deberá realizar una sumatoria del mismo respecto a cada valor de salida de dicha convolución. En el caso de los pasos, cada uno se empleó en el cálculo de cada valor Y_{ij} . En el caso de la entrada, hay valores $x \in X$ que se emplearon en el cálculo de distintos $\{y_1, y_2\} \in Y$. Esto se vio anteriormente en la figura 2.8.

Gradiente de Y_{11}^c

Se calcula el gradiente de Y_{11}^c respecto a cada peso.

$$Y_{11}^c = Z_{11}^c * K_{11}^c + Z_{12}^c * K_{12}^c + Z_{21}^c * K_{21}^c + Z_{22}^c * K_{22}^c \quad (3.97)$$

$$\frac{\partial Y_{11}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{11}^c * K_{11}^c + Z_{12}^c * K_{12}^c + Z_{21}^c * K_{21}^c + Z_{22}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.98)$$

$$\frac{\partial Y_{11}^c}{\partial K_{11}^c} = Z_{11}^c, \quad \frac{\partial Y_{11}^c}{\partial K_{12}^c} = Z_{12}^c \quad (3.99)$$

$$\frac{\partial Y_{11}^c}{\partial K_{21}^c} = Z_{21}^c, \quad \frac{\partial Y_{11}^c}{\partial K_{22}^c} = Z_{22}^c \quad (3.100)$$

Se calcula el gradiente de Y_{11}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{11}^c}{\partial Z_{11}^c} = K_{11}^c, \quad \frac{\partial Y_{11}^c}{\partial Z_{12}^c} = K_{12}^c, \quad \frac{\partial Y_{11}^c}{\partial Z_{13}^c} = 0 \quad (3.101)$$

$$\frac{\partial Y_{11}^c}{\partial Z_{21}^c} = K_{21}^c, \quad \frac{\partial Y_{11}^c}{\partial Z_{22}^c} = K_{22}^c, \quad \frac{\partial Y_{11}^c}{\partial Z_{23}^c} = 0 \quad (3.102)$$

$$\frac{\partial Y_{11}^c}{\partial Z_{31}^c} = 0, \quad \frac{\partial Y_{11}^c}{\partial Z_{32}^c} = 0, \quad \frac{\partial Y_{11}^c}{\partial Z_{33}^c} = 0 \quad (3.103)$$

Gradiente de Y_{12}^c

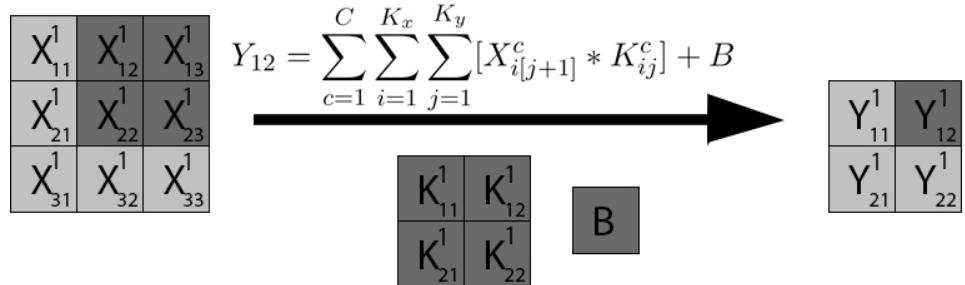


Figura 3.24: Ejemplo de propagación hacia delante en una capa convolucional

Del mismo modo, se calcula el gradiente de Y_{12}^c respecto a cada peso.

$$Y_{12}^c = Z_{12}^c * K_{11}^c + Z_{13}^c * K_{12}^c + Z_{22}^c * K_{21}^c + Z_{23}^c * K_{22}^c \quad (3.104)$$

$$\frac{\partial Y_{12}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{11}^c * K_{11}^c + Z_{12}^c * K_{12}^c + Z_{21}^c * K_{21}^c + Z_{22}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.105)$$

$$\frac{\partial Y_{12}^c}{\partial K_{11}^c} = Z_{12}^c, \quad \frac{\partial Y_{12}^c}{\partial K_{12}^c} = Z_{13}^c \quad (3.106)$$

$$\frac{\partial Y_{12}^c}{\partial K_{21}^c} = Z_{22}^c, \quad \frac{\partial Y_{12}^c}{\partial K_{22}^c} = Z_{23}^c \quad (3.107)$$

Se calcula el gradiente de Y_{12}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{12}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{12}^c}{\partial Z_{12}^c} = K_{11}^c, \quad \frac{\partial Y_{12}^c}{\partial Z_{13}^c} = K_{12}^c \quad (3.108)$$

$$\frac{\partial Y_{12}^c}{\partial Z_{21}^c} = 0, \quad \frac{\partial Y_{12}^c}{\partial Z_{22}^c} = K_{21}^c, \quad \frac{\partial Y_{12}^c}{\partial Z_{23}^c} = K_{22}^c \quad (3.109)$$

$$\frac{\partial Y_{12}^c}{\partial Z_{31}^c} = 0, \quad \frac{\partial Y_{12}^c}{\partial Z_{32}^c} = 0, \quad \frac{\partial Y_{12}^c}{\partial Z_{33}^c} = 0 \quad (3.110)$$

Gradiente de Y_{21}^c

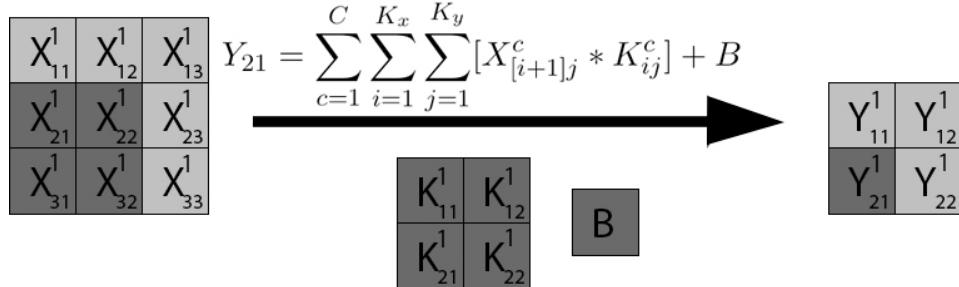


Figura 3.25: Ejemplo de propagación hacia delante en una capa convolucional

Se calcula el gradiente de Y_{21}^c respecto a cada peso.

$$Y_{21}^c = Z_{21}^c * K_{11}^c + Z_{22}^c * K_{12}^c + Z_{31}^c * K_{21}^c + Z_{32}^c * K_{22}^c \quad (3.111)$$

$$\frac{\partial Y_{21}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{21}^c * K_{11}^c + Z_{22}^c * K_{12}^c + Z_{31}^c * K_{21}^c + Z_{32}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.112)$$

$$\frac{\partial Y_{21}^c}{\partial K_{11}^c} = Z_{21}^c, \quad \frac{\partial Y_{21}^c}{\partial K_{12}^c} = Z_{22}^c \quad (3.113)$$

$$\frac{\partial Y_{21}^c}{\partial K_{21}^c} = Z_{31}^c, \quad \frac{\partial Y_{21}^c}{\partial K_{22}^c} = Z_{32}^c \quad (3.114)$$

Se calcula el gradiente de Y_{21}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{21}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{21}^c}{\partial Z_{12}^c} = 0, \quad \frac{\partial Y_{21}^c}{\partial Z_{13}^c} = 0 \quad (3.115)$$

$$\frac{\partial Y_{21}^c}{\partial Z_{21}^c} = K_{11}^c, \quad \frac{\partial Y_{21}^c}{\partial Z_{22}^c} = K_{12}^c, \quad \frac{\partial Y_{21}^c}{\partial Z_{23}^c} = 0 \quad (3.116)$$

$$\frac{\partial Y_{21}^c}{\partial Z_{31}^c} = K_{21}^c, \quad \frac{\partial Y_{21}^c}{\partial Z_{32}^c} = K_{22}^c, \quad \frac{\partial Y_{21}^c}{\partial Z_{33}^c} = 0 \quad (3.117)$$

Gradiente de Y_{22}^c

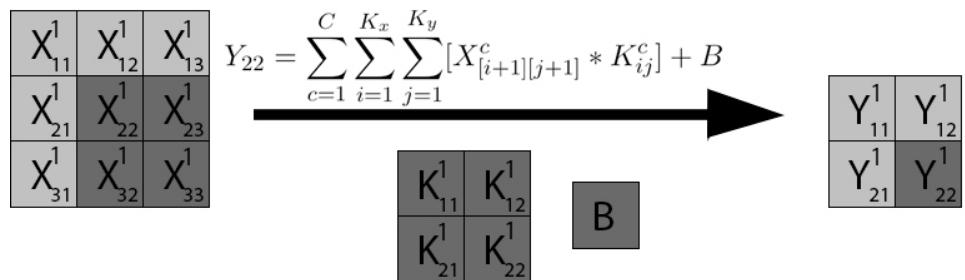


Figura 3.26: Ejemplo de propagación hacia delante en una capa convolucional

Se calcula el gradiente de Y_{22}^c respecto a cada peso.

$$Y_{22}^c = Z_{22}^c * K_{11}^c + Z_{23}^c * K_{12}^c + Z_{32}^c * K_{21}^c + Z_{33}^c * K_{22}^c \quad (3.118)$$

$$\frac{\partial Y_{22}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{11}^c * K_{11}^c + Z_{12}^c * K_{12}^c + Z_{21}^c * K_{21}^c + Z_{22}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.119)$$

$$\frac{\partial Y_{22}^c}{\partial K_{11}^c} = Z_{22}^c, \quad \frac{\partial Y_{22}^c}{\partial K_{12}^c} = Z_{23}^c \quad (3.120)$$

$$\frac{\partial Y_{22}^c}{\partial K_{21}^c} = Z_{32}^c, \quad \frac{\partial Y_{22}^c}{\partial K_{22}^c} = Z_{33}^c \quad (3.121)$$

Se calcula el gradiente de Y_{22}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{22}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{22}^c}{\partial Z_{12}^c} = 0, \quad \frac{\partial Y_{22}^c}{\partial Z_{13}^c} = 0 \quad (3.122)$$

$$\frac{\partial Y_{22}^c}{\partial Z_{21}^c} = 0, \quad \frac{\partial Y_{22}^c}{\partial Z_{22}^c} = K_{11}^c, \quad \frac{\partial Y_{22}^c}{\partial Z_{23}^c} = K_{12}^c \quad (3.123)$$

$$\frac{\partial Y_{22}^c}{\partial Z_{31}^c} = 0, \quad \frac{\partial Y_{22}^c}{\partial Z_{32}^c} = K_{21}^c, \quad \frac{\partial Y_{22}^c}{\partial Z_{33}^c} = K_{22}^c \quad (3.124)$$

Gradiente respecto a pesos como convolución

Finalmente, se calcula la sumatoria y con ello el gradiente de la función de pérdida respecto a cada peso K_{xy} .

$$\frac{\partial E}{\partial K_{11}^c} = \frac{\partial E}{\partial Y_{11}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{12}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{21}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{22}^c \quad (3.125)$$

$$\frac{\partial E}{\partial K_{12}^c} = \frac{\partial E}{\partial Y_{11}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{12}^c} * Z_{13}^c + \frac{\partial E}{\partial Y_{21}^c} * Z_{22}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{23}^c \quad (3.126)$$

$$\frac{\partial E}{\partial K_{21}^c} = \frac{\partial E}{\partial Y_{11}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{12}^c} * Z_{22}^c + \frac{\partial E}{\partial Y_{31}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{32}^c \quad (3.127)$$

$$\frac{\partial E}{\partial K_{22}^c} = \frac{\partial E}{\partial Y_{11}^c} * Z_{22}^c + \frac{\partial E}{\partial Y_{12}^c} * Z_{23}^c + \frac{\partial E}{\partial Y_{31}^c} * Z_{32}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{33}^c \quad (3.128)$$

Tal y como se observa en los cálculos obtenidos, estos coinciden con una convolución entre la entrada X y el gradiente respecto a la capa de salida Y. Esto se ve con detalle en la figura 3.27 [44].

