Introducción a las redes neuronales: Redes convolucionales

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Computación, bachillerato en Ingeniería en Computación,
PAttern Recongition and MAchine Learning Group (PARMA-Group)

1 de mayo de 2019

Resumen

Este material está basado en el capítulo de redes neuronales, del libro *Pattern recognition and Machine Learning* de Cristopher Bishop[1].

1 Redes profundas o Deep Learning

El aprendizaje profundo provee un marco de trabajo muy utilizado en la industria para el aprendizaje supervisado. Al agregar más capas y más unidades en una capa, la red profunda puede representar funciones de complejidad mayor.

Las redes de alimentación progresiva o perceptrón multicapa (MLP), son también llamadas **redes profundas de alimentación progresiva.** Como se vió anteriormente, estas redes están diseñadas para aproximar una función f^*, que permita asociar una entrada multivariable $\vec{x} \in \mathbb{R}^n$ a una categoría y en el caso de la clasificación, o un valor continuo, en el caso de la regresión:

$$y = f\left(\vec{x}, \vec{\theta}\right)$$

aprendiendo los parámetros $\vec{\theta}$ para encontrar la mejor aproximación a la función f^*, de modo que $f \rightarrow f^*$. Estos modelos se les llama de alimentación progresiva o *feedforward* porque la información fluye desde la evaluación de $\vec{x}$ hasta la definición de los pesos en la capa más externa que define a y. No existen conexiones hacia capas anteriores. Cuando estas conexiones existen, se le llaman **redes recurrentes**, las cuales se estudiarán más adelante.

La **cantidad de capas** en una red MLP define define su **profundidad**. Las capas ocultas son construídas a lo largo del entrenamiento para lograr la mejor aproximación de f^*. Las redes MLP se le llaman redes neuronales por la analogía entre las unidades en una red y las neuronas en el tejido cerebral, y las conexiones entre tales unidades y los enlaces sinápticos entre las neuronas.

Cada neurona recibe múltipes entradas de las neuronas en la capa anterior, y computa su valor de activación, mediante la **función de activación**, constituyendo una función $\mathbb{R}^n \to \mathbb{R}$. La cantidad de neuronas en cada capa se asocia con el **ancho** de la capa.

Una forma de entender las redes MLP es iniciar con los modelos lineales, y considerar como superar sus limitaciones. Algunos modelos lineales como la regresión logística y la regresión lineal, los cuales son muy utilizados pues pueden encajarse en un conjunto de datos usando expresiones cerradas u optimización convexa.

Para extender modelos lineales y representar funciones no lineales de $\vec{x}$, se puede aplicar tal modelo lineal a la entrada transformada:

$$\phi\left(\vec{x}\right)$$

donde ϕ es una transformación no lineal, la cual podemos pensar que provee un conjunto de características de $\vec{x}$, o una nueva representación de tal vector en un nuevo espacio. Las siguientes son opciones para escoger ϕ:

1. Usar una función genérica ϕ, implicitamente usadas en en las máquinas de núcleos (máquinas de soporte vectorial), de infinitas dimensiones, para más bien generar un espacio de más alta dimensión donde se facilite la clasificación.

2. Manualmente diseñar ϕ, enfoque utilizado en el reconocimiento de patrones, lo cual ha involucrado décadas de esfuerzo en áreas especializadas como visión artificial y reconocimiento del habla, con poca transferencia de conocimiento entre dominios, con el objetivo de disminuir la dimensionalidad.

3. La estrategia del aprendizaje profundo es aprender ϕ, por lo que se tiene un modelo que combina linealmente con los pesos $\vec{w}$ las salidas de al función $\phi\left(\vec{x}, \vec{\theta}\right)$, donde ahora $\vec{\theta}$ corresponde a los parámetros de la transformación no lineal ϕ, por lo que entonces:

   $$y = f\left(\vec{x}, \vec{\theta}, \vec{w}\right) = \phi\left(\vec{x}, \vec{\theta}\right)^T \vec{w}$$

   donde en una red MLP, como ya vimos, los parámetros $\vec{\theta}$ corresponden a la capa oculta y deben ser aprendidos. Para capturar los beneficios del primer enfoque, se escogen familias de $\vec{\theta}$ generales. Para combinar con el enfoque 2, se puede escoger la función ϕ con formas conocidas para representar el conocimiento a priori, resultado de la investigación en el dominio específico, de modo que el entrenamiento de la red encuentre los parámetros óptimos $\vec{\theta}$ para tal funcional.

2 Redes convolucionales

2.1 Introducción a la arquitectura de una red convolucional

Material basado en `http://cs231n.github.io/convolutional-networks/`

Sea una imagen de entrada, o matriz $X \in \mathbb{R}^{M \times M}$ y un filtro $F_1 \in \mathbb{R}^{N \times N}$, la convolución es básicamente el computo del producto punto en los puntos de la imagen. Un filtro se puede pensar como un conjunto de pesos que conecta la entrada con la salida, pero conectan solo unidades de entrada locales, con lo cual podemos interpretar un filtro como un arreglo de pesos $F_1 = \vec{w}$. En una red convolucional, a lo largo de su entrenamiento, los coeficientes del filtro se ajustarán en tiempo de entrenamiento.

Por ejemplo tómese el caso en que $M = 64$ y $N = 5$, como se ilustra en la Figura 1.

La convolución resulta en una **imagen filtrada o mapa de activación**:

$$X * F_1 = A_1$$

el cual tiene dimensiones $A_1 \in \mathbb{R}^{V \times V}$ con $V = M - N$. Las redes convolucionales están compuestas en su **capa convolucional** por un conjunto o banco de n filtros $\mathbf{F} = \{F_1, F_2, \ldots, F_n\}$, con lo cual, al **convolucionar cada filtro** con la imagen de entrada X, se forma un conjunto de n mapas de activación $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. En el ejemplo de la Figura 2, se definen $n = 6$ filtros, resultando en igual cantidad de mapas de activación. Tal conjunto de matrices usualmente conforma la entrada de la **capa de activación**, la cual aplica una función de activación, como por ejemplo la función

$$\text{máx}\,(0, x)$$

(correspondiente a una umbralización por cero), conocida como función de activación ReLU. Observe que esta capa no tiene parámetros por ser ajustados.

Posteriormente se define la **capa de agrupación** o *pooling*, la cual realiza una operación de sub-muestreo, resultando en un conjunto de matrices submuestreadas $\mathbf{S} = \{S_1, S_2, \ldots, S_n\}$, con $S_i \in \mathbb{R}^{W \times W}$. El submuestreo puede implementarse con distintas políticas, por ejemplo, una política de **submuestreo máxima** en una ventana *de* 2×2, toma, por cada ventana de 2×2, el valor máximo, como muestra la Figura 2. Otra alternativa es el **submuestreo de promediado**. Ello hace que a la salida de la capa de *agrupación*, las dimensiones de la matriz a la salida sean sustancialmente menores que las dimensiones de la matriz a la entrada, por lo que entonces $W < V$. Esta capa tie2ne la **función incrementar el nivel de abstracción**, tal cual sucede en algoritmos de determinación automática de características como SIFT u ORB, donde se ejecutan análisis multi-resolución de las imágenes.

