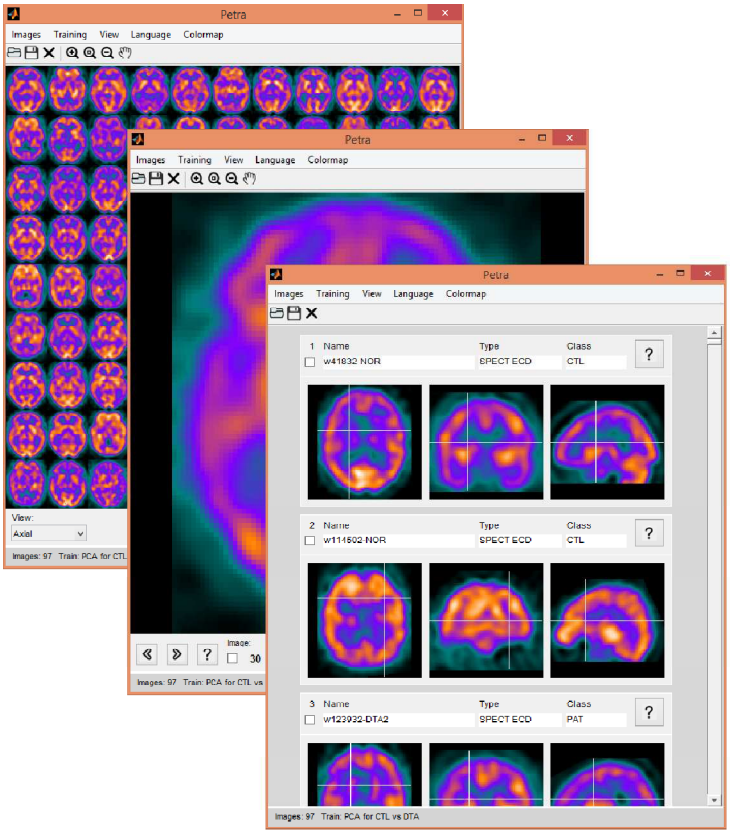
APRENDIZAJE PROFUNDO:

REDES NEURONALES CONVOLUCIONALES

4º Ingeniería Informática en Computación



César Hervás-Martínez
Pedro A. Gutiérrez peña

Grupo de Investigación AYRNA
Departamento de Informática y Análisis Numérico
Universidad de Córdoba
Campus de Rabanales. Edificio Einstein.
e-mail: chervas@uco.es, pagutierrez@uco.es

Redes neuronales profundas

Las redes neuronales profundas (DNN) inspiradas en el comportamiento del cerebro permiten que las computadoras resuelvan tareas cognitivas en las que los humanos sobresalen [1,2]. A falta de explicaciones para tales fenómenos cognitivos, a su vez, los científicos cognitivos han comenzado a utilizar las DNN como modelos para investigar la cognición biológica y sus bases neuronales, creando un importante debate. Aquí, reflexionaremos sobre este modelo desde la perspectiva de la filosofía de la ciencia.

Más allá de su poder para proporcionar predicciones y explicaciones de los fenómenos cognitivos, las DNN tienen el potencial de contribuir a un uso a menudo ignorado, pero omnipresente y fundamental, de los modelos científicos: la exploración.

En los últimos años, las redes neuronales profundas de inspiración neurológica (DNN) han revolucionado la visión por computadora [3] y, posteriormente, otros dominios, como el procesamiento del lenguaje natural [4], el control y la planificación (como juegos, por ejemplo, Atari y Go [5, 6]), y tareas de navegación (como encontrar la ruta más corta en un mapa del metro [7]). Las DNN son modelos computacionales que consisten en considerar muchas unidades de procesamiento simples (similares a las neuronas) que funcionan en paralelo y están dispuestas en capas interconectadas. Las redes neuronales simples constan de una capa de entrada, una capa de salida y en general una capa oculta; cuando se apilan más capas, las redes se llaman profundas [8, 9]. Una DNN aprende a realizar tareas particulares a través del entrenamiento, durante el cual se aprende la fuerza de las conexiones entre las unidades. Posteriormente, una red DNN entrenada se utiliza para realizar la misma tarea con patrones novedosos que forman el conjunto de test

Redes Neuronales Convolucionales (Introducción)

La investigación en redes neuronales artificiales comenzó hace casi 80 años [10]. Durante muchos años, no hubo un modelo biológico ampliamente aceptado para redes neuronales asociadas a la visión, hasta que un trabajo experimental aclaró la estructura y función de la corteza visual de los mamíferos [11]. A partir de entonces, las investigaciones teóricas construyeron modelos que guardan similitud con las redes neuronales biológicas [12]. Si bien el progreso teórico continuó con el desarrollo de métodos para la formación de redes, aún no existía un método práctico para la formación de grandes redes. Con el desarrollo de LeNet, por Yann Le Cunn, et al., Se implementó el primer *convnet* práctico para reconocer dígitos escritos a mano en cheques bancarios [13].

Desde que AlexNet se desarrolló y aplicó a la competición de clasificación de la base de datos ImageNet en 2012 [14], la investigación en redes convolucionales para aplicaciones de aprendizaje profundo ha aumentado exponencialmente. En 2015, el error de clasificación, en esta base de datos, era superior al 5%, esto es, un 6.67%, y se alcanzo con la red GoogLeNet [15], y se redujo a un 3.57%, con la red residual ResNet de Microsoft [16]. En los últimos años, se han desarrollado nuevas arquitecturas de redes neuronales que mejoran las arquitecturas anteriores.

Específicamente, analizaremos los módulos primitivos de GoogLeNet, y las redes residuales en ResNet de Microsoft [16]. También, examinaremos las redes neuronales convolucionales (*convnets*) para el reconocimiento de imágenes, y luego proporcionaremos una explicación de su arquitectura. Se examinará el papel de varios hiperparámetros de las redes *convnet*. Se considerará el problema de cómo dimensionar correctamente una red neuronal, en términos de la cantidad de capas convolucionales y el tamaño de capa. Se presentara un ejemplo para determinar la memoria GPU requerida para entrenar una arquitectura de red definida. Discutiremos el algoritmo de retropropagación del error en el contexto de desarrollos pasados, y recientes, que han mejorado la efectividad del aprendizaje. Se discutirán brevemente otras técnicas y consideraciones relacionadas con el aprendizaje de la red, como elegir una función de activación y una inicialización adecuada de los pesos de la red. Finalmente, se revisaran los desarrollos recientes en arquitecturas *convnet*.

Una red neuronal convolucional (CNN) es un tipo de red neuronal artificial utilizada en el reconocimiento y procesamiento de imágenes que está específicamente diseñada para procesar datos de píxeles.

Las redes CNN son potentes procesadores de imágenes, basadas en inteligencia artificial (AI) que utilizan el aprendizaje profundo para realizar tareas tanto generativas como descriptivas, a menudo con algoritmos que incluyen reconocimiento de imágenes y videos, junto con sistemas de recomendación y procesamiento de lenguaje natural (PNL).

Una red neuronal es un sistema de hardware y/o software de modelado que simula las operaciones de las neuronas del cerebro humano. Las redes neuronales tradicionales no son ideales para el procesamiento de imágenes y deben utilizarse con imágenes de resolución reducida. Las CNN tienen sus "neuronas" dispuestas como las del lóbulo frontal, el área responsable del procesamiento de estímulos visuales en humanos y otros animales. Las capas de neuronas están dispuestas de tal manera que cubren todo el campo visual, evitando el problema de procesamiento poco sistemático de imágenes de las redes neuronales tradicionales.

Una CNN usa un sistema muy parecido a un perceptrón multicapa que ha sido diseñado para reducir los requisitos de procesamiento. Las capas de una CNN consisten en una capa de entrada, una capa de salida y unas capas ocultas que incluye múltiples capas convolucionales, capas de agrupación, capas completamente conectadas y capas de normalización. La eliminación de las limitaciones y el aumento de la eficiencia para el procesamiento de imágenes dan como resultado un sistema mucho más efectivo, más sencillo para el procesamiento de imágenes y el procesamiento del lenguaje natural.