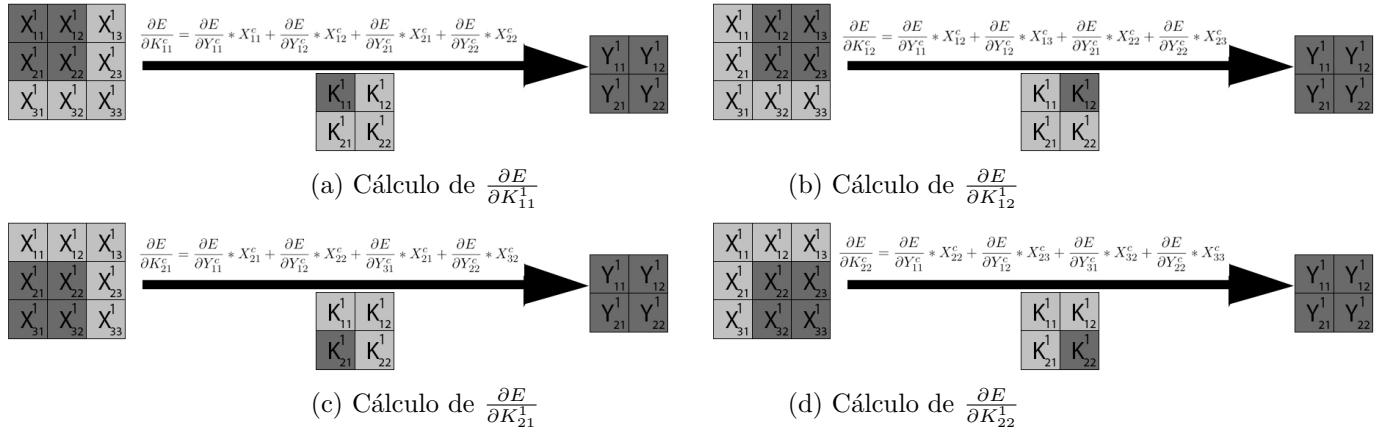


Figura 3.27: Cálculo del gradiente de la pérdida respecto a cada filtro como convolución entre X e Y

En la figura 3.27, cada subfigura $\{(a), (b), (c), (d)\}$ corresponde al cálculo del gradiente respecto a un peso distinto.

Gradiente respecto a entrada como convolución

En capas convolucionales usaremos ReLU como función de activación. Por tanto, ya se conoce su derivada pues se calculó previamente (3.70).

$$\frac{\partial E}{\partial A_{11}^c} = \frac{\partial E}{\partial Y_{11}^c} * K_{11}^c * \text{ReLU}'(A_{11}^c) \quad (3.129)$$

$$\frac{\partial E}{\partial A_{12}^c} = \left(\frac{\partial E}{\partial Y_{11}^c} * K_{12}^c + \frac{\partial E}{\partial Y_{12}^c} * K_{11}^c \right) * \text{ReLU}'(A_{12}^c) \quad (3.130)$$

$$\frac{\partial E}{\partial A_{13}^c} = \frac{\partial E}{\partial Y_{12}^c} * K_{12}^c * \text{ReLU}'(A_{13}^c) \quad (3.131)$$

$$(3.132)$$

$$\frac{\partial E}{\partial A_{21}^c} = \left(\frac{\partial E}{\partial Y_{11}^c} * K_{21}^c + \frac{\partial E}{\partial Y_{21}^c} * K_{11}^c \right) * \text{ReLU}'(A_{21}^c) \quad (3.133)$$

$$\frac{\partial E}{\partial A_{22}^c} = \left(\frac{\partial E}{\partial Y_{11}^c} * K_{22}^c + \frac{\partial E}{\partial Y_{12}^c} * K_{21}^c + \frac{\partial E}{\partial Y_{21}^c} * K_{12}^c + \frac{\partial E}{\partial Y_{22}^c} * K_{11}^c \right) * \text{ReLU}'(A_{22}^c) \quad (3.134)$$

$$\frac{\partial E}{\partial A_{23}^c} = \left(\frac{\partial E}{\partial Y_{12}^c} * K_{22}^c + \frac{\partial E}{\partial Y_{22}^c} * K_{12}^c \right) * \text{ReLU}'(A_{23}^c) \quad (3.135)$$

$$(3.136)$$

$$\frac{\partial E}{\partial A_{31}^c} = \frac{\partial E}{\partial Y_{21}^c} * K_{21}^c * \text{ReLU}'(A_{31}^c) \quad (3.137)$$

$$\frac{\partial E}{\partial A_{32}^c} = (\frac{\partial E}{\partial Y_{21}^c} * K_{22}^c + \frac{\partial E}{\partial Y_{22}^c} * K_{21}^c) * \text{ReLU}'(A_{32}^c) \quad (3.138)$$

$$\frac{\partial E}{\partial A_{33}^c} = \frac{\partial E}{\partial Y_{22}^c} * K_{22}^c * \text{ReLU}'(A_{33}^c) \quad (3.139)$$

Tal y como se observa en los cálculos obtenidos, estos coinciden con una convolución tipo completa o “full” entre el gradiente respecto a la capa de salida Y y los pesos K invertidos tanto horizontal como verticalmente. El cálculo del gradiente respecto a cada valor $x \in X$ se ve con detalle en la figura 3.29, mientras que la forma de invertir los pesos se muestra en la figura 3.28 [44].

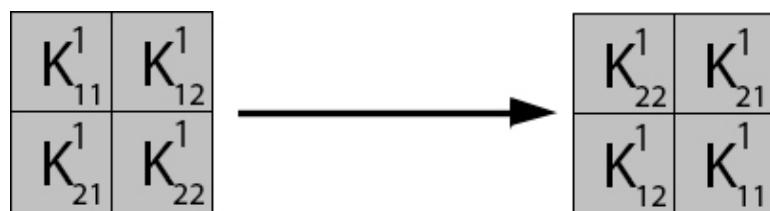


Figura 3.28: Invertir pesos en K tanto horizontal como verticalmente

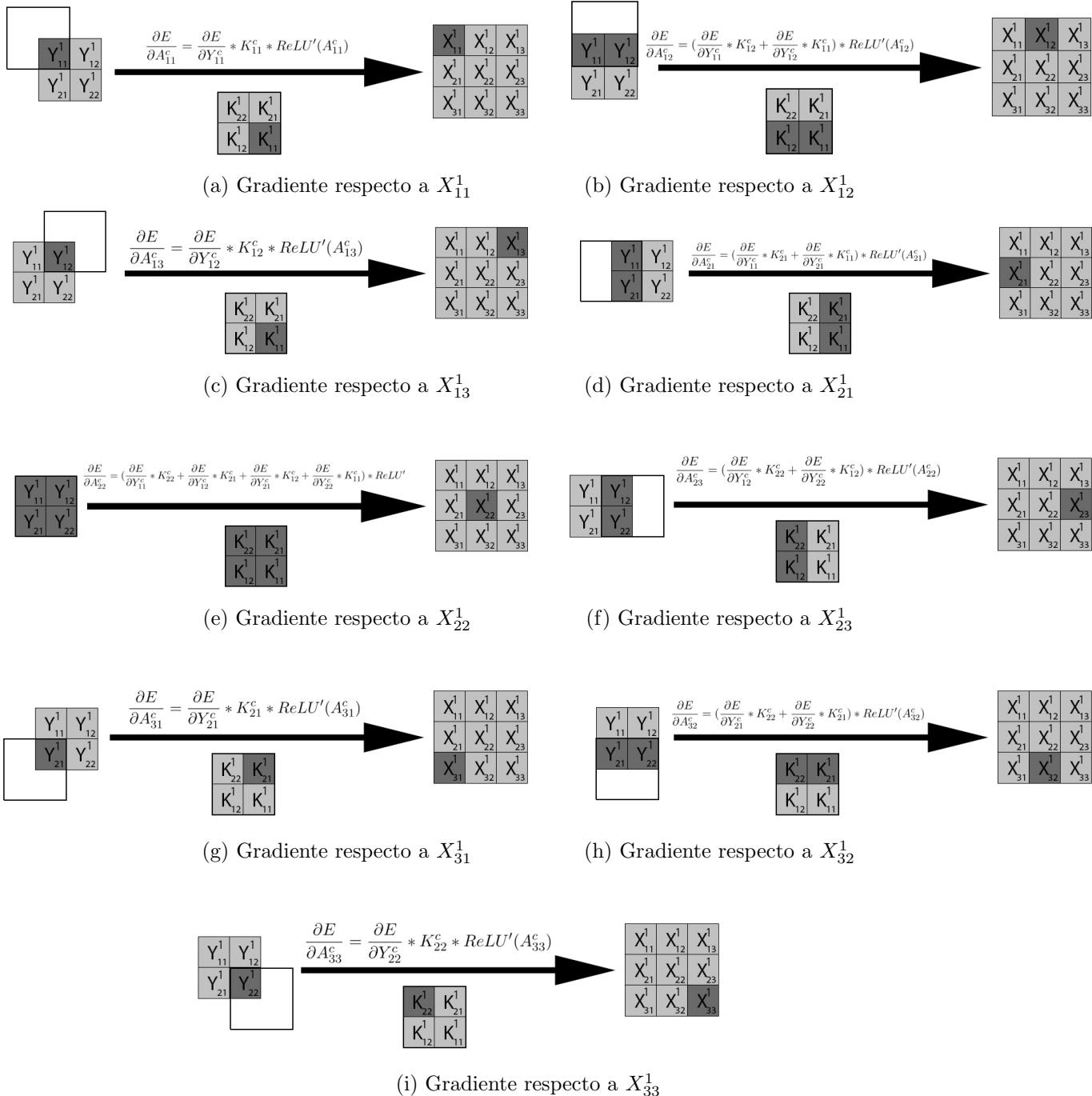


Figura 3.29: Cálculo del gradiente de la pérdida respecto a cada filtro como convolución entre X e Y

3.3.2. Propagación hacia detrás con relleno

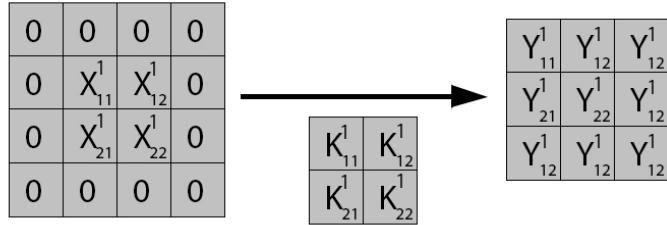


Figura 3.30: Ejemplo de propagación hacia detrás en una capa convolucional con relleno

A continuación se calcula el gradiente de la función de error respecto de los pesos y de los valores de entrada.