Finalmente, se define la **capa completamente conectada** la cual está compuesta por K unidades, cada una correspondiente a una de las K clases en las que se clasificarán los datos de entrada, si se utiliza como última capa. Como su nombre lo implica, cada una de las neuronas tendrá conexión con todos los

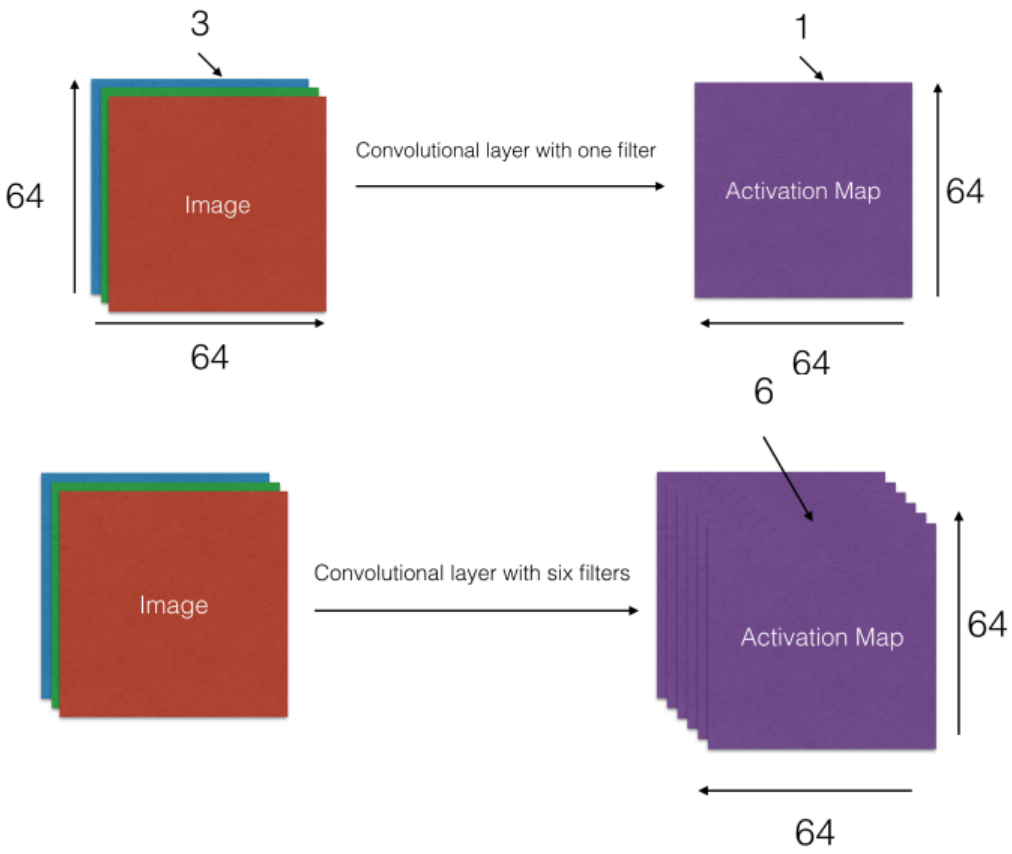


Figura 1: La salida de la capa convolucional es el mapa de activación, tomado de `http://cs231n.github.io/convolutional-networks/`.

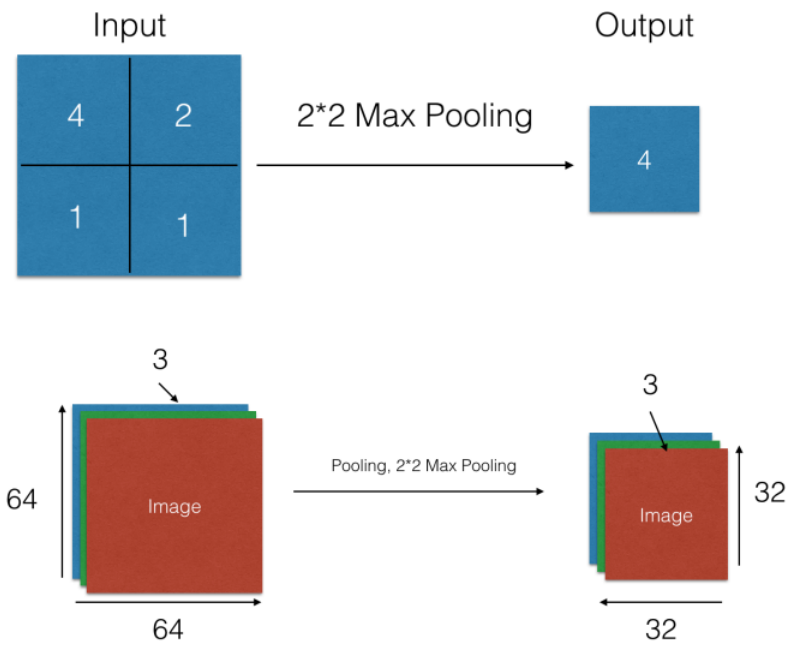


Figura 2: Ilustración del muestreo en la capa de *pooling*, tomado de `http://cs231n.github.io/convolutional-networks/`.

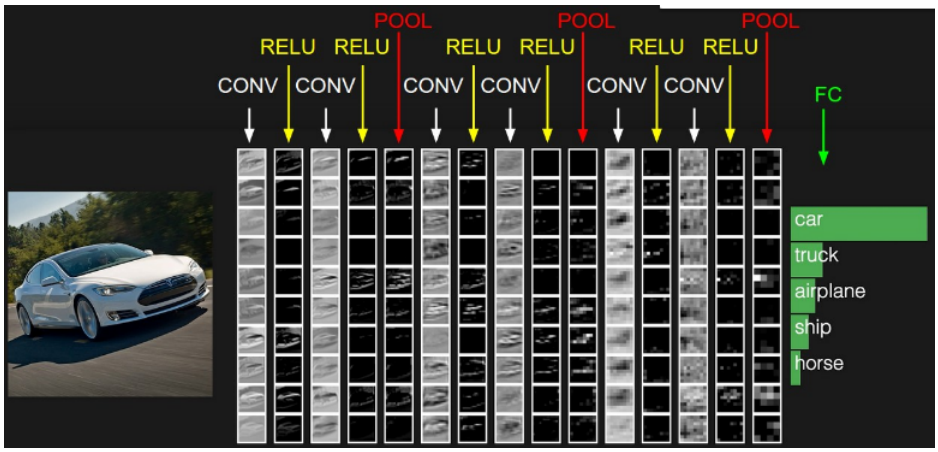


Figura 3: Una red convolucional típica, tomado de `http://cs231n.github.io/convolutional-networks/`.

pixeles resultantes de la *capa de agrupación,* por lo que se comporta como una capa en una red perceptrón multi-capa. Es importante señalar que el método del descenso de gradiente ajustará únicamente parámetros en la capa completamente conectada y en la convolucional.

La capa completamente conectada implementa usualmente como funciones de activación funciones como la sigmoidal y la tangente hiperbólica.

La Figura 3 ilustra una red convolucional completa, en la que se apilan múltiples series de capas convolucional-ReLU-pooling, en la cual se puede observar que los filtros en las primeras capas aprenden características de más bajo nivel.