Arquitectura de una red convolucional "convnet"

Hasta que no se pudieron realizar operaciones matriciales en paralelo en las unidades de procesamiento gráfico (GPU) en computadoras de escritorio, no fue posible desarrollar redes más grandes para clasificar imágenes con un gran número de clases, tomadas de ImageNet [17]. Las redes neuronales de grandes dimensiones tienen la capacidad de emular el comportamiento de funciones arbitrarias, complejas y no lineales. Cuando se entrenan de manera efectiva, las funciones emuladas por *convnet* tienen la capacidad de mapear datos de imágenes complejas y de alta dimensión en un espacio dimensional mucho más bajo de categorías definidas finitas, que consta de cientos o miles de clases de objetos. La distinción entre categorías puede ser mejorada, por ejemplo, un gato, en lugar de una pantera (ver Fig 1).

Figura 1: Un gato puede, en muchos casos, no ser discernible con facilidad de un lince

Las redes también se utilizan, entre otras tareas, para la detección de objetos, el reconocimiento de escenas, la estimación de posturas humanas, la generación de subtítulos de video, el reconocimiento de voz, la traducción de idiomas, etc. La idea de usar filtros locales (en lugar de capas totalmente conectadas) con *convnet* surgió primero de la observación de que, en imágenes, no hay una estructura significativa en escalas de distancia grande. "Más grande" es un término relativo aquí, que depende de la composición de la imagen. A grandes distancias, los píxeles están relacionados al azar. Pero, a distancias más cortas, se correlacionan sus valores.

Esto indica que se puede encontrar una estructura significativa dentro de los parches de las imágenes locales. Esto no es sorprendente, ya que los objetos están definidos por su estructura local y no por lo que está lejos de ellos. Un jarrón, por ejemplo, puede ser definido por sus contornos. Los píxeles alejados del jarrón no proporcionan más información sobre el jarrón, a menos que por coincidencia siempre se coloque al lado de un cubo pequeño. Con tal conjunto de imágenes, un clasificador de imágenes entrenado para reconocer jarrones también puede usar elementos del cubo para determinar si hay un jarrón en la imagen.

De hecho, antes del entrenamiento, no está claro qué tipo de características seleccionará una red para mejorar el reconocimiento. Una red entrenada por [18], por ejemplo, podría reconocer los coches de carreras, en parte, al desarrollar detectores para el contenido de texto de las pegatinas publicitarias pegadas a los lados de ellos, lo que no es necesariamente algo que uno asociaría de inmediato con un automóvil. Pero como es una característica distintiva de los coches de carreras, la red aprendió a reconocer el texto de los anuncios, y luego usó esta característica, en una representación distribuida, para activar la salida de red que indica la categoría de cada coche de carreras, y no otro tipo de coche.

La información estructural contenida en regiones locales de las imágenes motivó el uso de conexiones de tipo parches entre capas, en lugar de conexiones completas. Esto es lo mismo que usar conexiones donde cada peso es cero, excepto, posiblemente, para pesos dentro de alguna región del parche (filtro o *kernel*). Los ceros representan el hecho de que la información fuera del parche no determina nada sobre la estructura presente en el área local del filtro.

Solo las neuronas dentro de un (cuadrado) parche/ filtro/ *kernel* estarían completamente conectadas a las neuronas individuales en la siguiente capa (ver Figura 2). Al reducir el número de conexiones, se reduce el número de pesos o conexiones de la red. Esto reduce la computación, tanto en entrenamiento como durante la generalización. Además, los filtros mantienen los mismos pesos, ya que convolucionan en varias posiciones formando un mapa de características. Esto significa que un filtro está utilizando los mismos pesos para detectar el mismo tipo de entidad en varias ubicaciones de un mapa de características. Esto ayuda a reducir aún más el número de parámetros y también ayuda a prevenir el sobre-entrenamiento de la red en el conjunto de test. El uso de filtros locales (conexiones de parches) en lugar de conexiones completas también reduce el sobre-entrenamiento.

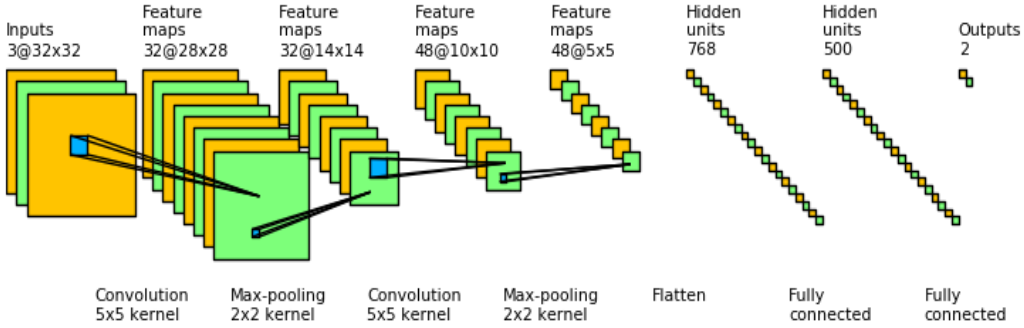


Figura 2: Una red neuronal convolucional con capas de *max-pooling*. Aquí, la función *max-pooling* elige el valor de píxel más alto en un parche de 2 x 2 trasladado en incrementos de 2 píxeles. Esto reduce la cantidad de píxeles por un factor de 4.

Filtro de convolución

La forma en la que un filtro se convierte en una imagen depende de varios parámetros, que se describen aquí. Al determinar los valores apropiados de tamaño de filtro, zancada y relleno de ceros (descritos a continuación), se debe realizar una aritmética básica para determinar si la convolución funcionará con las dimensiones del mapa de características. La Figura 3 muestra un ejemplo de cómo se calcula una convolución con un filtro en las tres posiciones superiores del filtrado en una imagen.

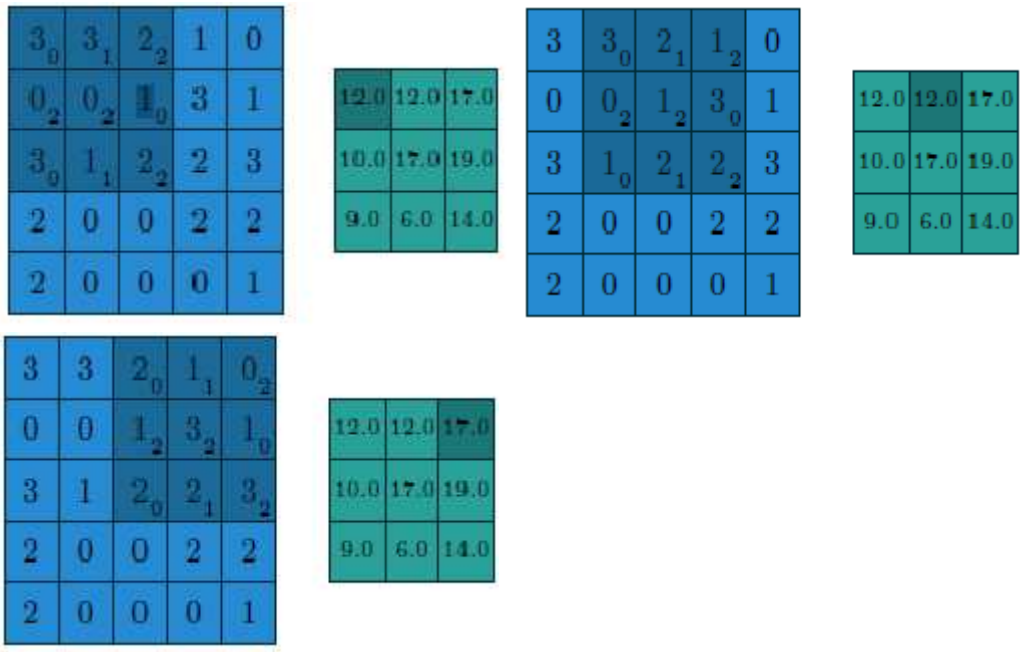


Figura 3: Cálculo de los valores de salida de una convolución discreta. Para las subparcelas, los cuadrados azul marino indican neuronas en la región del filtro. Los

cuadrados de color verde oscuro indican la neurona de salida para la que se calcula la activación total de entrada

Tamaño del filtro

Casi todos los filtros encontrados en la literatura son cuadrados. El tamaño del filtro con respecto al tamaño de la imagen (o capa de activación), determina qué características pueden ser detectadas por los filtros. En AlexNet, por ejemplo, la capa de entrada tiene filtros de tamaño 11x11, aplicados a imágenes que tienen 224 x 224 píxeles. La longitud del lado de cada filtro comprende solo el 4.3% de la longitud del lado de la imagen total (cuadrada). Estos filtros en las primeras capas no pueden extraer características que abarquen más del 0.24% del área de la imagen de entrada.