Gradiente de Y_{11}^c

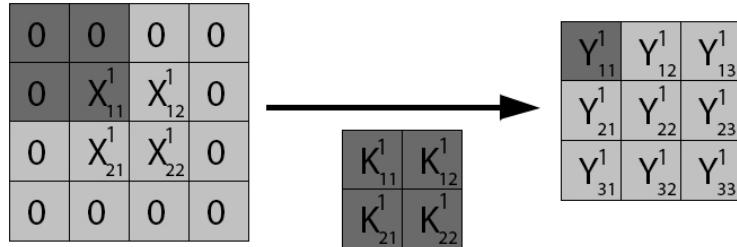


Figura 3.31: Retropropagación de Y_{11}^c

Se calcula el gradiente de Y_{11}^c respecto a cada peso.

$$Y_{11}^c = Z_{11}^c * K_{22}^c \quad (3.140)$$

$$\frac{\partial Y_{11}^c}{\partial K_{xy}^c} = \frac{\partial (Z_{11}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.141)$$

$$\frac{\partial Y_{11}^c}{\partial K_{11}^c} = 0, \quad \frac{\partial Y_{11}^c}{\partial K_{12}^c} = 0 \quad (3.142)$$

$$\frac{\partial Y_{11}^c}{\partial K_{21}^c} = 0, \quad \frac{\partial Y_{11}^c}{\partial K_{22}^c} = Z_{11}^c \quad (3.143)$$

Se calcula el gradiente de Y_{11}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{11}^c}{\partial Z_{11}^c} = K_{22}^c, \quad \frac{\partial Y_{11}^c}{\partial Z_{12}^c} = 0 \quad (3.144)$$

$$\frac{\partial Y_{11}^c}{\partial Z_{21}^c} = 0, \quad \frac{\partial Y_{11}^c}{\partial Z_{22}^c} = 0 \quad (3.145)$$

Gradiente de Y_{12}^c

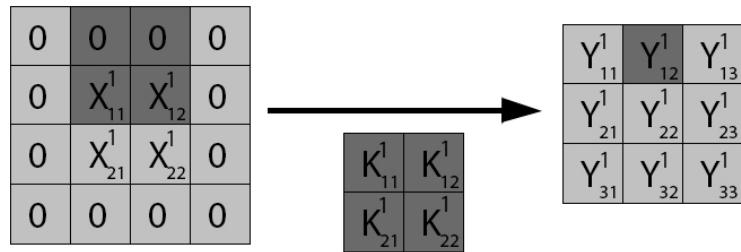


Figura 3.32: Retropropagación de Y_{12}^c

Se calcula el gradiente de Y_{12}^c respecto a cada peso.

$$Y_{12}^c = Z_{11}^c * K_{21}^c + Z_{12}^c * K_{22}^c \quad (3.146)$$

$$\frac{\partial Y_{12}^c}{\partial K_{xy}^c} = \frac{\partial (Z_{11}^c * K_{21}^c + Z_{12}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.147)$$

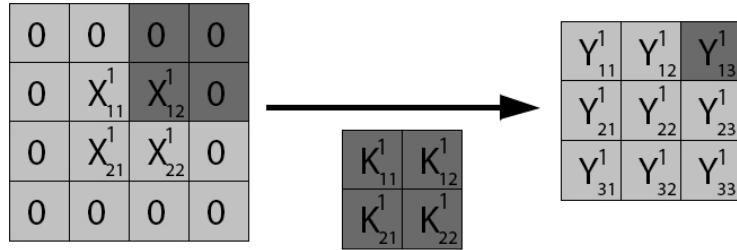
$$\frac{\partial Y_{12}^c}{\partial K_{11}^c} = 0, \quad \frac{\partial Y_{12}^c}{\partial K_{12}^c} = 0 \quad (3.148)$$

$$\frac{\partial Y_{12}^c}{\partial K_{21}^c} = Z_{11}^c, \quad \frac{\partial Y_{12}^c}{\partial K_{22}^c} = Z_{12}^c \quad (3.149)$$

Se calcula el gradiente de Y_{12}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{12}^c}{\partial Z_{11}^c} = K_{21}^c, \quad \frac{\partial Y_{12}^c}{\partial Z_{12}^c} = K_{22}^c \quad (3.150)$$

$$\frac{\partial Y_{12}^c}{\partial Z_{21}^c} = 0, \quad \frac{\partial Y_{12}^c}{\partial Z_{22}^c} = 0 \quad (3.151)$$

Gradiente de Y_{13}^c Figura 3.33: Retropropagación de Y_{13}^c

Se calcula el gradiente de Y_{13}^c respecto a cada peso.

$$Y_{13}^c = Z_{12}^c * K_{21}^c \quad (3.152)$$

$$\frac{\partial Y_{13}^c}{\partial K_{xy}^c} = \frac{\partial (Z_{12}^c * K_{21}^c)}{\partial K_{xy}^c} \quad (3.153)$$

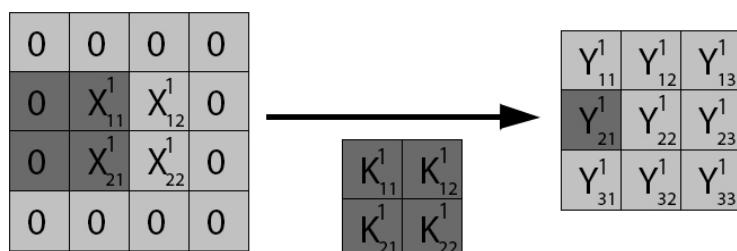
$$\frac{\partial Y_{13}^c}{\partial K_{11}^c} = 0, \quad \frac{\partial Y_{13}^c}{\partial K_{12}^c} = 0 \quad (3.154)$$

$$\frac{\partial Y_{13}^c}{\partial K_{21}^c} = Z_{12}^c, \quad \frac{\partial Y_{13}^c}{\partial K_{22}^c} = 0 \quad (3.155)$$

Se calcula el gradiente de Y_{13}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{13}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{13}^c}{\partial Z_{12}^c} = K_{21}^c \quad (3.156)$$

$$\frac{\partial Y_{13}^c}{\partial Z_{21}^c} = 0, \quad \frac{\partial Y_{13}^c}{\partial Z_{22}^c} = 0 \quad (3.157)$$

Gradiente de Y_{21}^c Figura 3.34: Retropropagación de Y_{21}^c

Se calcula el gradiente de Y_{21}^c respecto a cada peso.

$$Y_{21}^c = Z_{11}^c * K_{12}^c + Z_{21}^c * K_{22}^c \quad (3.158)$$

$$\frac{\partial Y_{21}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{11}^c * K_{12}^c + Z_{21}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.159)$$

$$\frac{\partial Y_{21}^c}{\partial K_{11}^c} = 0, \quad \frac{\partial Y_{21}^c}{\partial K_{12}^c} = Z_{11}^c \quad (3.160)$$

$$\frac{\partial Y_{21}^c}{\partial K_{21}^c} = 0, \quad \frac{\partial Y_{21}^c}{\partial K_{22}^c} = Z_{21}^c \quad (3.161)$$

Se calcula el gradiente de Y_{21}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{21}^c}{\partial Z_{11}^c} = K_{12}^c, \quad \frac{\partial Y_{21}^c}{\partial Z_{12}^c} = 0 \quad (3.162)$$

$$\frac{\partial Y_{21}^c}{\partial Z_{21}^c} = K_{22}^c, \quad \frac{\partial Y_{21}^c}{\partial Z_{22}^c} = 0 \quad (3.163)$$

Gradiente de Y_{22}^c

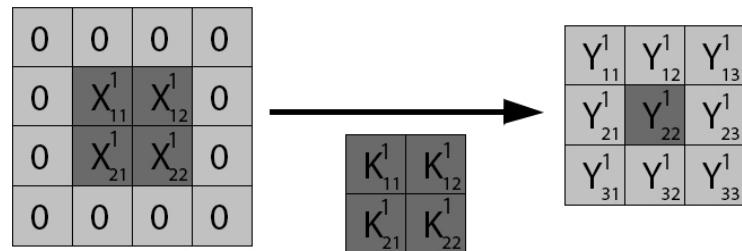


Figura 3.35: Retropropagación de Y_{22}^c

Se calcula el gradiente de Y_{22}^c respecto a cada peso.

$$Y_{22}^c = Z_{11}^c * K_{11}^c + Z_{12}^c * K_{12}^c + Z_{21}^c * K_{21}^c + Z_{22}^c * K_{22}^c \quad (3.164)$$

$$\frac{\partial Y_{22}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{11}^c * K_{11}^c + Z_{12}^c * K_{12}^c + Z_{21}^c * K_{21}^c + Z_{22}^c * K_{22}^c)}{\partial K_{xy}^c} \quad (3.165)$$

$$\frac{\partial Y_{22}^c}{\partial K_{11}^c} = Z_{11}^c, \quad \frac{\partial Y_{22}^c}{\partial K_{12}^c} = Z_{12}^c \quad (3.166)$$

$$\frac{\partial Y_{22}^c}{\partial K_{21}^c} = Z_{21}^c, \quad \frac{\partial Y_{22}^c}{\partial K_{22}^c} = Z_{22}^c \quad (3.167)$$

Se calcula el gradiente de Y_{22}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{22}^c}{\partial Z_{11}^c} = K_{11}^c, \quad \frac{\partial Y_{22}^c}{\partial Z_{12}^c} = K_{12}^c \quad (3.168)$$

$$\frac{\partial Y_{22}^c}{\partial Z_{21}^c} = K_{21}^c, \quad \frac{\partial Y_{22}^c}{\partial Z_{22}^c} = K_{22}^c \quad (3.169)$$