2.1.1 Las unidades de rectificación lineal y sus generalizaciones

Como se señaló anteriormente, las redes convolucionales utilizan usualmente la función de activación de rectificación lineal, la cual está dada por:

$$g\left(z\right) = \text{máx}\left\{0, z\right\}$$

y gráficamente corresponde a lo diagramado en la Figura 4. Se utiliza el término de rectificación para las Típicamente, su entrada viene dada por $z = \vec{w}^T \vec{x} + b$ para una red neuronal. Como se puede observar en la Figura 4, $z = 0$ existe una discontinuidad, que hace que la función g no sea derivable en tal punto. Esto hace que parezca que la función no pueda usarse para la activación de una unidad, pues la optimización de los pesos $\vec{w}$ de la red se basa en el calculo del gradiente de tal funcional. Para evitar este problema, definimos la derivada en dos tramos, la derivada izquierda de la función $g\left(z\right)$, correspondiente a este caso a cero y la derivada derecha de la función $g\left(z\right)$ correspon-

diente a 1, lo que matemáticamente corresponde a:

$$g'(z) = \begin{cases} 1 & \text{si } z > 0 \\ 0 & \text{si } z < 0 \end{cases}$$

Observe que para $z = 0$, la derivada no está definida. Su indefinición no es de preocupación si tomamos en cuenta que el cálculo de $z = \vec{w}^T\vec{x} + b$ muy dificilmente arribará al resultado $z = 0$, pero en tal caso, se procede a hacer que $z = \epsilon$, con $\epsilon \to 0$.

La ventaja de la función maximo respecto a la función identidad $g(z) = z$ es que mantiene el gradiente con una magnitud alta cuando la unidad está activa, y cuando se desactiva, el gradiente se hace cero. Un gradiente alto cuando la unidad está activa, y un gradiente muy pequeño cuando la unidad está inactiva es deseable, puesto que el gradiente nulo puede ocasionar un problema de **gradiente desvanecido.** Para ello se usa la generalización de la rectificación lineal, con la siguiente forma:

$$g(z) = \text{máx}\{0, z\} + \alpha\,\text{mín}\{0, z\}$$

Notese las siguientes particularizaciones de la formulación general anterior:

- **Rectificación absoluta** $\alpha = -1$: Fijando el α en tal valor, se obtiene $g(z) = |z|$, y es útil por ejemplo en el reconocimiento de objetos, donde la inversión de la polaridad de la iluminación no tiene un efecto sobre las características importantes para discriminar las clases.

- **Rectificación con fuga o *Leaky ReLU*** $0 \leq \alpha \leq 1$: La rectificación con fuga presenta un gradiente pequeño, pero no nulo, cuando la unidad no está activada, disminuyendo las posibilidades de que el cálculo del gradiente se desvanezca en la retro propagación. La Figura 4 muestra una ilustración de tal función de activación.

- **Rectificación parametrizada o PReLU**: En tal variante, el coeficiente α es aprendido según los datos.

2.1.2 Entrenamiento

Como observamos, cada funcional en la red convolucional, es una capa en sí misma. Esto permite, definir, por cada capa, su función gradiente por separado. Si la función en una capa específica, no tiene parámetros por ajustar, entonces no es necesario el cálculo del gradiente. En la red convolucional tratada hasta ahora, las siguientes son capas con pesos a ajustar, y por ende, gradiente a calcular:

- Capa convolucional

- Capa completamente conectada.

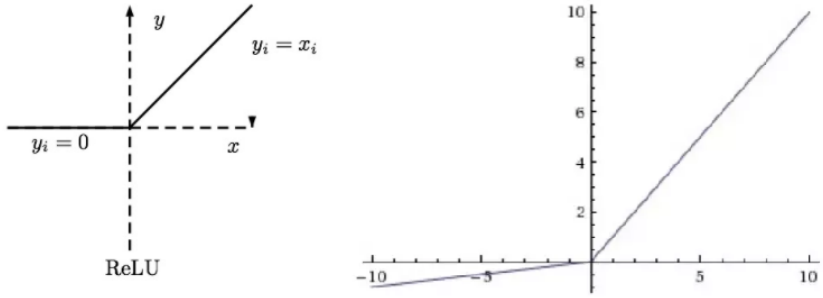


Figura 4: Función de rectificación lineal sin y con fuga.

El resto de capas no tiene parámetros a ajustar, por lo que su función gradiente no se calcula:

- Capa de Reducción o ReLU.
- Capa de agrupación.

2.2 Capa convolucional

Como se mencionó anteriormente, la capa convolucional recibe una matriz de entrada $X \in \mathbb{R}^{M \times M}$ y está compuesta por uan serie de filtros $F_i \in \mathbb{R}^{N \times N}$, cuyos coeficientes son *aprendidos* durante el proceso de entrenamiento. Los filtros tienen usualmente dimensiones pequeñas, del orden de 3×3, 5×5 o 7×7 (usualmente son cuadrados por lo que entonces $F_i \in \mathbb{R}^{U \times U}$,). Recordemos que el proceso de **deslizar el filtro y calcular el producto punto para cada pixel** en la entrada X resulta en el **mapa de activación**. Tal producto punto en cada pixel de la imagen de entrada, puede interpretarse como en fijar una conectividad *local* de las neuronas en la siguiente capa, respecto a la matriz de entrada X, como lo ilustra la imagen de la Figura 5. Intuitivamente, los filtros se entrenarán para encontrar características de bajo nivel en las capas más exteriores, como bordes, manchas de un color específico, etc (según estén definidas las etiquetas).

Observe que el tamaño de la ventana $N \times N$ define la conectividad local de las neuronas en la siguiente capa, de modo que entre más grande, e incluso si llega a ser de las dimensiones de la imagen de entrada X, de modo que $N = M$ y $N = M$, tendremos definida una capa completamente conectada, tal cual sucede en el perceptrón multicapa estudiado anteriormente. Sin embargo, ello se torna impráctico computacionalmente, pero sobre todo, no explota la correlación local existente en las imágenes, donde las características que por ejemplo usamos para distinguir un rostro, tienen una marcada localidad. Al tamaño de la ventanta en los filtros $N \times N$ se le llama el **campo receptivo,** y corresponde a un hiperparámetro de la red convolucional.

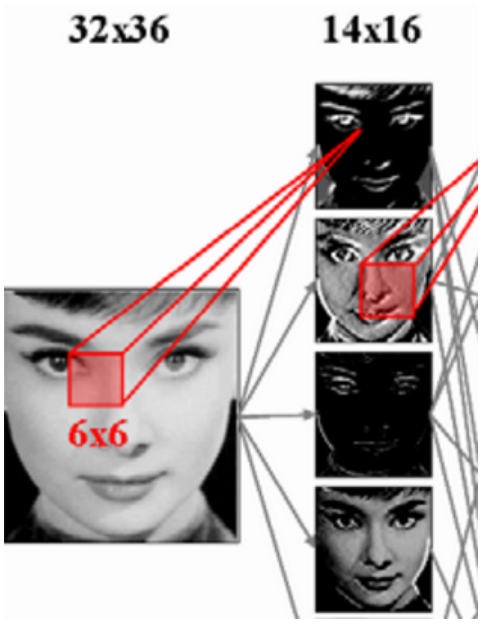


Figura 5: Conectividad local de las neuronas en la primer capa, tomado de `http://cs231n.github.io/convolutional-networks/`.