Relleno de la imagen

El relleno de la imagen es un conjunto de pixeles con valor 0 que se colocan alrededor de la imagen. La profundidad de relleno se puede configurar para que la salida de la capa convolucional actual no se haga más pequeña en tamaño después de la convolución. La figura 4 muestra este efecto. Con muchas capas convolucionales sucesivas, la reducción en la dimensión de salida puede convertirse en un problema, ya que algún área se pierde en cada convolución. En una red *convnet* que tiene muchas capas, este efecto de reducir el tamaño de la salida se puede contrarrestar con el relleno con ceros de la entrada. El relleno con ceros, como se muestra en la Figura 4, puede tener el efecto de cancelar la reducción dimensional y mantener la dimensión de entrada en la salida. En redes que tienen muchas capas, este enfoque puede ser necesario para evitar que los resultados se reduzcan demasiado..

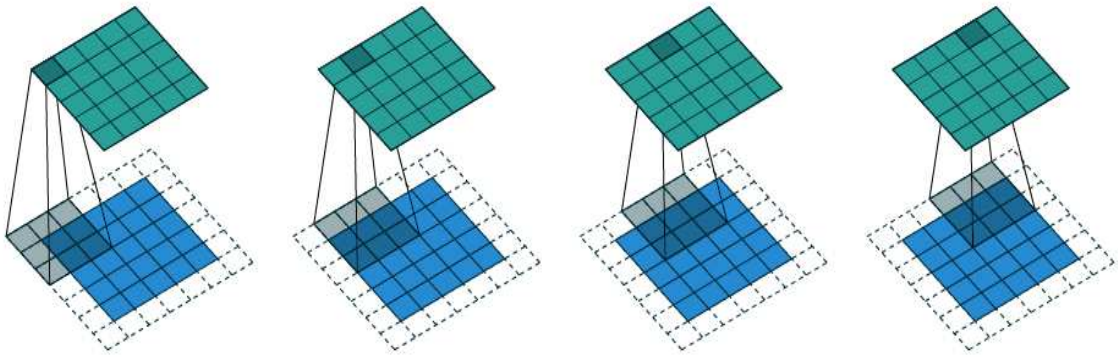


Figura 4: Con relleno con ceros, el efecto de la reducción del tamaño de salida se contrarresta para mantener el tamaño de la imagen de entrada idéntico al de la salida.

***Stride* o tamaño de desplazamiento (zancada)**

Usualmente denotado como S, es un parámetro que describe cuántos píxeles se trasladara un filtro horizontalmente (y luego verticalmente, mientras se convierte en la siguiente fila). En algunas arquitecturas de red, se utiliza *stride* en lugar de la agrupación máxima o *max-pooling* para reducir el tamaño de capa. Este enfoque fue utilizado en GoogLeNet [15].

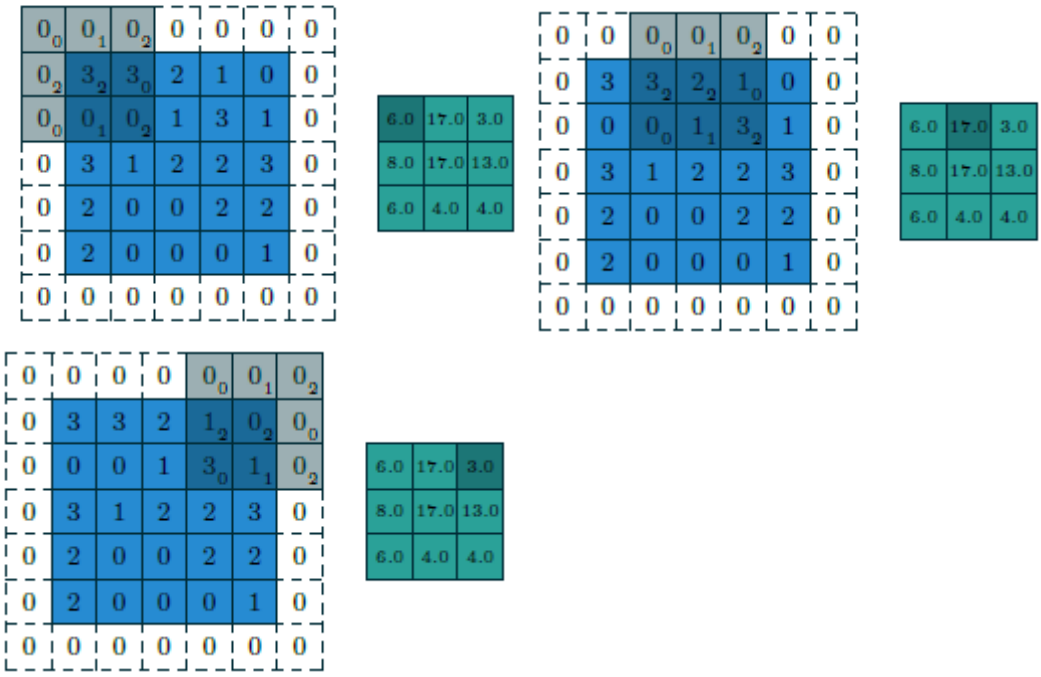


Figura 5.- Convolución con relleno con ceros y zancada > 1. Aquí solo se muestran las primeras tres subparcelas. El relleno es de profundidad 1, la zancada es 2 y el tamaño del filtro es 3x3.

Aritmética de la convolución

Al considerar el diseño de una red, es necesario saber cómo afectarán los parámetros a la dimensión de salida. Usaremos la variable N como la longitud del lado de la capa de activación actual. P es la profundidad de relleno de ceros. F es la longitud del lado del filtro. S es la longitud de zancada. Las siguientes ecuaciones relacionan el tamaño de la salida con el tamaño de la entrada, el tamaño del filtro, el relleno y la zancada.

$$L_{output} = ((N - F) /S) + 1 \quad (1)$$

$$L_{output} = ((N + 2P - F) / (S) + 1 \quad (2)$$

La ecuación (1) calcula la longitud del lado de una capa de activación después de la convolución sin relleno de ceros. La ecuación (2) se obtiene cuando rellenamos con ceros. El relleno con ceros puede contrarrestar la reducción normal del área de salida

después de la convolución. Esto se discute con más detalle más adelante en el epígrafe 'Determinación de los recursos de cómputo requeridos para una red virtual'.

Por razones computacionales, es mejor establecer el número de filtros para que sea una potencia de 2. Esto ayuda a vectorizar los cálculos. Los tamaños de filtro suelen ser números impares. Esto es para tener un píxel central del filtro. De esta manera, el filtro se aplica a alguna posición que existe en el tensor de entrada.

Hay que tener en cuenta que después de convolucionar una imagen de 5 x 5 píxeles con un filtro de 3 x 3 en la Figura 3, la salida (capa de activación) tiene dimensiones de 3 x 3 píxeles, lo que es consistente con la Ecuación 1.

Agrupación "Pooling"

Las capas de agrupación se desarrollaron para reducir la cantidad de parámetros necesarios para describir capas más profundas en la red. La agrupación también reduce la cantidad de cálculos necesarios para entrenar la red o simplemente para ejecutarla hacia adelante durante una tarea de clasificación. Las capas de agrupación proporcionan una cantidad limitada de invariancia traslacional y rotacional. Sin embargo, no tendrá en cuenta las grandes perturbaciones traslacionales o rotacionales, como una cara invertida en 180º.

Perturbaciones como estas solo podrían detectarse si las imágenes de caras, giradas 180º, estuvieran presentes en el conjunto de entrenamiento original. Tener en cuenta todas las orientaciones posibles de un objeto requeriría más filtros y, por lo tanto, se requerirían más parámetros o pesos.

Sin embargo, una cuestión más importante es si una simple *convnet* podría ser la arquitectura de red más apropiada para apuntar a varias representaciones de la misma clase de objeto. En 2015, Google desarrolló el módulo Inception, que era un módulo arquitectónico diseñado para detectar características con cierta invariancia de escala [15]. Esto se discute más adelante. El módulo de inicio es una extensión de arquitecturas de red convolucionales anteriores. Se han desarrollado otros módulos de red funcionalmente distintos para otros fines, como retener información en una red neuronal recurrente. Ejemplos de esto incluyen el módulo de memoria a corto y largo plazo (LSTM) y la unidad recurrente cerrada (GRU) [19, 20]. Aquí se especula que se pueden desarrollar módulos más efectivos para capturar la invariancia traslacional y rotacional.