Gradiente de Y_{23}^c

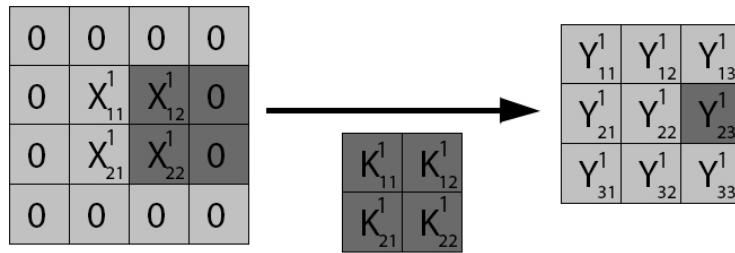


Figura 3.36: Retropropagación de Y_{23}^c

Se calcula el gradiente de Y_{23}^c respecto a cada peso.

$$Y_{23}^c = Z_{12}^c * K_{11}^c + Z_{22}^c * K_{21}^c \quad (3.170)$$

$$\frac{\partial Y_{23}^c}{\partial K_{xy}^c} = \frac{\partial (Z_{12}^c * K_{11}^c + Z_{22}^c * K_{21}^c)}{\partial K_{xy}^c} \quad (3.171)$$

$$\frac{\partial Y_{23}^c}{\partial K_{11}^c} = Z_{12}^c, \quad \frac{\partial Y_{23}^c}{\partial K_{12}^c} = 0 \quad (3.172)$$

$$\frac{\partial Y_{23}^c}{\partial K_{21}^c} = Z_{22}^c, \quad \frac{\partial Y_{23}^c}{\partial K_{22}^c} = 0 \quad (3.173)$$

Se calcula el gradiente de Y_{23}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{23}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{23}^c}{\partial Z_{12}^c} = K_{11}^c \quad (3.174)$$

$$\frac{\partial Y_{23}^c}{\partial Z_{21}^c} = 0, \quad \frac{\partial Y_{23}^c}{\partial Z_{22}^c} = K_{21}^c \quad (3.175)$$

Gradiente de Y_{31}^c

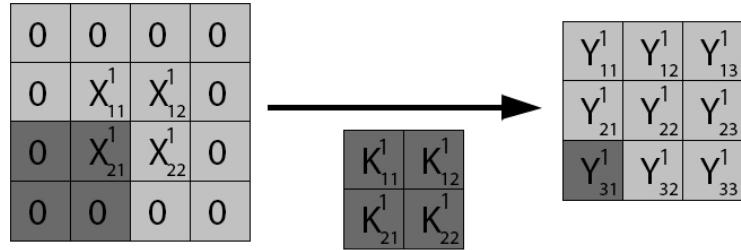


Figura 3.37: Retropropagación de Y_{31}^c

Se calcula el gradiente de Y_{31}^c respecto a cada peso.

$$Y_{31}^c = Z_{21}^c * K_{12}^c \quad (3.176)$$

$$\frac{\partial Y_{31}^c}{\partial K_{xy}^c} = \frac{\partial (Z_{21}^c * K_{12}^c)}{\partial K_{xy}^c} \quad (3.177)$$

$$\frac{\partial Y_{31}^c}{\partial K_{11}^c} = 0, \quad \frac{\partial Y_{31}^c}{\partial K_{12}^c} = Z_{21}^c \quad (3.178)$$

$$\frac{\partial Y_{31}^c}{\partial K_{21}^c} = 0, \quad \frac{\partial Y_{31}^c}{\partial K_{22}^c} = 0 \quad (3.179)$$

Se calcula el gradiente de Y_{31}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{31}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{31}^c}{\partial Z_{12}^c} = 0 \quad (3.180)$$

$$\frac{\partial Y_{31}^c}{\partial Z_{21}^c} = K_{12}^c, \quad \frac{\partial Y_{31}^c}{\partial Z_{22}^c} = 0 \quad (3.181)$$

Gradiente de Y_{32}^c

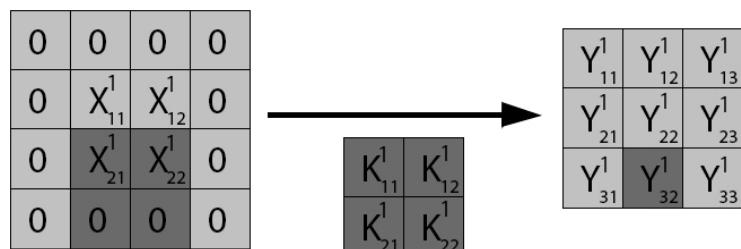


Figura 3.38: Retropropagación de Y_{32}^c

Se calcula el gradiente de Y_{32}^c respecto a cada peso.

$$Y_{32}^c = Z_{21}^c * K_{11}^c + Z_{22}^c * K_{12}^c \quad (3.182)$$

$$\frac{\partial Y_{32}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{21}^c * K_{11}^c + Z_{22}^c * K_{12}^c)}{\partial K_{xy}^c} \quad (3.183)$$

$$\frac{\partial Y_{32}^c}{\partial K_{11}^c} = Z_{21}^c, \quad \frac{\partial Y_{32}^c}{\partial K_{12}^c} = Z_{22}^c \quad (3.184)$$

$$\frac{\partial Y_{32}^c}{\partial K_{21}^c} = 0, \quad \frac{\partial Y_{32}^c}{\partial K_{22}^c} = 0 \quad (3.185)$$

Se calcula el gradiente de Y_{32}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{32}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{32}^c}{\partial Z_{12}^c} = 0 \quad (3.186)$$

$$\frac{\partial Y_{32}^c}{\partial Z_{21}^c} = K_{11}^c, \quad \frac{\partial Y_{32}^c}{\partial Z_{22}^c} = K_{12}^c \quad (3.187)$$

Gradiente de Y_{33}^c

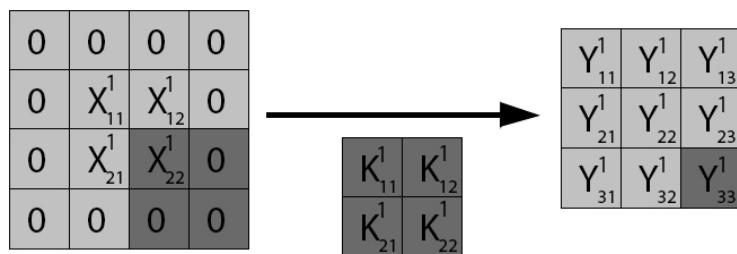


Figura 3.39: Retropropagación de Y_{33}^c

Se calcula el gradiente de Y_{33}^c respecto a cada peso.

$$Y_{33}^c = Z_{22}^c * K_{11}^c \quad (3.188)$$

$$\frac{\partial Y_{33}^c}{\partial K_{xy}^c} = \frac{\partial(Z_{22}^c * K_{11}^c)}{\partial K_{xy}^c} \quad (3.189)$$

$$\frac{\partial Y_{33}^c}{\partial K_{11}^c} = Z_{22}^c, \quad \frac{\partial Y_{33}^c}{\partial K_{12}^c} = 0 \quad (3.190)$$

$$\frac{\partial Y_{33}^c}{\partial K_{21}^c} = 0, \quad \frac{\partial Y_{33}^c}{\partial K_{22}^c} = 0 \quad (3.191)$$

Se calcula el gradiente de Y_{33}^c respecto a cada valor del volumen de entrada.

$$\frac{\partial Y_{33}^c}{\partial Z_{11}^c} = 0, \quad \frac{\partial Y_{33}^c}{\partial Z_{12}^c} = 0 \quad (3.192)$$

$$\frac{\partial Y_{33}^c}{\partial Z_{21}^c} = 0, \quad \frac{\partial Y_{33}^c}{\partial Z_{22}^c} = K_{11}^c \quad (3.193)$$

Gradiente respecto a pesos como convolución

Finalmente, se calcula la sumatoria y con ello el gradiente de la función de pérdida respecto a cada peso K_{xy} .

$$\frac{\partial E}{\partial K_{11}^c} = \frac{\partial E}{\partial Y_{22}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{23}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{32}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{33}^c} * Z_{22}^c \quad (3.194)$$

$$\frac{\partial E}{\partial K_{12}^c} = \frac{\partial E}{\partial Y_{21}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{31}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{32}^c} * Z_{22}^c \quad (3.195)$$

$$\frac{\partial E}{\partial K_{21}^c} = \frac{\partial E}{\partial Y_{12}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{13}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{23}^c} * Z_{22}^c \quad (3.196)$$

$$\frac{\partial E}{\partial K_{22}^c} = \frac{\partial E}{\partial Y_{11}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{12}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{21}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{22}^c \quad (3.197)$$

Tal y como se observa en los cálculos obtenidos, estos coinciden con una convolución entre la entrada X con relleno y el gradiente respecto a la capa de salida Y (figura 3.40).

0	0	0	0
0	X_{11}^1	X_{12}^1	0
0	X_{21}^1	X_{22}^1	0
0	0	0	0

$$\frac{\partial E}{\partial K_{11}^c} = \frac{\partial E}{\partial Y_{22}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{23}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{32}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{33}^c} * Z_{22}^c$$

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix}$$

(a) Cálculo de $\frac{\partial E}{\partial K_{11}^c}$

0	0	0	0
0	X_{11}^1	X_{12}^1	0
0	X_{21}^1	X_{22}^1	0
0	0	0	0

$$\frac{\partial E}{\partial K_{12}^c} = \frac{\partial E}{\partial Y_{21}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{31}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{32}^c} * Z_{22}^c$$

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix}$$

(b) Cálculo de $\frac{\partial E}{\partial K_{12}^c}$

0	0	0	0
0	X_{11}^1	X_{12}^1	0
0	X_{21}^1	X_{22}^1	0
0	0	0	0

$$\frac{\partial E}{\partial K_{21}^c} = \frac{\partial E}{\partial Y_{12}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{13}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{23}^c} * Z_{22}^c$$

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix}$$

(c) Cálculo de $\frac{\partial E}{\partial K_{21}^c}$

0	0	0	0
0	X_{11}^1	X_{12}^1	0
0	X_{21}^1	X_{22}^1	0
0	0	0	0

$$\frac{\partial E}{\partial K_{22}^c} = \frac{\partial E}{\partial Y_{11}^c} * Z_{11}^c + \frac{\partial E}{\partial Y_{12}^c} * Z_{12}^c + \frac{\partial E}{\partial Y_{21}^c} * Z_{21}^c + \frac{\partial E}{\partial Y_{22}^c} * Z_{22}^c$$

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix}$$

(d) Cálculo de $\frac{\partial E}{\partial K_{22}^c}$

Figura 3.40: Cálculo del gradiente de la pérdida respecto a cada filtro como convolución entre X e Y