Lo anterior se ilustra mejor al comparar la conectividad de una red perceptrón multicapa. Recapitulando el concepto de la convolución, la Figura 6 lo resume. Si tomamos la imagen de entrada como los valores en la capa de entrada:

$$\vec{x} = \begin{bmatrix} a & b & c & d & e & f & g & i & j & k \end{bmatrix}$$

de un perceptrón multicapa, el cual en su función de peso neto se realiza con la combinación lineal $s_i = \vec{w}^T \vec{x}$, la cual se desarrolla en la Figura 6. Como se observa en tal desarrollo, cada recuadro en la parte inferior corresponde al valor en la neurona de salida s_i. El vector de pesos en este caso estaría compuesto por los coeficientes del filtro o *kernel*, en este caso dado por*:*

$$\overrightarrow{w} = \begin{bmatrix} w & x & y & z \end{bmatrix}$$

Supóngase un ejemplo más sencillo, en el el cual se convolucionan dos funciones $\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}$ y $\vec{w} = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}$, con lo que $\overrightarrow{w} \in \mathbb{R}^3$. El vector de salidas está dado entonces por:

$$\overrightarrow{s} = \vec{x} * \vec{w} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & s_5 \end{bmatrix},$$

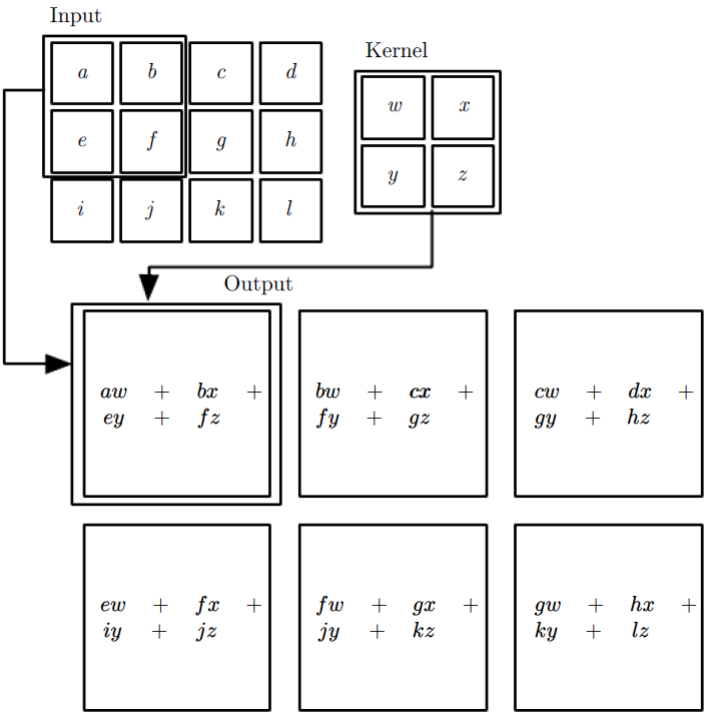


Figura 6: Convolución matricial. Tomado [2].

donde

$$
\begin{aligned}
s_1 &= w_2 x_1 + w_3 x_2 \\
s_2 &= w_1 x_1 + w_2 x_2 + w_3 x_3 \\
s_3 &= w_1 x_2 + w_2 x_3 + w_3 x_4 \\
s_4 &= w_1 x_3 + w_2 x_4 + w_3 x_5 \\
s_5 &= w_1 x_4 + w_2 x_5
\end{aligned}
$$

La Figura 7 ilustra este simple ejemplo usando notación de grafos y la compara con la arquitectura de un perceptrón multicapa, también conocido como una **red completamente conectada**. Para el caso de tal red completamente conectada, cada arista corresponde a un peso w_i distinto, por lo que se tiene un total de $5 \times 5 = 25$ pesos distintos y por ende $\overrightarrow{w} \in \mathbb{R}^{25}$. Tales pesos deben ser definidos en la etapa de entrenamiento. La repetición de los pesos para el caso de una red convolucional ilustrada en el grafo de la Figura 7, y su consecuente menor dimensionalidad, se conoce como **el compartir parámetros** o *parameter sharing*. Tal como se mencionó anteriormente, la analogía con una ventana deslizante a realizarse en una capa convolucional otorga la capacidad de la red de *desacoplar* de su aprendizaje la posición de ciertas características que contribuyen a discernir la clase de una entrada específica, dándole **invariancia a la traslación** de las características.

2.2.1 Propagación hacia atrás en las redes convolucionales

Para entender mejor el proceso de propagación hacia atrás en una red convolucional, extendamos el ejemplo anterior a dos dimensiones, acorde a la Figura 8, donde se hace una convolución $H = X * W$, con $W \in \mathbb{R}^{2\times 2}$ y $X \in \mathbb{R}^{3\times 3}$.

De esta los coeficientes de la imagen filtrada H vienen dados por:

$$
\begin{aligned}
H_{1,1} &= W_{1,1}X_{1,1} + W_{1,2}X_{1,2} + W_{2,1}X_{2,1} + W_{2,2}X_{2,2} \\
H_{1,2} &= W_{1,1}X_{1,2} + W_{1,2}X_{1,3} + W_{2,1}X_{2,2} + W_{2,2}X_{2,3} \\
H_{2,1} &= W_{1,1}X_{2,1} + W_{1,2}X_{2,2} + W_{2,1}X_{3,1} + W_{2,2}X_{3,2} \\
H_{2,2} &= W_{1,1}X_{2,2} + W_{1,2}X_{2,3} + W_{2,1}X_{3,2} + W_{2,2}X_{3,3}
\end{aligned}
\tag{1}
$$

En general, cuándo realizamos una pasada hacia adelante y hacia atrás, sucede lo ilustrado en la Figura 9. Respecto a la convolución desarrollada en las ecuaciones anteriores, al hacer la pasada hacia atrás, se tiene que:

$$
\frac{\partial L}{\partial z} = \frac{\partial L}{\partial H_{i,j}} = \partial H_{i,j}
$$

$$
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial W_{m,n}} = \frac{\partial L}{\partial H_{i,j}} \frac{\partial H_{i,j}}{\partial W_{m,n}} = \partial W_{m,n}
$$

Dado que un coeficiente del filtro W afecta todos los pixeles en H, es necesario sumar su contribución en el gradiente, por lo que siguiendo la regla de la cadena ilustrada en la Figura 9 en cada término de las Ecuaciones 1

$$
\frac{\partial H_{i,j}}{\partial W_{m,n}} = X_{m,n}
$$

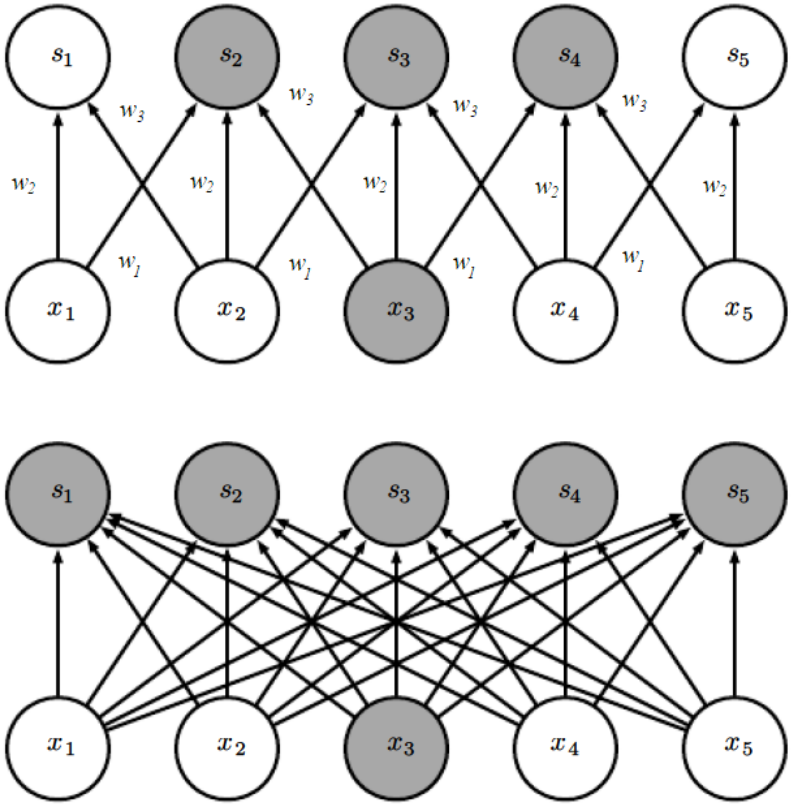


Figura 7: Arquitectura de una red completamente conectada comparada con una red convolucional, con un solo filtro. Tomado [2].