La Figura 6 muestra la agrupación por máximos, *maxpooling*, aplicada en 3 x 3 parches. Debido a que la agrupación máxima aumenta la intensidad de los píxeles, uno podría

esperar que una mejor manera de agrupar sería mantener la intensidad promedio por píxel igual. Sin embargo, se descubrió que este no es generalmente el caso.

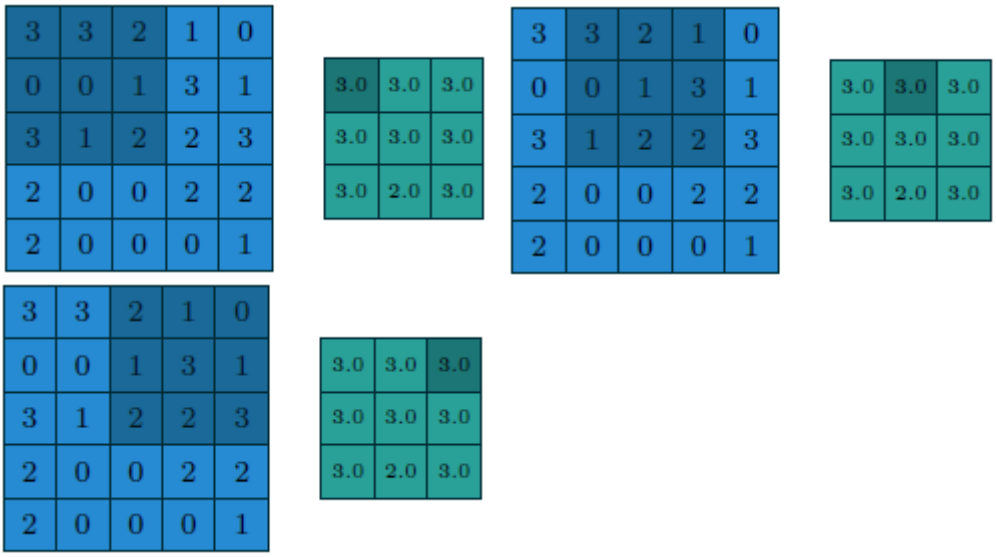


Figura 6: Este es un ejemplo de cómo calcular la salida de *maxpool* en tres ubicaciones de parches de 3 x 3 en una capa de activación. La combinación máxima se aplica en parches de 3 x 3. La agrupación máxima aumenta la intensidad de los píxeles en la agrupación, ya que solo se seleccionó el píxel de intensidad máxima de cada área agrupada.

Algunas arquitecturas de *convnet*, como GoogLeNet, están reemplazando las capas de agrupación máxima mediante el uso de pasos más grandes.

Cuestiones acerca del diseño de red

1.¿Cómo se determina un número adecuado de filtros?
2. Dado un número de categorías de imágenes (clases) y el número de imágenes por clase, ¿Cuál es la capacidad de aprendizaje de una red neuronal?
3. ¿Cómo se pueden expresar las tareas de clasificación de imágenes en términos de requisitos de capacidad de aprendizaje?

Más clases implica que se deben aprender más características. Más características requieren más filtros y más filtros requieren más parámetros. Parece que la profundidad de la red debe estar determinada por alguna medida de complejidad común a casi todas las imágenes en el conjunto de datos. Las redes residuales funcionan mejor en cantidades arbitrariamente grandes de capas. La relación especulada aquí entre un número suficiente de capas y una medida de complejidad de las imágenes en el conjunto de datos se aplica a las redes *convnet* y no a las redes residuales. Si el número de capas ocultas requeridas pudiera determinarse desde el principio, solo el número y tamaño de

los filtros para cada capa quedaría por determinar. El número de clases determinaría principalmente el número de filtros. Sin embargo, debido a la efectividad de la representación distribuida, el número de filtros requeridos no aumentará linealmente con el número de clases. En cambio, puede aumentar proporcionalmente a alguna potencia fraccional del mismo.

Durante el entrenamiento, los parches en las capas iniciales de la red, generalmente se utilizan como detectores de borde [14]. Esto ayuda a la red a construir representaciones más complejas en los filtros de la siguiente capa. Además, el pequeño tamaño de los filtros utilizados en la primera capa les deja poca flexibilidad, pero para elegir solo estructuras pequeñas, casi sin características, como bordes o manchas de color, por ejemplo. Los filtros para las siguientes capas se convierten en combinaciones de características del filtro de la capa actual.

Entrenamiento *Convnet*

Durante el entrenamiento de la red, los pesos de los filtros se ajustan para mejorar el rendimiento de clasificación de la red. Esto se puede hacer usando el algoritmo de retropropagación del error, donde el gradiente de una función de error se calcula con respecto a todos los pesos de la red, yendo desde las conexiones de la capa de salida hasta las conexiones de entrada de la red. Los pesos de la red se actualizan mediante la siguiente ecuación que relaciona el tamaño del paso, la magnitud del cambio de los pesos, con el gradiente y con la tasa de aprendizaje, η.

$$W_{new} = W - \eta \frac{dE}{dW} \tag{3}$$

Una función de error (o "puntuación") puede expresarse como una suma de diferencias al cuadrado entre la salida de la red y la salida correcta, sobre todos los puntos discretos en la salida. Este tipo de función de puntuación funciona para casos en los que la salida de la red es un vector, matriz o tensor de valores reales continuos.

$$E(W,b) = \frac{1}{2N} \sum_{i=1}^{N} \left\| h_{W,b}(x_i) - y_i \right\|^2 \tag{4}$$

El gradiente de esta función de error se situaría en la Ecuación 3. Para las redes de clasificación, la función de error se calcula de manera diferente. Debido a que la salida es un vector uno de N, donde el componente de mayor valor representa la mejor estimación de la red de la clase de la imagen (es decir, automóvil, barco, avión, etc.), se necesita una función de error para la salida categórica. A esta función se la denomina

entropía cruzada [21], y se utiliza para estimar el error en salidas categóricas, como con las redes *convnets* de clasificación de imágenes.

Métodos de optimización

Los nombres de los métodos de optimización, tanto antiguos como nuevos, se enumeran a continuación.

Descenso estocástico de gradiente (SGD)

SGD se usa con el procesamiento por lotes, donde el gradiente de un lote se promedia y se usa para realizar actualizaciones de los pesos de la red.

SGD con parámetro de momento

Cuando se agrega un parámetro de momento, se reduce la fluctuación en las direcciones del gradiente, así, con el momento se construyen direcciones poco profundas.

Gradiente Acelerado de Nesterov (NAG)

NAG evalúa la siguiente posición indicada por el momento. Esto se llama un gradiente "anticipado". En la práctica, NAG casi siempre converge más rápido que SGD con momento.

AdaGrad

AdaGrad escala el gradiente de forma inversamente proporcional a la raíz cuadrada de la suma del cuadrado de los gradientes anteriores. En la práctica, esto significa que puede tener una mayor tasa de aprendizaje que en las direcciones empinadas. Dado que la suma del cuadrado del gradiente anterior solo aumentará, la tasa de aprendizaje con AdaGrad finalmente cae a cero y el aprendizaje de la red se detiene.

RMSProp

RMSProp es similar a AdaGrad, excepto que se aplica un parámetro de disminución a la suma, almacenada en caché, de los valores anteriores de gradiente al cuadrado. Esto tiene el efecto de evitar que la tasa de aprendizaje vaya a cero, al reactivar constantemente la búsqueda.

ADAM

ADAM combina el concepto de momento con AdaGrad [22].

El descenso estocástico de gradiente obtiene su nombre de las fluctuaciones aleatorias (estocásticas) del vector de gradiente de la función de error con respecto a los pesos a través de la red mediante un método por lotes. Los gradientes calculados se promedian y las funciones de error de la red se minimizan utilizando el gradiente promedio.