Gradiente respecto a entrada como convolución

En capas convolucionales usaremos ReLU como función de activación. Por tanto, ya se conoce su derivada pues se calculó previamente (3.70).

$$\frac{\partial E}{\partial A_{11}^c} = (\frac{\partial E}{\partial Y_{11}^c} * K_{22}^c + \frac{\partial E}{\partial Y_{12}^c} * K_{21}^c + \frac{\partial E}{\partial Y_{21}^c} * K_{12}^c + \frac{\partial E}{\partial Y_{22}^c} * K_{11}^c) * \text{ReLU}'(A_{11}^c) \quad (3.198)$$

$$\frac{\partial E}{\partial A_{12}^c} = (\frac{\partial E}{\partial Y_{12}^c} * K_{22}^c + \frac{\partial E}{\partial Y_{13}^c} * K_{21}^c + \frac{\partial E}{\partial Y_{22}^c} * K_{12}^c + \frac{\partial E}{\partial Y_{23}^c} * K_{11}^c) * \text{ReLU}'(A_{21}^c) \quad (3.199)$$

$$\frac{\partial E}{\partial A_{21}^c} = (\frac{\partial E}{\partial Y_{21}^c} * K_{22}^c + \frac{\partial E}{\partial Y_{22}^c} * K_{21}^c + \frac{\partial E}{\partial Y_{31}^c} * K_{12}^c + \frac{\partial E}{\partial Y_{32}^c} * K_{11}^c) * \text{ReLU}'(A_{21}^c) \quad (3.200)$$

$$\frac{\partial E}{\partial A_{22}^c} = (\frac{\partial E}{\partial Y_{22}^c} * K_{22}^c + \frac{\partial E}{\partial Y_{23}^c} * K_{21}^c + \frac{\partial E}{\partial Y_{32}^c} * K_{12}^c + \frac{\partial E}{\partial Y_{33}^c} * K_{11}^c) * \text{ReLU}'(A_{22}^c) \quad (3.201)$$

Tal y como se observa en los cálculos obtenidos, estos coinciden con una convolución entre el gradiente respecto a la capa de salida Y y los pesos K invertidos tanto horizontal como verticalmente. Esto se ilustra en detalle en la figura 3.41.

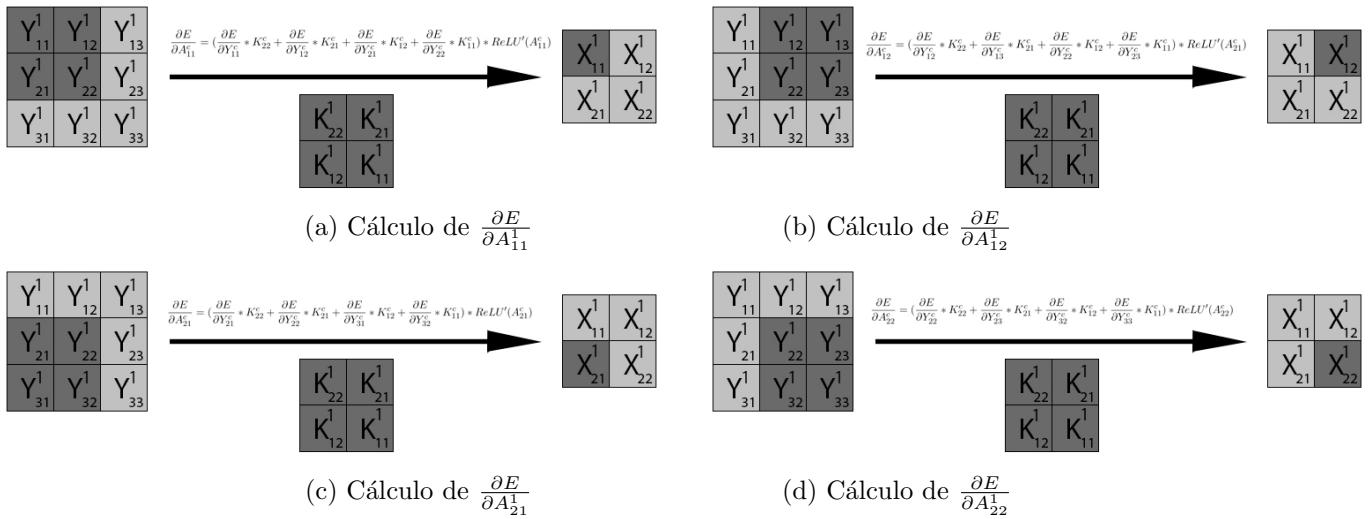
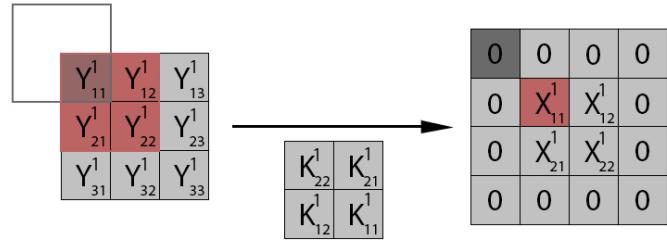


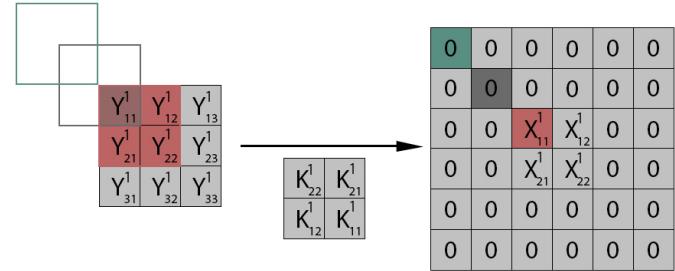
Figura 3.41: Cálculo del gradiente de la pérdida respecto de la entrada como convolución

Una vez desarrolladas ambas retropropagaciones en una capa convolucional (con y sin relleno), se pueden comparar las figuras 3.29 y 3.41 y ver

que, aunque ambas calculan el gradiente de la pérdida respecto al volumen de entrada, en una figura se emplea una convolución completa mientras que en la otra no.



(a) Retropropagación con un nivel de relleno



(b) Retropropagación con dos niveles de relleno

Figura 3.42: Cálculo del gradiente de la pérdida respecto a la entrada X con uno o dos niveles de relleno

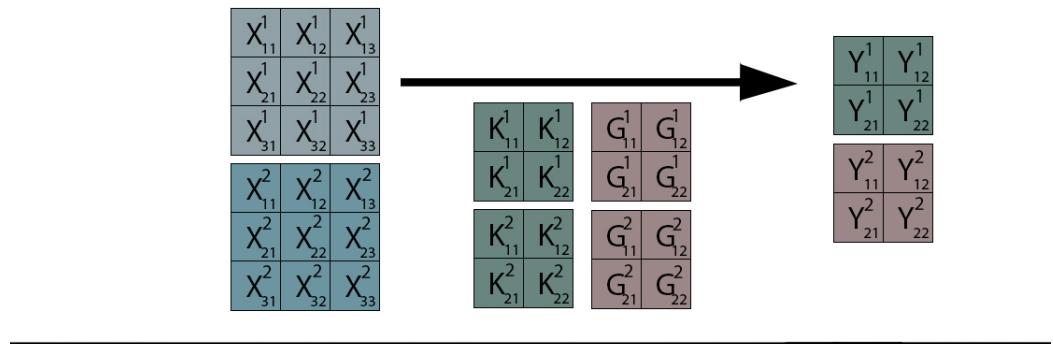
El motivo de ello se presenta en detalle en la figura 3.42. Al realizar una convolución total “desde el principio”, se empiezan a calcular los gradientes comenzando por la esquina superior izquierda de X . Sin embargo, sabiendo que X tiene relleno, no es necesario realizar el cálculo de dichos gradientes pues no se emplean en ningún cálculo posterior.

Capítulo 4

Adaptación GPU

4.1. Convolución como GEMM

ESTÁNDAR



GEMM

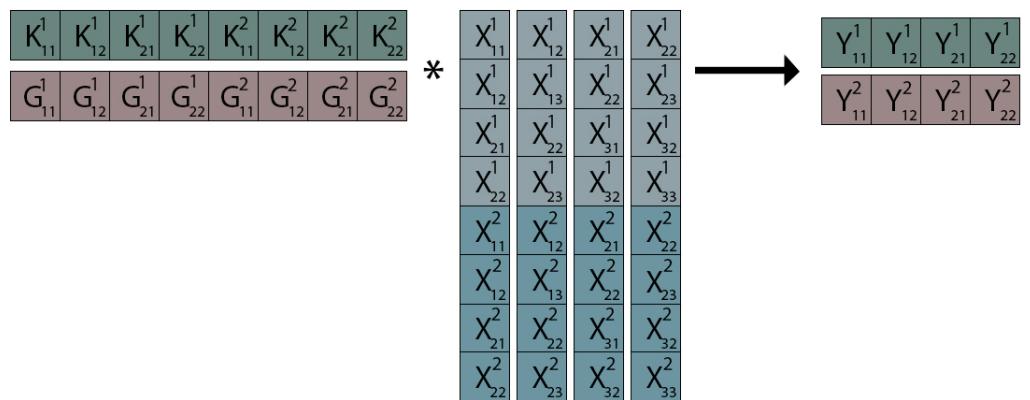


Figura 4.1: Imagen de una convolución estándar frente a una convolución empleando GEMM

Se trata de un enfoque ampliamente conocido en el mundo de deep learning, empleándolo gran cantidad de librerías del sector como Caffe, Torch-cunn, Theano-CorrMM, o incluso CuDNN [45].

El enfoque GEMM (General Matrix Multiply o Multiplicación General de Matrices) en redes neuronales convolucionales permite reducir el tiempo de cómputo requerido en una convolución a cambio de aumentar el espacio necesario para la misma. Este método consiste en “desenrollar” tanto el volumen de entrada X como la matriz de filtros K, además de realizar una serie de duplicaciones en X, de tal forma que cada columna de esta nueva matriz 2D X_unroll contenga todos los elementos de X implicados en el cálculo de una posición distinta del volumen de salida Y. Como los resultados de cada convolución se suman a lo largo de cada canal de profundidad, estos se pueden concatenar en una matriz de gran tamaño. De esta forma, cada kernel de pesos se transforma en una fila de una gran matriz de pesos, tal y como se muestra en la figura 4.1 con los kernels K y G (colores verde y marrón).

Mientras que el método estándar requiere de varias iteraciones para calcular cada valor del volumen de salida Y (figura 2.8), en la parte inferior de la figura 4.1 se observa como la alternativa GEMM permite que cada valor de Y sea el resultado de multiplicar una fila de pesos (K o G en la figura 4.1) por una columna de X_unroll (azul) [7].