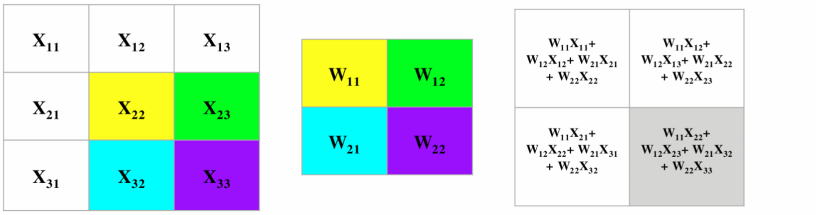


Figura 8: Convolución en dos dimensiones, tomado de `https://bit.ly/2s6lZto`.

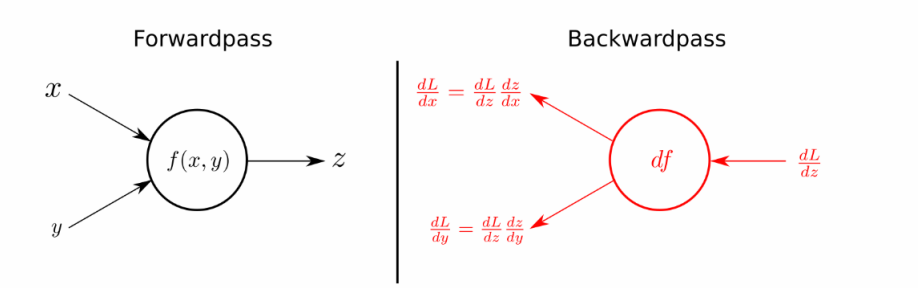


Figura 9: Pasada hacia adelante y hacia atrás en un nodo con dos entradas.

, y sumando cada contribución se tiene que:

$$\begin{aligned}
\partial W_{1,1} &= X_{1,1}\partial H_{1,1} + X_{1,2}\partial H_{1,2} + X_{2,1}\partial H_{2,1} + X_{2,2}\partial H_{2,2} \\
\partial W_{1,2} &= X_{1,2}\partial H_{1,1} + X_{1,3}\partial H_{1,2} + X_{2,2}\partial H_{2,1} + X_{2,3}\partial H_{2,2} \\
\partial W_{2,1} &= X_{2,1}\partial H_{1,1} + X_{2,2}\partial H_{1,2} + X_{3,1}\partial H_{2,1} + X_{3,2}\partial H_{2,2} \\
\partial W_{2,2} &= X_{2,2}\partial H_{1,1} + X_{2,3}\partial H_{1,2} + X_{3,2}\partial H_{2,1} + X_{3,3}\partial H_{2,2}
\end{aligned}$$

con cada término correspondiendo a

$$\frac{\partial L}{\partial H_{i,j}}\frac{\partial H_{i,j}}{\partial W_{m,n}} = \frac{\partial H_{i,j}}{\partial W_{m,n}}\partial H_{i,j}.$$

2.2.2 Otros hiperparámetros en la capa convolucional

1. La cantidad de los n filtros $\mathbf{F} = \{F_1, F_2, \ldots, F_n\}$ a utilizar en la capa convolucional, los cuales definen la ***profundidad*** o cantidad de mapas de activación $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Observe que al finalizar el proceso de convolución, en un elemento (x, y) de cada una de las matrices A_i (correspondiente a la entrada $A_i(x, y)$), se habrá calculado la respuesta de el filtro F_i para la región circundante a tal elemento (x, y) en la matriz de entrada X, lo cual en una notación más compacta corresponde a $\mathbf{A}(x, y)$ y se le denomina **fibra**.

2. El ancho de cada filtro o núcleo (kernel) k. A esto también se le conoce como el espacio receptivo, y define la sensibilidad a características de distintos tamaños.

3. El **paso** p a utilizarse en la convolución, con lo que por ejemplo, si el paso es unitario, el filtro se desliza de pixel en pixel, y si el paso es de 2, el filtro se deslizará cada par de pixeles, y así sucesivamente.

4. La **cantidad de ceros** c añadidos a los bordes de la imagen. El añadir ceros en los bordes de la imagen permite preservar las dimensiones de la entrada $X \in \mathbb{R}^{M_1 \times M_2}$ en los mapas de activación $A_i \in \mathbb{R}^{V_1 \times V_2}$, de modo

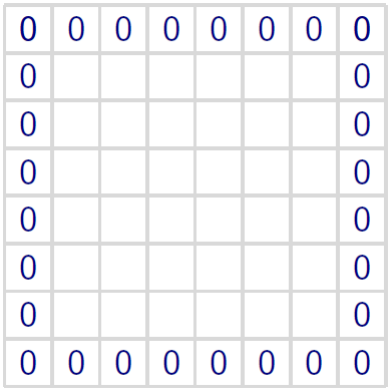


Figura 10: Agregado de ceros en los bordes de la imagen.

que $V_1 = M_1$ y $V_2 = M_2$, si se hace que

$$c = \frac{k-1}{2}.$$

En la Figura 10 se ilustra el caso en el que $k = 3 \Rightarrow c = 1$.

5. **Coeficientes para el aprendizaje**: Estos son heredados de las ecuaciones de actualización de los pesos en un perceptrón multi-capa:

   (a) **Coeficiente de aprendizaje** α, el cual define el *paso* con el cual se varían los pesos en la superficie de error:

   $$W_i(\tau + 1) = W_i(\tau) - \alpha \triangle W_i(\tau),$$

   el cual se define idéntico para toda capa i.

   (b) **El coeficiente de moméntum** μ: en los algoritmos más comunes de entrenamiento estocástico, se incluye la influencia de los pesos anteriores, para mejorar la exploración de regiones locales en la superficie de error, modelando la *inercia* de un cuerpo:

   $$W_i(\tau + 1) = W_i(\tau) - \alpha \triangle W_i(\tau) + \mu \triangle W_i(\tau - 1)$$

   (c) El **tamaño del lote para el entrenamiento** o la cantidad de muestras por mini lote Q.