Regularización

La regularización es otro mecanismo para evitar que las redes se sobreajusten a los datos del conjunto de entrenamiento. La técnica de regularización L_2 impone una penalización en la suma de pesos al cuadrado, con un parámetro de disminución de peso, denotado por λ

$$E(W,b) = \frac{1}{2N}\sum_{i=1}^{N}\left\|h_{W,b}(x_i) - y_i\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2 \quad (5)$$

Este tipo de regularización da como resultado pesos más pequeños y distribuidos, lo que reduce el sobreajuste o sobreentrenamiento.

Otra técnica, llamada regularización L_1, impone una penalización a la suma de los valores absolutos de los pesos.

$$E(W,b) = \frac{1}{2N}\sum_{i=1}^{N}\left\|h_{W,b}(x_i) - y_i\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left|W_{ji}^{(l)}\right| \quad (6)$$

La regularización L_1 promueve la disminución de pesos, y muchos pesos terminan siendo cero.

Table 1
Current training optimization algorithms for DL.

Training algorithms	Equation
Batch Gradient Descent (GD)	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \eta \times \frac{\partial Error(w,x,y)}{\partial w_{i,j}^{n_l}}$
Stochastic Gradient Descent (SGD)	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \eta \times \frac{\partial Error(w,x^q,y^q)}{\partial w_{i,j}^{n_l}}$
Mini-batch Gradient Descent	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \eta \times \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$
Mini-batch GD with momentum	$v_{k+1} = \gamma v_k - \eta \times \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$ $w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) + v_{k+1}$
Mini-batch GD with Nesterov acceleration	$v_{k+1} = \gamma v_k - \eta \times \frac{\partial Error((w+\gamma v_k),x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$ $w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) + v_{k+1}$
Regularized Update Descent	$v_{k+1} = \gamma v_k - \eta \times \frac{\partial Error((w+v_k),x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$ $w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) + v_{k+1}$
Adagrad	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \frac{\eta}{\sqrt{G_{k,ij}+\epsilon}} \times \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$ $G_{k,ij} = \sum_{\tau=1}^{k}\left(\frac{\partial Error(w_\tau,x^{q\to q+B},y^{q\to q+B})}{\partial w_{\tau,i,j}^{n_l}}\right)^2$
Adadelta	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \frac{RMS[\Delta w]_k}{RMS[g]_{k+1}} \times g_{k+1}$ $RMS[\Delta w]_k = \sqrt{E[\Delta w^2]_{k+1}+\epsilon}$ $E[\Delta w^2]_{k+1} = \gamma E[\Delta w^2]_{k+1} + (1-\gamma)[\Delta w^2]_{k+1}$ $RMS[g]_{k+1} = \sqrt{E[g^2]_{k+1}+\epsilon}$ $E[g^2]_{k+1} = \gamma E[g^2]_{k+1} + (1-\gamma)[g^2]_{k+1}$ $g_{k+1} = \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$
RMSProp	$v_{k+1} = \gamma v_k + (1-\gamma)\left[\frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]_{k+1}^2$ $w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \frac{\eta}{\sqrt{v_{k+1}+\epsilon}} \times \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$

Adam	$v_{k+1} = \beta_2 v_k + (1-\beta_2)\left[\frac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]_{k+1}^2$ $m_{k+1} = \beta_1 m_k + (1-\beta_1)\left[\frac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]_{k+1}$ $\hat{m}_{k+1} = \frac{m_{k+1}}{1-\beta_1^t}$ $\hat{v}_{k+1} = \frac{v_{k+1}}{1-\beta_2^t}$ $w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \frac{\eta \cdot \hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1}}+\epsilon}$
AdaMax	$u_{k+1} = Max\left(\beta_2 u_k, \left\lvert\frac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right\rvert_{k+1}\right)$ $m_{k+1} = \beta_1 m_k + (1-\beta_1)\left[\frac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]_{k+1}$ $w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \frac{\eta}{(1-\beta_1^t)} \times \frac{m_{k+1}}{u_{k+1}}$

Inicialización de los pesos de la red

Se debe considerar cuidadosamente cómo se inicializan los pesos de una red. Esta consideración debe hacerse, teniendo en cuenta el tipo de función de activación no lineal que se utiliza. Si los pesos se inicializan en una magnitud demasiado grande, entonces una función de saturación no lineal, como tanh o sigmoide, podría saturarse en las regiones de las asíntotas positivas o negativas. Esto es malo porque durante el entrenamiento por retropropagación hace que el gradiente de las funciones no lineales vaya a un valor cercano a cero, para aquellas neuronas que están en la región de saturación. A medida que se encadenan más derivadas cercanos a cero, el gradiente se vuelve esencialmente cero con respecto a cualquier peso, y el entrenamiento de la red por retropropagación no será posible, dado que se estanca.

Un método para inicializar los pesos de la red es distribuir los valores aleatoriamente, a partir de una distribución gaussiana, pero con la varianza inversamente proporcional a la suma de las entradas y salidas de la red [23]. Una red con más neuronas de entrada y salida se inicializaría con pesos aleatorios extraídos de una distribución gaussiana más estrecha, centrada en cero, en comparación con una red con menos neuronas de entrada y salida. Este enfoque fue propuesto para evitar que el gradiente de propagación hacia atrás desaparezca (se haga cero) o explote (tienda a infinito). Sin embargo, en la práctica, el marco de aprendizaje profundo del software *Caffe* ha implementado una expresión ligeramente diferente para la varianza, que es inversamente proporcional a

solo el número de entradas de red [24]. Investigaciones posteriores han descubierto que, cuando se utilizan unidades de activación ReLU, multiplicar el inverso del número de entradas por 2 produce una variación que proporciona los mejores resultados [25].

Eliminación de pesos*, Dropout*

La eliminación de pesos fue una técnica desarrollada en [26], para prevenir el sobreaprendizaje de una red, debido a las variaciones particulares del conjunto de entrenamiento. En las capas completamente conectadas (las últimas de la arquitectura de la red), que son propensas a un sobreentrenamiento, se puede usar la eliminación aleatoria de pesos, donde los nodos en una capa se eliminarán con probabilidad, p. Los nodos que se eliminan no participan en el entrenamiento. Después del entrenamiento, son reemplazados en la red con sus pesos originales. Esto evita que los pesos de las capas completamente conectadas se sobreajusten al conjunto de datos de entrenamiento y mejore el rendimiento sobre el conjunto de test..

Aumento de datos de imagen

El aumento de datos es una forma efectiva de aumentar el tamaño del conjunto de datos de entrenamiento, sin degradar la calidad del conjunto de datos. La capacidad de generalización del conjunto de datos de entrenamiento debido a este aumento produce redes que generalizan mejor que las entrenadas mediante un conjunto de datos de entrenamiento no aumentado. Para aumentar un conjunto de datos de imágenes, las imágenes originales se pueden girar y trasladar horizontal y verticalmente, y las submuestras de las imágenes originales se pueden seleccionar en posiciones aleatorias en el original. Estas imágenes submuestreadas también se les puede dar la vuelta horizontal y verticalmente [14].

Funciones de activación

La relación entre la clase de la imagen (su etiqueta) y los datos de la imagen (sus características) no es lineal. Para que una red neuronal construya la relación no lineal entre los datos y la clase de la imagen, la función de activación debe tener una no linealidad. Sin funciones de activación no lineales, una red neuronal sería capaz solamente de una clasificación lineal.

Se han utilizado funciones de activación tangentes hiperbólicas y sigmoides antes de que se descubriera que las funciones ReLU podían mitigar el problema de la explosión / desaparición del gradiente, al tiempo que reducían el tiempo de entrenamiento. Las ReLU requieren solo dos operaciones de punto flotante para producir una salida.

Unidades lineales rectificadas (ReLU)

La función de rectificación lineal, Rectified Linear Units, ReLU, ([14], [27]), es de la forma

$$y = f_{relu}(z) = \max(0,z).$$

La derivada de esta función coincide con la función escalón utilizada por las neuronas de Mc Culloch y Pitts. Es una función asimétrica puesto que la respuesta a un patrón de entrada inhibidor es 0, esto es, no hay respuesta. Como ventaja esta función permite que una red obtenga fácilmente representaciones dispersas. Por ejemplo, si hacemos una inicialización uniforme de los pesos, por ejemplo en el intervalo [-1, 1] o una Normal (0,1), alrededor del 50% de los valores de salida continuos de las unidades ocultas son ceros y este porcentaje puede aumentar fácilmente con la regularización inducida por la dispersión

Las neuronas lineales rectificadas se limitan a realizar una combinación lineal de sus pesos y entradas, tras lo que generan una salida no lineal utilizando un umbral, de ahí que también reciban el nombre de neuronas lineales con umbral [*linear threshold neurons*].