De esta forma, con N kernels de tamaño KxK, un volumen de entrada con C canales de profundidad, y un volumen de salida de dimensiones $N \times H_{out} \times W_{out}$, una multiplicación matricial entre la matriz de pesos M_1 con N filas y $K^2 * C$ columnas, y X_unroll con $K^2 * C$ filas y $H_{out} * W_{out}$ columnas, produce el mismo volumen de salida Y que una convolución ordinaria con N kernels distintos.

Por último, aunque se haya omitido para simplificar la comprensión del método planteado, tras realizar dicha multiplicación matricial se debe sumar el sesgo y aplicar la función de activación correspondiente a cada elemento del volumen de salida Y.

4.1.1. Memoria requerida al emplear GEMM

Al realizar una convolución, un mismo filtro de pesos puede iterar más de una vez por varios valores de X. Esto implica la duplicación dichos valores de X en la matriz X_unroll tantas veces como se requiera el acceso a los mismos. En el caso de una entrada X(3x3) y un kernel de pesos K(2x2) (ejemplo de la figura 4.1), el valor central de X deberá duplicarse 4 veces, mientras que el valor central de cada lateral se duplicará 2 veces y los valores de cada esquina solo requieren 1 acceso por lo que no se duplican. Así, una matriz inicial X(3x3) con 9 valores, se transforma en X_unroll con $4*1 + 2*4 + 1*4 = 16$ valores, contando con un ratio de expansión de $16/9 = 1.8$.

Como cada valor de salida Y_{ij}^m es producto de una convolución de $K*K$ pe-

sos a lo largo de X sobre C canales de profundidad, el número de columnas de X_unroll se define por $C*K*K$. De la misma forma, como el resultado de cada fila multiplicada por cada columna de las matrices “desenrolladas” aportan un valor Y_{ij}^m , X_unrolled tiene tantas columnas como Y elementos ($M * H_{out} * W_{out}$).

El ratio de expansión se define mediante $\frac{C*K*K*H_{out}*W_{out}}{C*H*W}$, donde H y W hacen referencia a las filas y columnas de X, y H_{out} y W_{out} a las filas y columnas de la salida Y respectivamente. En general, si la entrada X y la salida Y poseen unas dimensiones mucho más grandes que el filtro de pesos K, el ratio de expansión será de $K*K$.

Dado que cada kernel de pesos presenta unas dimensiones de $K*K$ y se representa como una fila en la matriz total de pesos, esta tendrá $K*K$ filas y M columnas, siendo M el número de filtros de pesos a aplicar sobre X.

Realizar varias convoluciones sobre una misma entrada X (una con cada filtro de pesos distinto) implica compartir una única matriz X_unrolled. Sin embargo, para valores comunes de filtros de pesos (5 o superior), una matriz de entrada con dimensiones expandidas con un ratio de $K*K$ puede ser excesivamente grande. Como el consumo de memoria por almacenar todas las matrices de entrada expandidas en un minibatch puede suponer un problema, simplemente se reservará un espacio de $C * K * K * H_{out} * W_{out}$ (una instancia de X_unrolled), de forma que se reutilice para cada dato del minibatch [7].

4.2. Retropropagación GEMM en capa convolucional

En la figura 3.41 se vio como la retropropagación respecto a la entrada en una capa convolucional consistía en una convolución entre los kernels de pesos y el volumen de salida. Del mismo modo, en la figura 3.40 también se mostró como el gradiente respecto a los pesos consistía en una convolución entre el volumen de entrada y el de salida.

Como ambos gradientes se pueden implementar como convoluciones, y en la sección 4.1 se mostró como implementar convoluciones con el enfoque GEMM, la retropropagación en una capa convolucional se puede implementar mediante el enfoque GEMM.

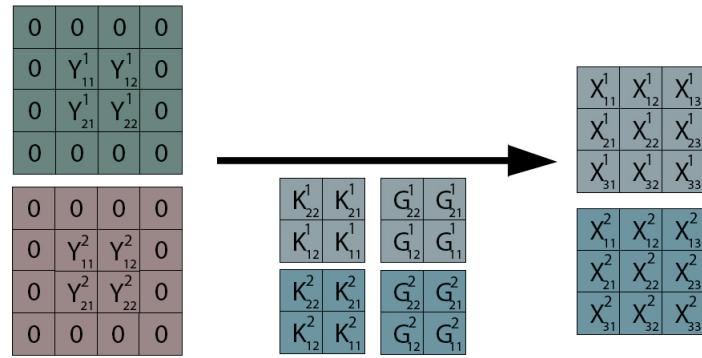
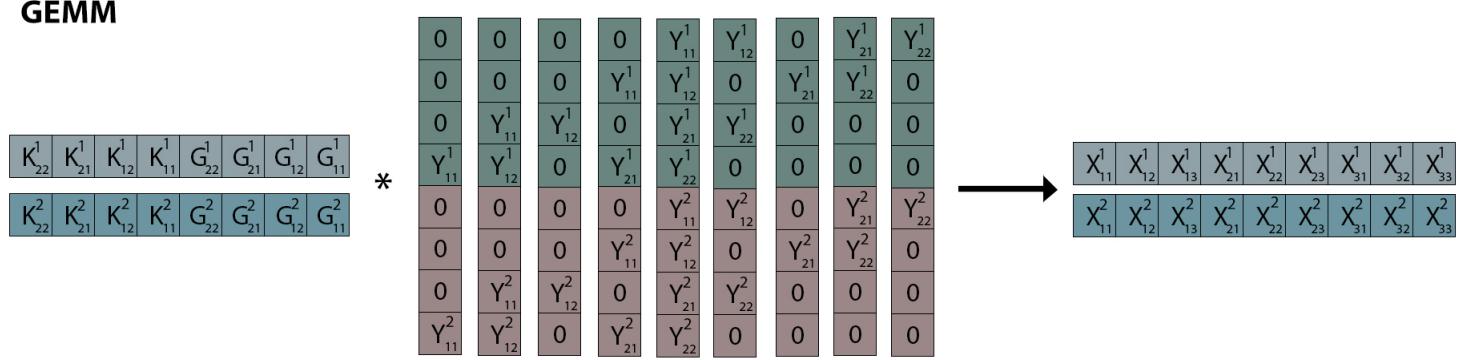
ESTÁNDAR**GEMM**

Figura 4.2: Retropropagación en una capa convolucional de forma estándar frente a GEMM respecto a la entrada

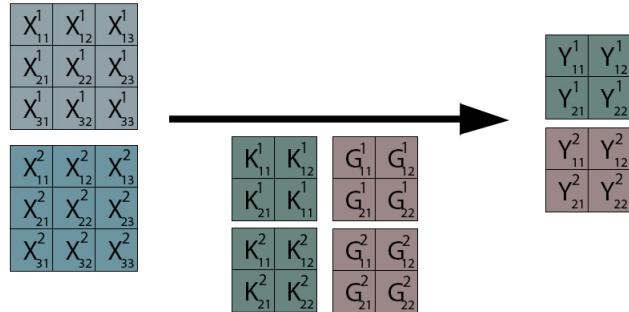
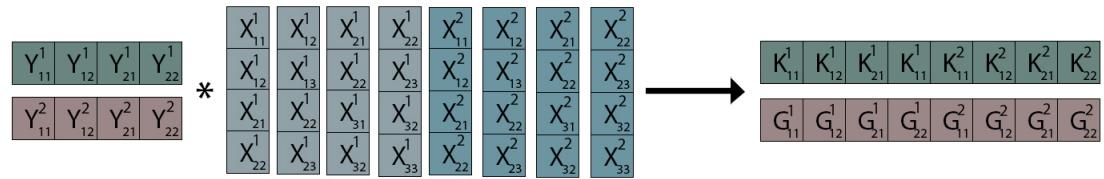
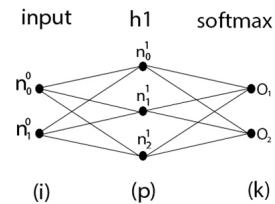
ESTÁNDAR**GEMM**

Figura 4.3: Retropropagación en una capa convolucional de forma estándar frente a GEMM respecto a los pesos

4.3. Capa totalmente conectada como GEMM [5]

4.3.1. Propagación hacia delante

ESTÁNDAR



$$a_j^1 = \sum_{i=0}^{i=1} [z_i^0 * w_{i,j}^0] + b_j^1$$

GEMM

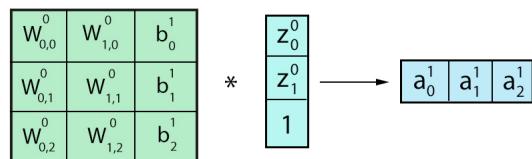


Figura 4.4: Propagación GEMM hacia delante en una capa totalmente conectada

$N = \text{tamaño minibatch}$

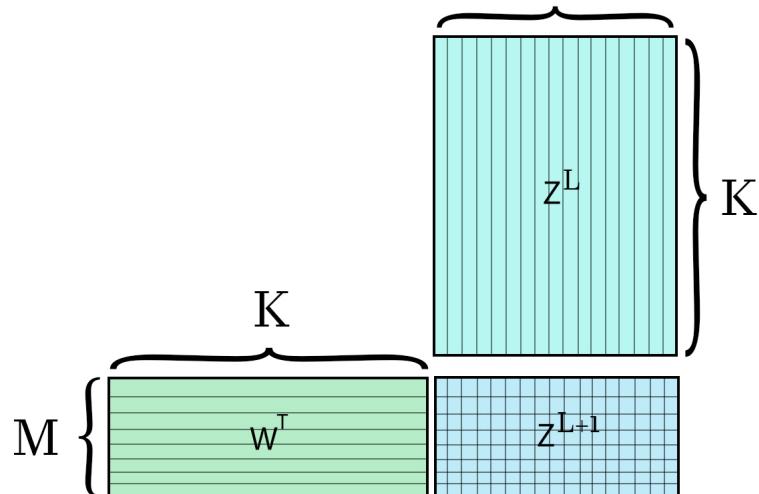


Figura 4.5: Propagación GEMM de un minibatch entero hacia delante en una capa totalmente conectada

4.3.2. Retropropagación

Gradiente respecto a la entrada

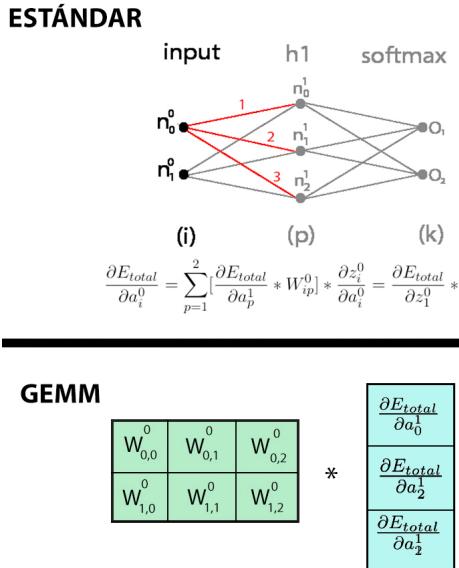


Figura 4.6: Cálculo del gradiente respecto a la entrada en una capa totalmente conectada