La definición de los primeros 4 hiperparámetros influye en las dimensiones de los n mapas de activación resultantes al aplicar una capa de convolución con n filtros de dimensiones $k \times k$, con un paso p, m pixeles de ancho en la imagen de entrada, y c filas/columnas de *padding*, cada mapa de activación tendrá de alto y ancho:

$$\frac{m - k + 2c}{p} + 1$$

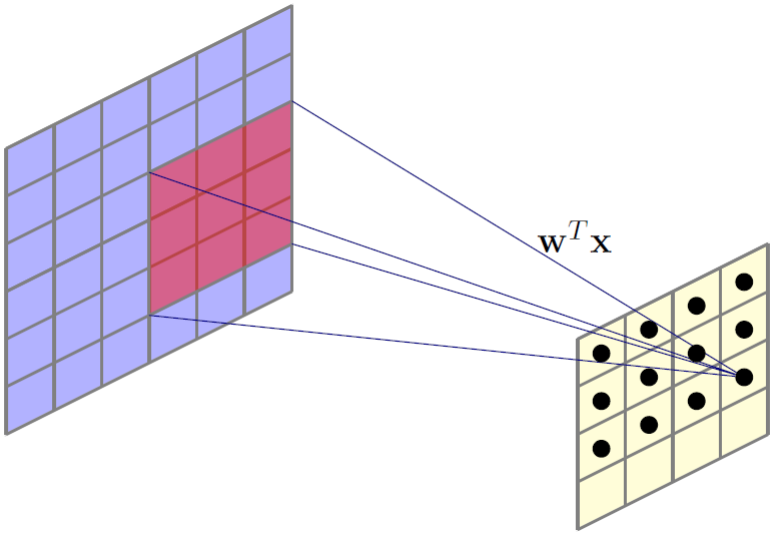


Figura 11: Capa convolucional con $n = 1$ filtros e igual número de mapas de activación resultantes.

Con una entrada de $m \times m$ pixeles. Tómese por ejemplo la aplicación de una capa convolucional a una imagen con $m = 6$ pixeles de alto y ancho, usando un filtro de ancho y alto $k = 3$, y un paso $p = 1$, como se ilustra en la Figura 11. El ancho y alto del mapa de activación viene entonces dado por:

$$\frac{6-3}{1} + 1 = 4$$

como también se ilustra en la Figura 11. Esto en el caso en el que no se concatenen ceros en los extremos de la imagen.

3 Arquitecturas de redes convolucionales avanzadas

3.1 La Arquitectura LeNet 5

Una de las primeras arquitecturas de redes convolucionales profundas, puede encontrarse en el artículo *Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998* donde se definió la red **LeNet 5**, cuya arquitectura se ilustra en la Figura 12.

Las características básicas de la arquitectura propuesta por Y. LeCun et. al. son las siguientes:

- Sub-muestreo de promediado o *average pooling* en ventanas de 2×2.

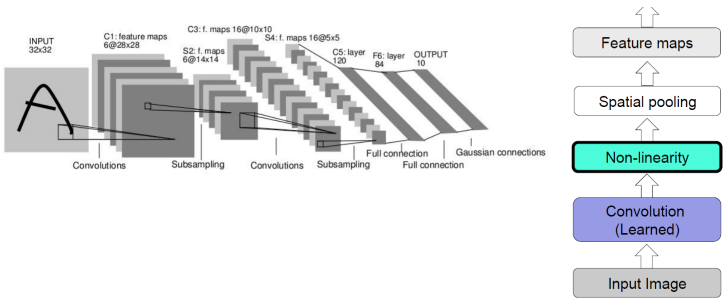


Figura 12: Arquitectura LeNet 5.

- Funciones de activación sigmoidales o tangentes hiperbólicas.
- Capa completamente conectada antes de la salida del modelo.
- Un paso $p = 1$.
- Entrenada y utilizada para la base de datos muestral MNIST de digitos escritos a mano.
- La cantidad de parámetros por capa está dada por:
  - **Capa 1 (C1)**: Compuesta por 6 filtros de 5×5 dimensiones o con tal tamaño de **campo receptivo**, con $(5 \times 5 + 1) \times 6 = 156$ parámetros por aprender. Si hubiese sido completamente conectada, se hubiesen necesitado definir $(32 \times 32 + 1)\,(28 \times 28) \times 6 = 4821600$ parámetros. Al realizarse la convolución, no se efectúa el *zero padding* de la imagen original, por lo que las dimensiones de los mapas de características o *feature maps* se reducen a 28×28.
    - **Capa 2, S2**: Posterior al aplicar la función no lineal de activación (función sigmoidal o hiperbólica), se realiza un sub-muestreo, aplicando el promediado local en una ventana de 2×2, en celdas disjuntas (no ventanas), lo que resulta en mapas de características de 14×14.
  - **Capa 3 (C3):** Genera 16 mapas de activación con dimensiones de 10×10, con conexiones arbitrarias entre los mapas de características de S2 y la capa C3 . Presenta 1516 parámetros entrenables.
    - **Capa 4, S4**: Después de aplicar la función no lineal de activación se realiza un sub-muestreo, aplicando el promediado local en una ventana de 2×2, en celdas disjuntas (no ventanas), lo que resulta en 16 mapas de características de 5×5.
  - **Capa 5 (C5)**: Esta capa realiza un *aplanado* de los datos de forma *pesada* o ponderada, con una convolución usando 120 mapas de activación de 1×1 dimensiones. Esos mapas de activación están conectados a todos los pixeles de los 16 mapas de 5×5 en la capa anterior. De

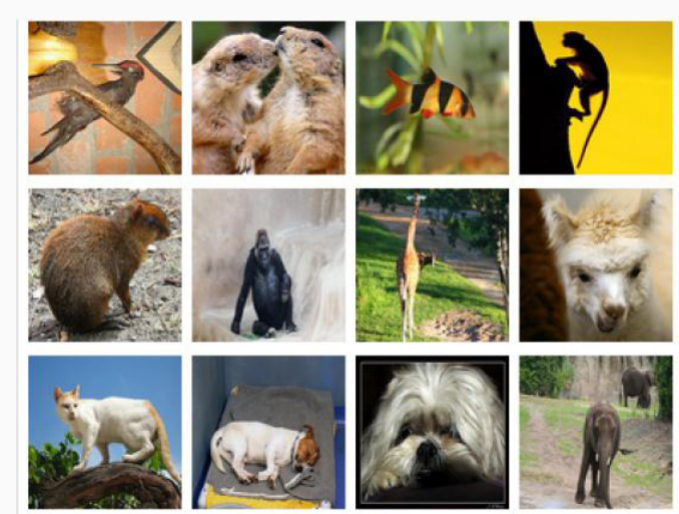


Figura 13: Muestras de ejemplo de ImageNet.

esta forma, en esta capa se deben definir $120 \times (16 \times (5 \times 5) + 1) = 48120$ pesos.

  - **Capa 6 (F6):** Capa completamente conectada, con 84 unidades que conecta las 120 unidades de la capa anterior, lo que hace necesario que se necesiten definir $84 \times (120 + 1) = 10164$ parámetros.

- En total LeNet 5 presenta 60 000 parámetros.

3.2 ImageNet: Competencia de clasificación de imágenes

ImageNet es un repositorio de imágenes con 15 millones de imágenes RGB de resolución variable, 22 mil categorías, recolectadas de la red, y anotadas manualmente con la herramienta de *crowd-sourcing Mechanical Turk*. El repositorio fue creado para la competencia anual ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010 fue la primer edición). En tal competición se utilizan 1000 categorías y 1.2 millones de imágenes de entrenamiento, con 50 000 imágenes de validación y 150 000 imágenes de prueba. Existen dos modos de validación en la competencia:

- El error de cima 1: realizar una estimación de la etiqueta.
- El error de cima 5: Realizar 5 estimaciones de la etiqueta de una imagen.