Al no requerir el uso de funciones trascendentales, como la exponenciación de la función logística o de la tangente hiperbólica, su evaluación es mucho más eficiente y el entrenamiento de redes neuronales con unidades ReLU suele ser mucho más rápido que con neuronas sigmoides.

Además, un modelo numérico, como en el fondo es una red neuronal artificial, generalmente es más sencillo de optimizar cuando su comportamiento es lineal, lo que sucede siempre que se utilizan unidades ReLU. Cuando se activa, una neurona ReLU es equivalente a una simple neurona lineal. Cuando no se activa, permanece completamente inoperante y no interfiere en el comportamiento del resto de la red.

La actividad realmente nula de las neuronas ReLU es algo que también se observa en las redes neuronales biológicas, una característica esencial de un cerebro biológico que le permite ahorrar energía y funcionar de forma más eficiente. El diseño de una unidad ReLU pretende capturar tres propiedades observadas en las neuronas biológicas: i) para algunas entradas, se mantienen completamente inactivas (cuando $z \leq 0$); ii) para otras entradas, su salida es proporcional a su entrada (comportamiento lineal); y, por último, iii) la mayor parte del tiempo, se mantienen inactivas (activaciones dispersas).

Sin embargo, la principal limitación de las unidades ReLU es que son incapaces de aprender cuando su nivel de activación es nulo, al ser su función de activación

completamente nula en esta zona. Por este motivo, conviene inicializar los sesgos de las neuronas ReLU con un valor positivo pequeño (p.e. b = 0.1), para hacer posible que las neuronas se activen durante las etapas iniciales del entrenamiento de la red.

Dado que algunos investigadores argumentan que la pendiente cero evita el aprendizaje de valores negativos [28, 29], otras funciones de activación han sido propuestas en la última década. Así, se han propuesto algunas generalizaciones de las neuronas ReLU con el objetivo de facilitar su entrenamiento, aunque sea a costa de prescindir de su plausibilidad biológica. Usualmente, la función de activación se modifica para que tenga una pendiente no nula (α) para valores negativos de la entrada neta a la neurona, z.
Así, la primera función alternativa a la tradicional ReLU es la LReLU.

$$y = f_{ret}(z) = \max(0, z) + \alpha \min(0, z),$$

donde la rectificación en valor absoluto [30] utiliza $\alpha = -1$ y se emplea en situaciones en las que no importa la polaridad de la señal de entrada,

$$y = f_{aret}(z) = \max(0,z) - \min(0,z) = |z|.$$

En estas situaciones no importa la polaridad de la señal de entrada, como en el reconocimiento de objetos en imágenes.

De este tipo es la función ReLU con pérdidas [*leaky ReLU*] [28], en la que se utiliza un pequeño valor de α p.ej. 0.01, sólo para evitar una derivada nula y permitir el ajuste de los pesos durante el entrenamiento de la red.

$$y = f_{leakyrelu}(z) = \max(0, z) + 0.01 \min(0, z)$$

o también

$$f_l = \begin{cases} z & z > 0 \\ \alpha . z & z \leq 0 \end{cases}, \alpha \text{ es un parámetro a definir previamente}$$

He, Zhang [29] propusieron una unidad lineal rectificada paramétrica (PReLU) que generaliza la ReLU tradicional. La pendiente de la parte negativa, parámetro α en la ecuación anterior por ejemplo, se aprende de los datos en lugar de predefinir su valor. PReLU mejora el ajuste del modelo con un costo computacional adicional casi nulo y un pequeño riesgo de sobreajuste. Su función fp se define como:

$$f_p = \begin{cases} z & z > 0 \\ \alpha_i z & z \leq 0 \end{cases}, \alpha_i \text{ es un parámetro entrenable}$$

o también

$$y_j = f_{prelu}(z, \alpha_j) = \max(0, z) + \alpha_j \min(0, z)$$

donde *i* representa el índice del canal. Se puede utilizar una la versión de PReLU de canal compartido. El canal compartido significa que el coeficiente es compartido por todos los canales de una capa convolucional. El valor inicial de α_i se establece en 0.25. Se actualiza mediante retro-propagación y se optimiza simultáneamente en todas las capas [29][31].

Como sucedía con las funciones de activación logísticas, se pueden diseñar generalizaciones de las unidades lineales rectificadas, como es el caso de las unidades *maxout*. Una unidad de este tipo agrupa sus entradas en subgrupos y combina las salidas para cada subgrupo, lo que equivale a utilizar múltiples conjuntos de pesos en cada neurona. La función trabajos de activación "maxout", que ha logrado un rendimiento de vanguardia en múltiples puntos de referencia de aprendizaje automático (Goodfellow et al., 2013). La función de activación maxout calcula el máximo de un conjunto de funciones lineales y tiene la propiedad de que puede aproximarse a cualquier función convexa de la entrada.

Por tanto, las unidades maxout difieren de las no linealidades tradicionales de la red neuronal en que toman como entrada la salida de múltiples funciones lineales y devuelven la más grande. Esto es

$$f_{\text{maxout}}(z) = \max_{k \in \{1,\ldots,K\}} (w^k . z + b^k)$$

Tanto las funciones ReLU convencionales como las *leaky* ReLU se pueden considerar casos particulares de este tipo de unidades, usando dos conjuntos de pesos: $(w_1, w_2) = (0, w)$ para las unidades ReLU y $(w_1, w_2) = (w, -\alpha w)$ para las *leaky* ReLU. Las neuronas *maxout*, por tanto, gozan de las ventajas de las unidades ReLU (funcionamiento lineal, sin saturación), sin tener sus inconvenientes (activación nula que dificulta su entrenamiento), a costa de duplicar su número de parámetros.

Recientemente Clevert et al. [33] han propuesto utilizar una función de activación con salidas negativas llamadas Unidades Lineales Exponenciales (ELU). La función de activación de ELU se define como

$$f_{ELU}(z_k) = \begin{cases} z_k & \text{if } z_k > 0 \\ \alpha(\exp(z_k) - 1) & \text{if } z_k \leq 0 \end{cases}$$

donde z_k es la salida del filtro en el k-ésimo canal y α es un parámetro para controlar el valor al cual una ELU se satura para entradas negativas. Las derivadas de la función ELU son.

$$\frac{\partial f_{ELU}(z_k)}{\partial w_k} = \frac{\partial f_{ELU}(z_k)}{\partial z_k}\frac{\partial z_k}{\partial w_k} \quad \text{y} \quad \frac{\partial f_{ELU}(z_k)}{\partial z_k} = \begin{cases} 1 & \text{if } z_k > 0 \\ f_{ELU}(z_k) + \alpha & \text{if } z_k \leq 0 \end{cases}$$

Si comparamos las derivadas de ELU y RELU, se observa que las derivadas para z_k negativo son positivas para ELU pero son cero para ReLU. Aunque la función de activación ReLU es efectiva para evitar el problema de la degradación del gradiente, las salidas de la función de activación ReLU en los valores negativos son siempre cero y la media de las salidas de la función de activación ReLU se vuelve positiva. Esto significa que la función de activación ReLU introduce el efecto del cambio de covariable interno.

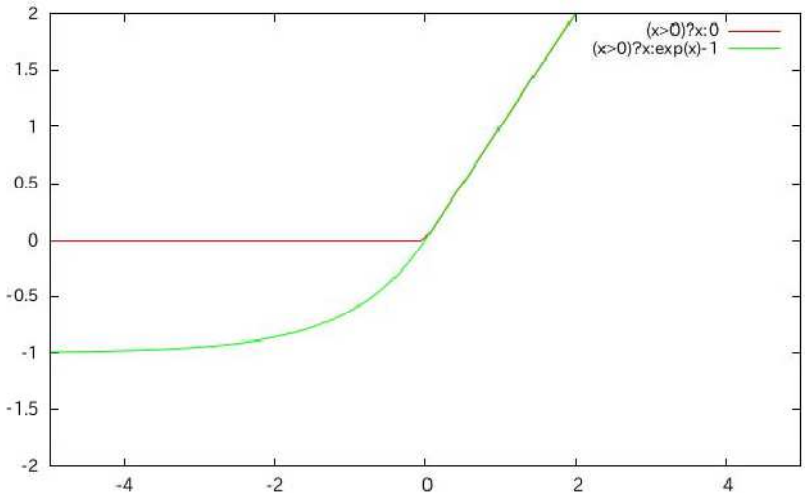


Figura 7.-. El gráfico de las funciones de activación ReLU y ELU. ReLU se muestra con la línea roja y ELU se muestra con la línea verde.