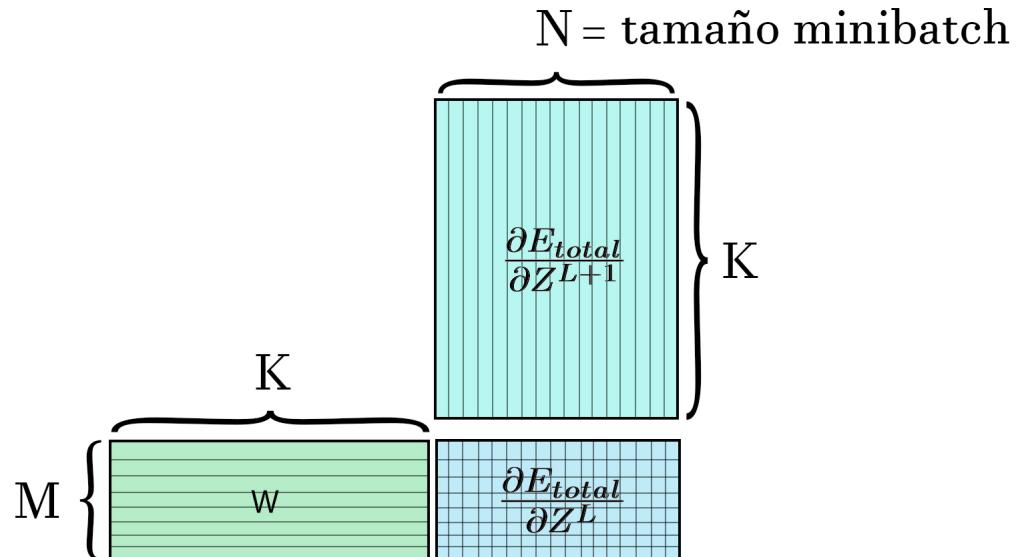


Figura 4.7: Cálculo del gradiente respecto a la entrada de todo un minibatch en una capa totalmente conectada

Gradiente respecto a los pesos

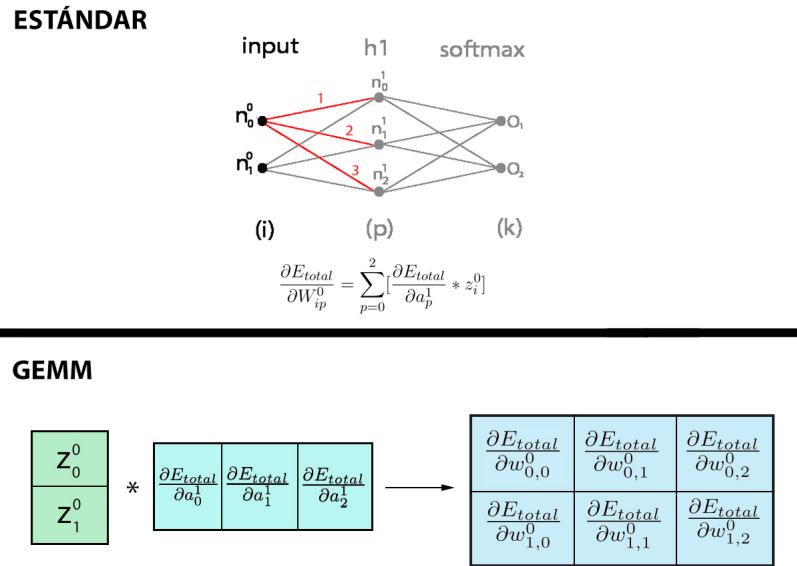


Figura 4.8: Cálculo del gradiente respecto a los pesos en una capa totalmente conectada

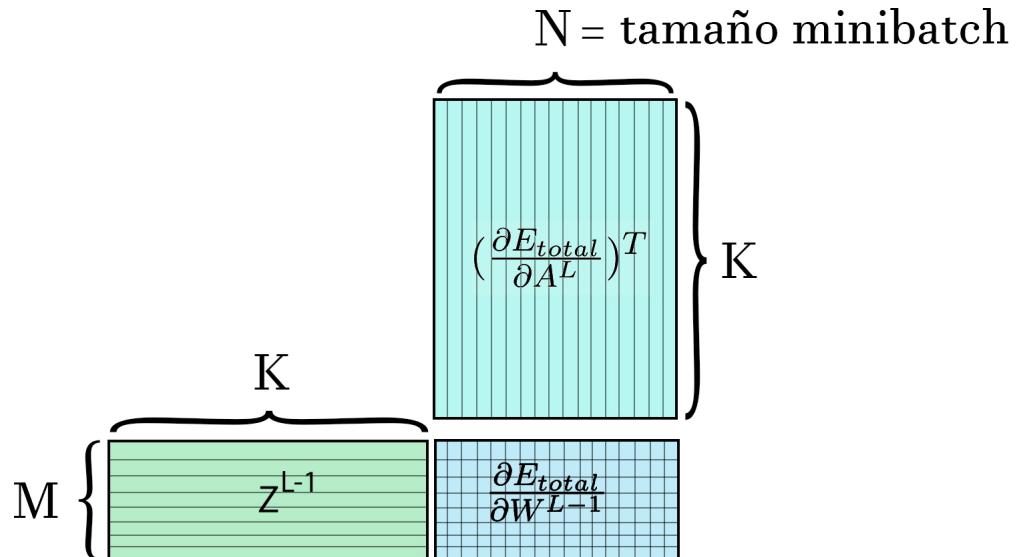


Figura 4.9: Cálculo del gradiente respecto a los pesos de todo un minibatch en una capa totalmente conectada

4.4. CUDA

CUDA es una plataforma de cómputo paralelo de propósito general y modelo de programación que aprovecha la capacidad computacional de las GPUs de NVIDIA para resolver gran cantidad de problemas computacionalmente costosos de manera eficiente. CUDA permite acceder a la GPU de forma similar al común acceso a CPU que experimenta un desarrollador promedio.

El modelo de programación CUDA permite ejecutar aplicaciones en sistemas de computación heterogéneos, caracterizándose estos por CPUs y GPUs, cada uno con su propia memoria separada por un bus PCI-Express.

Cuando se lanza una función kernel desde CPU, la ejecución se traslada a GPU, esta genera un gran número de hebras y cada una de ellas ejecuta las órdenes especificadas en dicho kernel.

Las hebras se organizan en una cuadrícula o grid compuesta por varios bloques de hebras. De esta forma, cada hebra pertenece a un bloque de hebras, y cada bloque de un mismo kernel pertenece a un mismo grid. Todas las hebras de un mismo grid comparten la misma memoria global. Por tanto, hebras de distintos grids no pueden cooperar. A su vez, las hebras de un mismo bloque se pueden sincronizar y compartir memoria a nivel de bloque, siendo esta más escasa pero presentando una latencia considerablemente menor que la memoria global. CUDA organiza los grids y bloques mediante estructuras que pueden ser 1D, 2D o incluso 3D [6].

4.4.1. Multiplicación de matrices en CUDA

Dadas dos matrices $A(M \times K)$ y $B(K \times N)$, se obtiene $C(K \times N)$ como producto de la multiplicación AB . Basándome en mi propia experiencia, una primera aproximación de cara a realizar dicha operación en CUDA consiste en crear un bloque $Block(K \times N)$, de forma que cada hebra calcule una posición de la matriz resultado C . Dada la naturaleza extremadamente simple de tal implementación, el desarrollador que la llevó a cabo optará por buscar formas de reducir el tiempo de cómputo requerido por la misma, optando en la mayoría de los casos por emplear memoria compartida de bloque. Tras desarrollar esta “segunda versión”, el siguiente paso del desarrollador consiste en comparar ambas implementaciones para verificar si efectivamente se obtiene una ganancia notable sobre la primera. En el proceso se observa como para matrices de tamaño reducido ambas implementaciones parecen funcionar. Sin embargo, a medida que aumenta el tamaño de las mismas se ve un claro “tope” pues ambas comparten un mismo defecto, el tamaño de bloque. Por ejemplo, para calcular $C(50 \times 50) = A(50 \times 10) \times B(10 \times 50)$ ambas implementaciones requerirían un bloque de $50 \times 50 = 2500$ hebras, lo cual es imposible pues excede el tamaño límite (CUDA solo permite 1024 hebras por bloque).

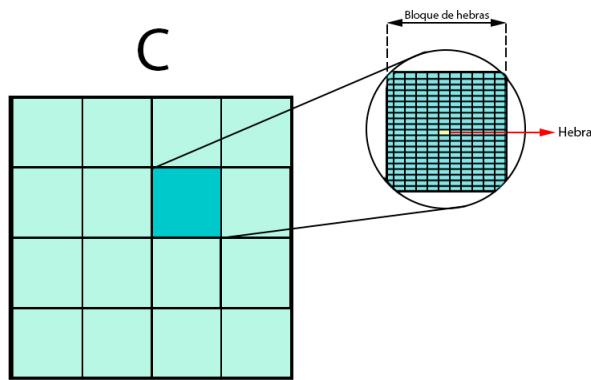


Figura 4.10: Tercera implementación de multiplicación matricial con CUDA

Por tanto, resulta evidente la existencia de una “tercera” implementación carente de este gran defecto. Esta se caracteriza por la división la matriz C en submatrices o tiles, de forma que cada tile sea un bloque cuda. Partiendo de las implementaciones anteriores, el cambio resulta inmediato y los resultados descomunales pues aporta la capacidad de multiplicación matricial con independencia del tamaño de A y B, además de mejoras en cuanto a rendimiento [46].

Sin embargo, esta última implementación carece de los beneficios de la memoria compartida nivel de bloque, por lo que una vez más indica la muy posible existencia de una mejora sobre la misma.