La Figura 13 presenta algunas muestras de este conjunto de datos.

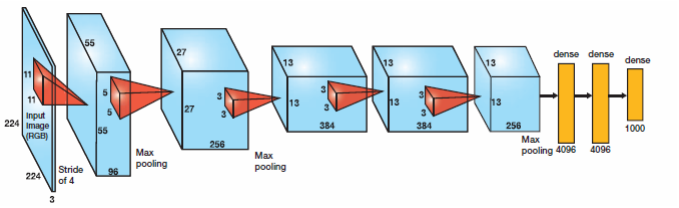


Figura 14: Arquitectura de AlexNet.

3.3 La Arquitectura AlexNet

La arquitectura AlexNet fue propuesta en el artículo *ImageNet Classification with Deep Convolutional Neural Networks, por Alex Krizhevsky et. al.* y se ilustra en la Figura 14, y fue la primer arquitectura basada en CNNs en ganar tal competencia.

AlexNet presenta las siguientes características:

- Introdujo el uso de las funciones de activación ReLU, las cuales agilizaron el proceso de entrenamiento. .
- Utilizó capas de normalización, aunque en otras arquitecturas más recientes no se utiliza.
- Se basó en el uso intensivo de aumentado de datos, con rotaciones de imágenes, achicamientos, etc.
- Entrenada con el descenso de gradiente estocástico.
- Introdujo el uso del *dropout* en los pesos de la red para mejorar su generalización. El *dropout* como se estudió anteriormente, consiste en *apagar* aleatoriamente pesos en la red, para evitar la sobre-especialización de un campo receptivo a una característica.
- Las capas se definieron como:
  - $[227 \times 227 \times 3]$ Entrada: La entrada es una imagen RGB de 227×227.
  - $[55 \times 55 \times 96]$ **Capa 1**: 96 filtros con $k = 11$ de alto y ancho, con un paso $p = 4$, lo que resulta en dimensiones del mapa de activación de $\frac{227-11}{4} + 1 = 55$, resultando en una matriz a la salidad de dimensiones $55 \times 55 \times 96$. La red fue diseñada para procesar imágenes RGB (3 capas), por lo que entonces la **cantidad total de parámetros a entrenar en esta capa** es de $11 \times 11 \times 3 \times 96 = 35000$.
    - $[27 \times 27 \times 96]$ **SubCapa 1**: Compuesta por filtros de máxima agrupación o *max pooling* de 3×3 dimensiones aplicados con un paso $p = 2$, lo que resulta en una dimensionalidad a la salida de $\frac{55-3}{2} + 1 = 27$, $27 \times 27 \times 96$, con 0 parámetros por estimar.

        * $[27 \times 27 \times 96]$ **Subcapa 2**: Normalización sustrayendo la media y dividiendo por la desviación estándar del lote.
    - $[27 \times 27 \times 256]$ **Capa 2**: 256 filtros con $k = 5$ de alto y ancho, con un paso $p = 4$, y *padding* $z = 2$, conservando la dimensionalidad.
        * $[13 \times 13 \times 256]$ **Subcapa 1**: Máxima agrupación con filtros de 3×3 dimensiones aplicados con un paso $p = 2$.
        * $[13 \times 13 \times 256]$ **Subcapa 2**: Normalización sustrayendo la media y dividiendo por la desviación estándar del lote.
    - $[13 \times 13 \times 384]$ **Capa** 3: 384 filtros con $k = 3$ de alto y ancho, con un paso $p = 1$, y *padding* $z = 1$, conservando la dimensionalidad.
    - $[13 \times 13 \times 384]$ **Capa** 4: 384 filtros con $k = 3$ de alto y ancho, con un paso $p = 1$, y *padding* $z = 1$, conservando la dimensionalidad.
    - $[13 \times 13 \times 256]$ **Capa** 5: 256 filtros con $k = 3$ de alto y ancho, con un paso $p = 1$, y *padding* $z = 1$, conservando la dimensionalidad.
        * $[6 \times 6 \times 256]$ **Subcapa 1**: Máxima agrupación con filtros de 3×3 dimensiones aplicados con un paso $p = 2$.
    - $[4096]$ **Capa 6**: Capa completamente conectada de 4096 neuronas.
    - $[4096]$ **Capa** 7: Capa completamente conectada de 4096 neuronas.
    - $[1000]$ **Capa** 8: Capa completamente conectada de 1000 neuronas, correspondientes a los puntajes de las clases.

- En total AlexNet presenta 60 000 000 de parámetros.

3.4 Las Arquitecturas VGG16 y VGG19

La arquitectura VGG fue introducida por Simonyan y Zisserman en su artículo del 2014, *Very Deep Convolutional Networks for Large Scale Image Recognition*, en el contexto del reconocido grupo de investigación de Oxford Visual Geometry Group. El concepto implementado en la arquitectura es el apilado de una serie de convoluciones con pequeños campos receptivos, lo cual disminuye la cantidad de parámetros efectivos a calcular, y potencia la capacidad de abstracción de la red. La Figura 15 muestra una comparativa de las dos variantes con 16 y 19 capas. Por ejemplo, una pila de 3 convoluciones de 3×3 con un paso $p = 1$, tiene un mismo campo receptivo de una capa con un filtro de 7×7 dimensiones, pero más profundo y mayor número de no linealidades. Además, presenta menos parámetros, pues para la primer variante existen un total de $3\left(3^2\right)$ parámetros, mientras que para la segunda 7^2 de parámetros. El modelo completo de VGG16 tiene un total de 138 000 000 parámetros a calcular.

3.5 La arquitectura GoogleNet

La arquitectura GoogleNet propone el uso de una capa con convoluciones aplicadas a la misma entrada, creando un mapa de activación producto de la concatenación de los resultados de tales convoluciones. De esta forma, se implementa un análisis multi-resolución *ingenuo.* Esta capa es referida como *un* ***módulo***

Figura 15: Variantes de VGG 16 y VGG 19, tomado de *Fei-Fei Li & Justin Johnson & Serena Yeung*.

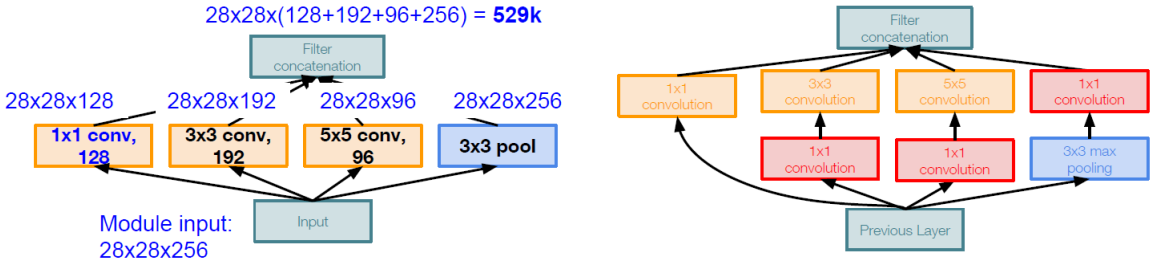


Figura 16: Módulo de Inception ingenuo y con cuello de botella tomado de *Fei-Fei Li & Justin Johnson & Serena Yeung*.