Otro tipo de funciones de activación han sido propuestas [34], donde los autores proponen la función de activación *softplus*(x)= s_+= log $(1+e^x)$ para evaluar el impacto potencial de este efecto. Esta función es una versión suavizada de la función ReLU, de forma tal que en x= 0 la función pasa a ser continua y derivable. De hecho su derivada es la función sigmoide.

Pese a sus mejores propiedades desde el punto de vista teórico, empíricamente suele ofrecer peores resultados que el uso de unidades ReLU, que además resultan más eficientes al prescindir de funciones trascendentales en su evaluación.

Ejemplo para la determinación de los recursos informáticos necesarios para una red convolucional, *Convnet*

Aquí se presenta un ejemplo sobre cómo determinar el número de parámetros que requiere una *convnet* y cómo determinar el número de FLOP requeridos para una ejecución hacia adelante o hacia atrás a través de la red. Sea el *convnet* representado en la Figura 7. para la base de datos CIFAR-1000

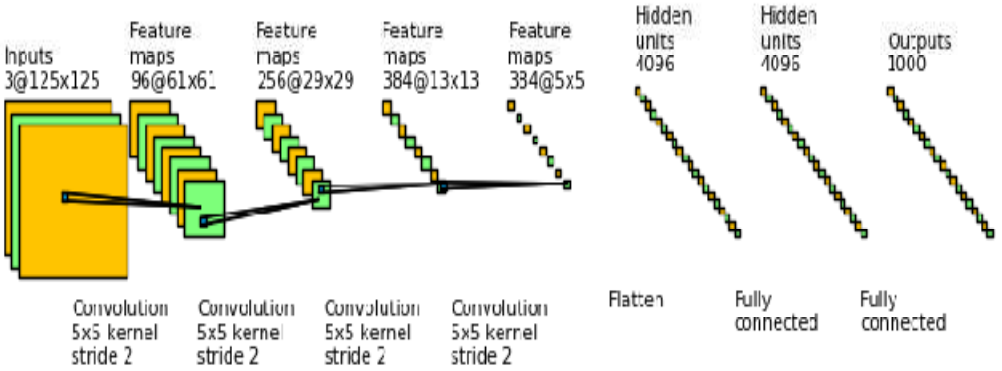


Figura 8: Una red neuronal convolucional con capas de agrupación por máximos reemplazadas por capas de convolución que tienen zancada 2.

La red representada consta de una capa de entrada, una capa de salida y cinco capas ocultas. Las últimas dos capas ocultas están completamente conectadas a la capa anterior. Estas se llaman capas totalmente conectadas o afines. Si se le da a la red una imagen de entrada de 125x125, y los conjuntos de filtros son de tamaños 96, 256, 384 y 384 y de dimensión 5x5; estos deben ser convolucionados en las capas convolucionales primera, segunda, tercera y cuarta, sin relleno de ceros y con zancada 2, entonces los mapas de activación se reducen de dimensión, como se muestra en la tabla expuesta más abajo.

Número de parámetros

Para comenzar a contar el número total de parámetros necesarios para describir una red, consideraremos la red de la Figura 7, la primera convolución se realiza con 96 filtros. Usaremos la variable K para representar el número de filtros. Así

$N_{params} = K (5 \times 5 \times 3 + 1)$

El extra de 3 surge de los tres canales de color para la imagen de entrada.

Tamaños de la capa de activación después de la convolución	
Capas *Convnet*	Tamaño de la capa de activación
Entrada	125x125x3
Capa Conv 1	L_{output} = ((N - F) / S) + 1=((125-5)/2)+1=61, 61x61x96
Capa Conv 2	L_{output} = ((N - F) / S) + 1=((61-5)/2)+1=29, 29x29x256
Capa Conv 3	L_{output} = ((N - F) / S) + 1=((29-5)/2)+1=13, 13x13x384
Capa Conv 4	L_{output} = ((N - F) / S) + 1=((13-5)/2)+1=5, 5x5x384
Capa Completamente Conectada 1	1x1x4096
Capa Completamente Conectada 2	1x1x4096
Salida	1x1x1000

Tabla 1: Debido al paso o zancada de 2, las capas de activación se reducen aproximadamente por un factor de 2 después de cada convolución. En este caso, la agrupación máxima se reemplaza por *convnets* que tienen zancadas > 1.

Las neuronas en cada capa sucesiva tienen un término de sesgo que se agrega a la suma de las 25 entradas ponderadas del filtro. Hay que tener en cuenta que solo hay un sesgo en los tres canales de color. Esto lleva a 126 parámetros por filtro: 125 para los pesos y uno para el término de sesgo. En total, el número de parámetros, N_{params}, es entonces de 12.096 (12.000 pesos y 96 términos de sesgo). Los parámetros de peso describen completamente todos los parámetros 5x5x3+ 1 para cada uno de los 96 filtros.

Requisitos de memoria

El requisito de memoria está determinado por la memoria para las activaciones del mapa de características. La cantidad de memoria para una red es dos veces el total dado en la Tabla 2, porque los gradientes también deben almacenarse a lo largo del paso directo. A medida que la red se activa hacia adelante, las derivadas se pueden calcular en paralelo, y tanto las activaciones de las neuronas como el gradiente de la salida con respecto a los pesos de las neuronas se pueden almacenar en la memoria caché. Los valores almacenados se utilizarán cuando se calcule la propagación hacia atrás durante

el entrenamiento. Algunos métodos de aprendizaje profundo almacenan en caché estos valores durante el pase directo.

Duplicar el valor de la memoria en la Tabla 2 produce 9.22MB como la cantidad de memoria total para una red. En una GPU con 8GB, se podrían encajar 867 redes en GPU a la vez. Sin embargo, hay un límite para el tamaño del lote, más allá del cual no hay mejora en el gradiente promedio. Un tamaño de lote de 256 es lo más común, si la cantidad de la memoria de red cabe en la memoria de la GPU 256 veces.

Antes de que podamos comenzar a estimar el número de parámetros requeridos, debemos considerar varios parámetros relacionados con el filtro en sí. Estos son el paso, el tamaño del filtro y la profundidad de relleno cero.

Tamaños del tensor de salida del mapa de activación			
Capa Convnet	Expresión	Número de parámetros	Memoria necesaria
Capa Conv 1	61x61x96	357.216	1.43MB
Capa Conv 2	29x29x256	215.296	0.86MB
Capa Conv 3	13x13x384	64.896	0.26MB
Capa Conv 4	5x5x384	9.600	0.04MB
Capa Completamente Conectada 1	1x1x4.096	4096	16.4KB
Capa Completamente Conectada 2	1x1x4.096	4096	16.4KB
Salida	1x1x1000	1000	4KB
Total			4.61MB

Tabla 2: Número de activaciones presentes en cada capa y la memoria requerida para los valores de activación de precisión únicos. Observe que las activaciones de red ocupan mucha más memoria hacia la entrada. Multiplique el requisito de memoria por dos para tener en cuenta también la memoria necesaria para almacenar en caché los gradientes durante el paso directo, para una posterior propagación hacia atrás.

Red similar, con más capas y stride S=1

Supongamos por un momento que hay 20 capas convolucionales en el lugar del ejemplo anterior. El tamaño del mapa de activación se reducirá sustancialmente después de 20 convoluciones. Sin embargo, tener un paso 1 significa que las salidas del mapa de activación no disminuirán en tamaño tan rápido como lo hicieron en el caso del paso 2. En cada capa, la dimensión se reduce en F-1, esto es en 4. Después de 20 capas, la imagen de entrada de 125x125 se reducirá a mapas de activación de 45x45 (125 - 20 x 4). La imagen de entrada es en realidad un tensor de 125x125x3. Las tres dimensiones

adicionales surgen de los canales RGB de la imagen de entrada original. Un tensor de entrada de 125x125x3 se transformaría en un tensor de volumen de 109x109x384 después de la cuarta convolución. El multiplicar por 384 adicional aquí surge del hecho de que 384 filtros diferentes se aplican al tensor de entrada en la cuarta capa).