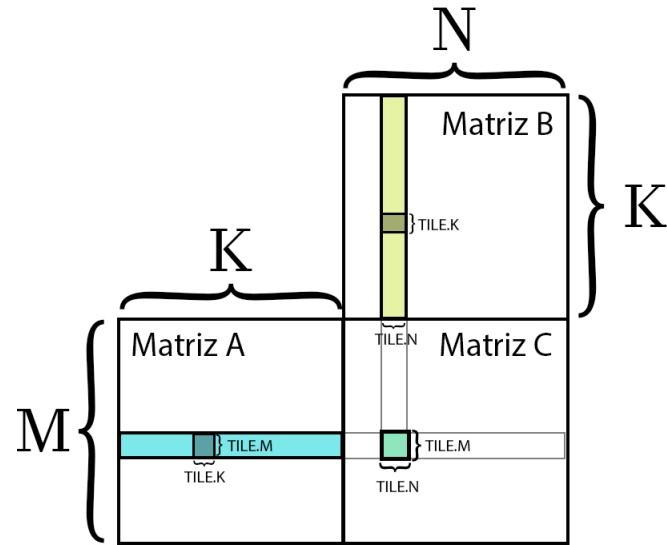


Figura 4.11: Cuarta implementación de multiplicación matricial con CUDA

Para realizar el cálculo de cada valor de C es necesario multiplicar una fila de A por una columna de B. Es decir, multiplicar dos vectores de K elementos. Una idea inicial consiste en cargar 2K elementos en memoria compartida para posteriormente usarlos. Sin embargo, si cada hebra de cada bloque requiere 2K elementos, esto no es viable pues la memoria compartida es limitada y escasa. En su lugar, un mejor enfoque consiste en, dado un bloque 2D de dimensiones TILExTILE, almacenar 2*TILE*TILE. Su objetivo es dividir el cálculo de cada valor de C en iteraciones, y en cada una de ellas, cada hebra multiplica TILE elementos de A por TILE de B, además de acumular y sumar los resultados obtenidos en cada iteración [47].

Capítulo 5

Comparación con distintas plataformas

5.1. cuDNN

Bibliografía

- [1] mehran@mldawn.com. Back-propagation through cross-entropy softmax, 2021. <https://www.mldawn.com/back-propagation-with-cross-entropy-and-softmax/> [Accessed:29/02/2024].
- [2] mehran@mldawn.com. The derivative of softmax function wrt z, 2021. <https://www.mldawn.com/the-derivative-of-softmaxz-function-w-r-t-z/> [Accessed:05/03/2024].
- [3] Prakash Jay. Back-propagation is very simple. who made it complicated ?, 2017. <https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c> [Accessed:06/03/2024].
- [4] Chamanth mvs. No more confusion on back-propagation, 2022. <https://pub.aimind.so/no-more-confusion-on-backpropagation-7adfc271539f> [Accessed:07/03/2024].
- [5] NVIDIA. Linear/fully-connected layers user's guide, 2024. <https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html> [Accessed:11/05/2024].
- [6] Ty McKercher John Cheng, Max Grossman. *Professional Cuda C Programming*. John Wiley and Sons, Inc., 10475 Crosspoint Boulevard, 1 edition, 2014.
- [7] Izzat El Hajj Wen-emi W.Hwu, David B.kirk. *Programming Massively Parallel Processors*. Morgan Kaufmann, 50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States, 4 edition, 2022.
- [8] Hsuan-Tien Lin Yaser S. Abu-Mostafa, Malik Magdon-Ismail. *Learning From Data*. California Institute of Technology Pasadena, CA 91125, USA, 1 edition, 2012.

- [9] Douglas Karr. Unidad lineal rectificada, 2024. <https://es.martech.zone/acronym/relu/> [Accessed:25/02/2024].
- [10] Javi. La función sigmoide: Una herramienta clave en redes neuronales, 2023. <https://jacar.es/la-funcion-sigmoide-una-herramienta-clave-en-redes-neuronales/> [Accessed:25/02/2024].
- [11] Jason Brownlee. Softmax activation function with python, 2020. <https://machinelearningmastery.com/softmax-activation-function-with-python/> [Accessed:24/02/2024].
- [12] Saurav Maheshkar. What is cross entropy loss? a tutorial with code, 2023. <https://wandb.ai/sauravmaheshkar/cross-entropy/reports/What-Is-Cross-Entropy-Loss-A-Tutorial-With-Code--VmlldzoxMDA5NTMx#:~:text=Cross%20entropy%20loss%20is%20a,close%20to%200%20as%20possible.> [Accessed:29/02/2024].
- [13] Descenso del gradiente, 2024. https://es.wikipedia.org/wiki/Descenso_del_gradiente#:~:text=El%20descenso%20del%20gradiente%20o,en%20direcci%C3%B3n%20contraria%20al%20gradiente. [Accessed:26/02/2024].
- [14] Gradiente, 2024. <https://es.wikipedia.org/wiki/Gradiente> [Accessed:26/02/2024].
- [15] Robert Kwiatkowski. Gradient descent algorithm — a deep dive, 2021. [https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21#:~:text=Gradient%20descent%20\(GD\)%20is%20an,e.g.%20in%20a%20linear%20regression\).](https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21#:~:text=Gradient%20descent%20(GD)%20is%20an,e.g.%20in%20a%20linear%20regression).) [Accessed:26/02/2024].
- [16] ml4a. How neural networks are trained, 2020. https://ml4a.github.io/ml4a/how_neural_networks_are_trained/ [Accessed:27/02/2024].
- [17] Sebastian Raschka. How is stochastic gradient descent implemented in the context of machine learning and deep learning?, 2024. <https://sebastianraschka.com/faq/docs/sgd-methods.html> [Accessed:27/03/2024].
- [18] Wikipedia. Stochastic gradient descent, 2024. https://en.wikipedia.org/wiki/Stochastic_gradient_descent [Accessed:22/03/2024].

- [19] Aishwarya V Srinivasan. Stochastic gradient descent — clearly explained !!, 2019. <https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31> [Accessed:22/03/2024].
- [20] Jorge Calvo Martin. La importancia de las funciones de activación en una red neuronal, 2022. <https://www.linkedin.com/pulse/la-importancia-de-las-funciones-activaci%C3%B3n-B3n-en-una-red-calvo-martin/> [Accessed:10/06/2024].
- [21] Miguel Sotaquirá. La función de activación, 2018. <https://www.codificandobits.com/blog/funcion-de-activacion/> [Accessed:10/06/2024].
- [22] Jason Brownlee. Weight initialization for deep learning neural networks, 2021. <https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/> [Accessed:14/03/2024].
- [23] Sandeep Jain. Kaiming initialization in deep learning, 2023. <https://www.geeksforgeeks.org/kaiming-initialization-in-deep-learning/> [Accessed:14/03/2024].
- [24] Adrian Rosebrock. Understanding weight initialization for neural networks, 2021. <https://pyimagesearch.com/2021/05/06/understanding-weight-initialization-for-neural-networks/> [Accessed:14/03/2024].
- [25] Yahia Zakaria. Initial bias values for a neural network, 2017. <https://stackoverflow.com/questions/44883861/initial-bias-values-for-a-neural-network> [Accessed:14/03/2024].
- [26] Glen Meyerowitz. Bias initialization in a neural network, 2018. <https://medium.com/@glenmeyerowitz/bias-initialization-in-a-neural-network-2e5d26fed0f0> [Accessed:14/03/2024].
- [27] Afshine Amidi y Shervine Amidi. Convolutional neural networks cheatsheet, 2018. <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks> [Accessed:19/03/2024].
- [28] Stanford. Convolutional neural networks (cnns / convnets), 2020. <https://cs231n.github.io/convolutional-networks/> [Accessed:19/03/2024].

- [29] savyakhosla. Cnn — introduction to padding, 2024. <https://www.geeksforgeeks.org/cnn-introduction-to-padding/> [Accessed:02/04/2024].
- [30] Avinash Kumar, Sobhangi Sarkar, and Chittaranjan Pradhan. *Malaria Disease Detection Using CNN Technique with SGD, RMSprop and ADAM Optimizers*, pages 211–230. 01 2020.
- [31] Archana David. Backprop through max-pooling layers?, 2007. <https://datascience.stackexchange.com/questions/11699/backprop-through-max-pooling-layers> [Accessed:24/03/2024].
- [32] Muhammad Baqir. How to backpropagate through max-pooling layers, 2024. <https://www.educative.io/answers/how-to-backpropagate-through-max-pooling-layers> [Accessed:24/03/2024].
- [33] Piotr Skalski. Let's code convolutional neural network in plain numpy, 2020. <https://towardsdatascience.com/lets-code-convolutional-neural-network-in-plain-numpy-ce48e732f5d5> [Accessed:24/03/2024].
- [34] NVIDIA. Nvidia cudnn, 2024. <https://developer.nvidia.com/cudnn> [Accessed:02/08/2024].
- [35] NVIDIA. Nvidia cudnn, 2024. <https://docs.nvidia.com/deeplearning/cudnn/latest/developer/overview.html#dev-overview> [Accessed:02/08/2024].
- [36] NVIDIA. Nvidia cudnn, 2024. <https://docs.nvidia.com/deeplearning/cudnn/latest/developer/core-concepts.html> [Accessed:02/08/2024].
- [37] NVIDIA. Nvidia cudnn, 2024. <https://docs.nvidia.com/deeplearning/cudnn/latest/api/cudnn-cnn-library.html#cudnnconvolutionforward> [Accessed:03/08/2024].
- [38] NVIDIA. Nvidia cudnn, 2024. <https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn-897/api/index.html#cudnnPoolingForward> [Accessed:03/08/2024].
- [39] NVIDIA. Nvidia cudnn, 2024. <https://docs.nvidia.com/deeplearning/cudnn/latest/api/cudnn-cnn-library.html#cudnnconvolutionbackwardfilter> [Accessed:03/08/2024].
- [40] NVIDIA. Nvidia cudnn, 2024. <https://docs.nvidia.com/deeplearning/cudnn/latest/api/cudnn-cnn-library.html#cudnnconvolutionbackwarddata> [Accessed:03/08/2024].

- [41] Vishakh Hegde and Sheema Usmani. Parallel and distributed deep learning. 2016.
- [42] Sunwoo Lee, Dipendra Jha, Ankit Agrawal, Alok Choudhary, and Weikeng Liao. Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, pages 183–192, 2017.
- [43] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *ArXiv*, abs/1404.5997, 2014.
- [44] Hide Inada. Calculate cnn backprop with padding and stride set to 2, 2024. https://hideyukiinada.github.io/cnn_backprop_strides2.html [Accessed:01/04/2024].
- [45] H. Howie Huang¶ Zhufan Wang†‡ Weimin Zheng†‡ †Department of Computer Science Xiaqing Li†‡§, Guangyan Zhang†‡§, Tsinghua University ‡Tsinghua National Laboratory for Information Science Technology §State Key Lab of Mathematical Engineering, China ¶Department of Electrical Advanced Computing, Wuxi, and George Washington University Computer Engineering. Performance analysis of gpu-based convolutional neural networks. *International Conference on Parallel Processing*, 2016.
- [46] Mary Thomas. Comp 605: Introduction to parallel computing lecture : Cuda matrix-matrix multiplication, 2017. <https://edoras.sdsu.edu/~mthomas/sp17.605/lectures/CUDA-Mat-Mat-Mult.pdf> [Accessed:10/05/2024].
- [47] NVIDIA. Matrix multiplication background, user's guide — nvidia docs, 2023. <https://docs.nvidia.com/deeplearning/performance/pdf/Matrix-Multiplication-Background-User-Guide.pdf> [Accessed:10/05/2024].