Si esta red se hiciera más profunda, después de otras diez convoluciones, los mapas de activación de salida se reducirían a 5x5. Una convolución más lo reduciría a 1x1. Si porl último, la convolución se realizó con mil filtros diferentes de 5x5, la salida final, después de 31 convoluciones, sería un tensor 1x1x1000 (ver nota al pie relacionada).

Esta posibilidad parecería ofrecer una alternativa a ejecutar varias capas de conexión completas cerca de la salida de la red. Sin embargo, esto también puede disminuir la efectividad de aprovechar la representación distribuida completa en las capas completamente conectadas. Las activaciones en las capas afines finales representan la presencia de características de alto nivel en la imagen. Las combinaciones del primer vector de representación abstracto de alto nivel serán recombinadas por la segunda capa totalmente conectada. Esto permite la recombinación múltiple de características de alto nivel que luego se conectan completamente a la capa de clasificación. La capa de clasificación tiene una cantidad de neuronas igual al número de clases de imágenes. La salida se pasa a través de una función softmax, por lo que cada elemento del vector de clasificación es una probabilidad entre cero y uno.

A medida que disminuye la dimensión en las capas de activación, progresando hacia la salida de la red, podría producirse pérdida de información si el número de filtros por capa no aumenta a un ritmo mínimo. Con una red profunda, es posible mantener el tamaño de la capa rellenando las capas antes de convolucionarlas.

Recuento de parámetros de red			
Capa Convnet	Expresión	Número de parámetros	Memoria necesaria
Capa Conv 1	(5x5x3+1)x96	12.096	0.05MB
Capa Conv 2	(5x5x96+1)x256	614.656	2.46MB
Capa Conv 3	(5x5x256+1)x256	1,638.656	6.55MB
Capa Conv 4	(5x5x384+1)x4096	39,325.696	157MB
Capa Completamente Conectada 1	4096x(4096+1)	16,781.312	67.1MB
Capa Completamente Conectada 2	4096x(4096+1)	16,781.312	67.1MB
Salida	4096x(1000+1)	4,100.096	16.4MB
Total			321MB

Tabla 3: Número de parámetros necesarios para cada capa de red y la memoria requerida para parámetros de precisión individuales. Los requisitos de memoria suponen que cada parámetro es de 4 bytes, o un solo número de punto flotante de precisión.

Sin embargo, cuando se mantiene el tamaño de la capa, se produce un efecto secundario, que es que los píxeles más cercanos al borde tienen un valor inferior o contienen información redundante. Este efecto aumenta en capas de red más profundas. Incluso con relleno, todavía se produce cierta pérdida de información.

Al aumentar el número de filtros a medida que disminuye el área efectiva de las capas de activación, la información de imagen de alto nivel y baja dimensión se transforma en información de bajo nivel y alta dimensión. El desafío central en las redes neuronales es obtener redes que realicen esta transición de modo que se produzca una representación efectiva, de alto nivel, de información comprimida en la salida, por ejemplo, la clase de la imagen.

Las arquitecturas de *convnet* más nuevas están comenzando a reemplazar las capas completamente conectadas en favor de capas más convolucionales. Con las tendencias actuales, se observa un mejor rendimiento cuando la agrupación promedio sobre un tensor de salida de los mapas de activación cuando cada sección transversal es 9x9, o un poco más grande. La agrupación promedio aquí reemplaza las conexiones completas.

Este enfoque se ha utilizado con GoogLeNet, que funcionó mejor que las arquitecturas anteriores al haberse conectado completamente capas cerca de la salida. Esto sugiere que la recombinación de características de alto nivel en sucesivas capas completamente conectadas es excesiva, y que se puede obtener un mejor rendimiento con menos recombinación.

Referencias

[1] Cichy R. M. and Kaiser D. (2019) Deep Neural Networks as Scientific Models. Trends in Cognitive Sciences, Vol. 23, No. 4.

[2] Vincent Dumoulin, *et al*, "A guide to convolution arithmetic for deep learning", 2015, arXiv:1603.07285

[3] Krizhevsky, A. et al. (2012) Imagenet classification with deep convolutional neural networks. Adv. Neural Inform. Process. Syst. 1, 1097-1105

[4] Graves, A. et al. (2013) Speech recognition with deep recurrent neural networks. IEEE International Conference on Acoustics, Speech and Signal Processing 2013, 6645-6649

[5] Mnih, V. et al. (2015) Human-level control through deep reinforcement learning. Nature 518, 529-533
[6] Silver, D. et al. (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529, 484-489
[7] Graves, A. et al. (2016) Hybrid computing using a neural network with dynamic external memory. Nature 538, 471-476
[8] LeCun, Y. et al. (2015) Deep learning. Nature 521, 436-444
[9] Hinton, G.E. (2007) Learning multiple layers of representation. Trends Cogn. Sci. 11, 428-434
[10] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5:115-133, 1943.
[11] David Hubel and Torsten Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology, 160:106-154, 1962.
[12] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition una_ected by shift in position. Biological Cybernetics, 36:193-202, 1980.
[13] Yann Le Cun, Léon Bottou, and Yoshua Bengio. Reading checks with multilayer graph transformer networks. In Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on, volume 1, pages 151-154. IEEE, 1997.
[14] Alex Krizhevsky, Ilya Sutskever, and Geofrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097-1105, 2012.
[15] Christian Szegedy, Wei Liue, Yangqing Jia, Pierre Sermanet Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. arXiv preprint arXiv:1409.4842, 2014
[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.
[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition.IEEE, 2009.
[18] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In Computer vision_ECCV 2014, pages 818-833. Springer, 2014.
[19] Hochreiter, Sepp, and Schmidhuber. Long short-term memory. Neural Computation, 9.8:1735-1780, 1997.
[20] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. CoRR, 2015.
[21] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic

compositionality over a sentiment treebank. Proceedings of the conference on empirical methods in natural language processing (EMNLP), 1631, 2013.
[22] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014
[23] Glorot Xavier and Yoshua Bengio. Understanding the dificulty of training deep feedforward neural networks. Aistats, 9, 2010.
[24] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the ACM International Conference on Multimedia, pages 675_678. ACM, 2014.
[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. arXiv preprint arXiv:1502.01852, 2015.
[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky. Ilya Sutskever Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15:1929-1958, 2014.
[27] X. Glorot, A. Bordes, Y. Bengio (2011). Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, y Miroslav Dudík, editors, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, volume 15 of JMLR W&CP, Fort Lauderdale, FL, USA, 11-13 Apr 2011. PMLR. URL http://proceedings.mlr. press/v15/glorot11a.
[28] A. L. Maas, A. Y. Hannun, y A. Y. Ng. Rectifier nonlinearitiesimprove neural network acoustic models. En ICML Workshop on DeepLearning for Audio, Speech and Language Processing, 2013. URL https://goo.gl/5x3Cj6.
[29] He, K., Zhang, X., Ren, S., y Sun, J. (2016b). Delving deep into rectifiers: Surpassing human-level performance on Imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision. Institute of Electrical and Electronics Engineers Inc. volume 11-18 pp. 1026-1034.
[30] K. Jarrett, K. Kavukcuoglu, Marco Aurelio Ranzato, y Y. Le Cun. What is the best multi-stage architecture for object recognition? En IEEE 12thInternational Conference on Computer Vision, ICCV 2009, Kyoto, Japan, 2009, pp 2146-2153, 2009. ISBN 978-1-4244-4420-5.
[31] Yu-Dong Zhanga, Chichun Panc, Xianqing Chend, Fubin Wange, Abnormal breast identification by nine-layer convolutional neural network with parametric rectified linear unit and rank-based stochastic pooling Journal of Computational Science 27 (2018) 57–68
[32] Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. arXiv preprint arXiv:1302.4389, 2013.
[33] D.A. Clevert, T. Unterhiner and S. Hockreiter, ”Fast and accurate deep network learning by exponential linear units (ELUs)”, ICLR, 2016

[34] Ch. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, y R. García. Incorporating second-order functional knowledge for better option pricing. En Proceedings of the 13th International Conference on Neural Information Processing Systems, NIPS'00,pages 451-457, Cambridge, MA, USA,2000. MIT Press. URL https://goo.gl/doB3GN