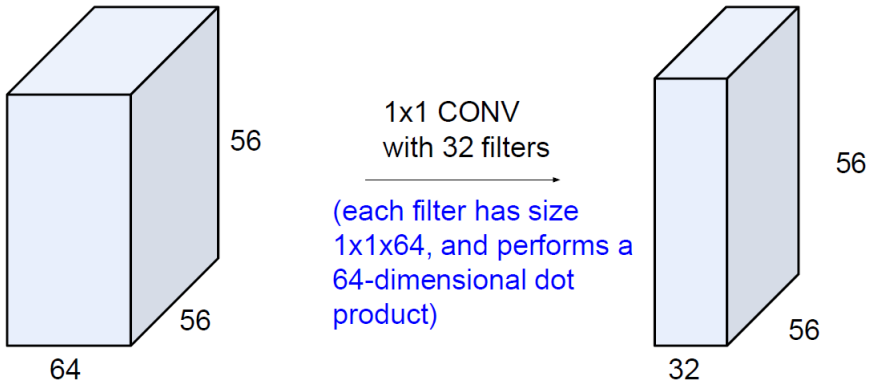


Figura 17: Convolución de 1×1 para reducción de dimensionalidad de la entrada, tomado de *Fei-Fei Li & Justin Johnson & Serena Yeung*.

inception. La implementación directa de este enfoque sin embargo, hace necesario el cálculo de muchísimos parámetros por capa, como lo ilustra la Figura 16. La variante con *cuello de botella*, la cual utiliza un **filtrado con un núcleo de** 1×1 dimensiones, permite reducir la profundidad o cantidad de mapas de activación, como lo ilustra también la Figura 17.

17

En total, la **arquitectura original de GoogleNet (Inception V1) apila 22** capas *inception*, logrando 12 veces menos de parámetros, y venciendo otras arquitecturas en la ILSCRC del 2014.

Inception V2 (*Rethinking the Inception Architecture for Computer Vision*, 2015) mejora el rendimiento apilando convoluciones de menor dimensionalidad, como lo muestra la Figura 18, donde se apilan por ejemplo dos convoluciones de 3×3 para reemplazar una convolución de 5×5. La arquitectura **Inception V3**, propuesta en el mismo artículo, usa el optimizador RMSProp, y suavizacion de

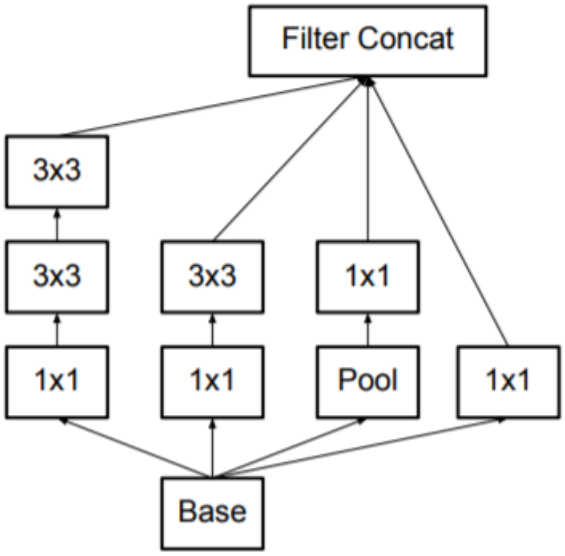


Figura 18: Módulo Inception V2, tomado de *Rethinking the Inception Architecture for Computer Vision*, 2015.

las etiquetas, para regularizar el modelo.

Inception V4 e **Inception ResNet** fueron propuestas en el artículo *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, 2016*, e incorporaron una simplificación de la arquitectura, manteniendo la precisión, y el concepto de conexiones residuales, introducido en la arquitectura ResNet.

3.6 La arquitectura ResNet

La tónica en las recientes arquitecturas ha sido aumentar el número de capas para mejorar el poder de generalización del modelo, junto con técnicas de regularización como *dropout y* normalización de los datos por lote (*batch normalization*). Sin embargo, los autores del articulo *Deep residual learning for image recognition*, notaron que incrementar la cantidad de capas de un modelo luego de un número específico, acarrea una disminución en la precisión del modelo, haciendo los modelos más superficiales mejores en tales circunstancias. Este comportamiento se explica por los fenómenos conocidos como el gradiente **desvaneciente y el gradiente explosivo**, puesto que la propagación del error involucra series de multiplicaciones. Para ello la arquitectura ResNet propone copiar las capas aprendidas, haciendo que la salida sume la entrada original y la transformación $F(x)$, de forma que cada capa tenga una salida $H(x) = F(x) + x$, forzando a que el modelo aprenda un término residual $F(x) = H(x) - x$, como lo ilustra la Figura 19.

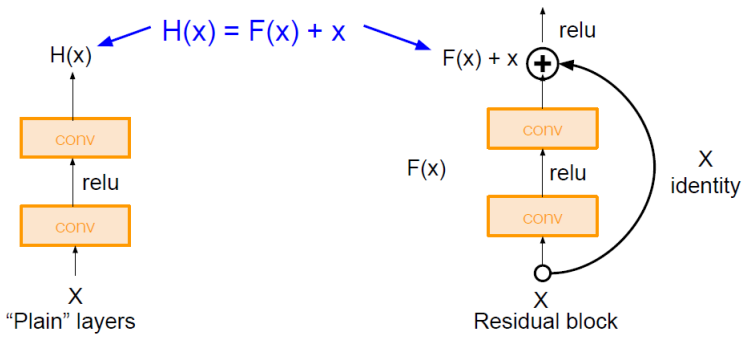


Figura 19: Capa de la arquitectura ResNet.

Además, la arquitectura ResNet propone usar **capas de promediado global** (GAP) en inglés, para promediar los mapas de activación de la última capa, y pasar tales promedios a un modelo completamente conectado de una capa, para disminiuir la necesidad de más parámetros a estimar, conservando la precisión de los resultados según los autores.

Veamos un ejemplo más concreto, ilustrado en la Figura 20.

Tómese una red con dos capas completamente conectadas, que incluyen el sesgo, como la ilustrada en el diagrama de la izquierda, de la Figura 20, y una función de activación $g(x)$ ReLU. La salida X_2 viene entonces dada por:

$$X_2 = g(X_1 W_a) = g(Y_1)$$

y de forma similar

$$X_3 = g(X_2 W_b) = g(Y_2)$$
$$L = (X_3 - T)^2$$

Al calcular entonces el gradiente respecto a W_a, se debe entonces utilizar la regla de la cadena:

$$\frac{\partial L}{\partial W_a} = \frac{\partial L}{\partial X_3}\frac{\partial X_3}{\partial Y_2}\frac{\partial Y_2}{\partial X_2}\frac{\partial X_2}{\partial Y_1}\frac{\partial Y_1}{\partial W_a}$$

Observe que $\frac{\partial Y_2}{\partial X_2} = W_b$, con lo cual, la regla de la cadena implica que al propagar el gradiente, se deben realizar más multiplicaciones por los pesos W_i, con lo cual si los coeficientes en tales matrices $W_i \ll 0$ o si $W_i \gg 1$, se genera lo que se conoce como el **desvanecimiento del gradiente** o la **explosión del gradiente**, respectivamente.

Observemos ahora qué sucede cuando agregamos la conexión de salto, ilustrado a la derecha de la Figura 20.

$$X_1 = g(X_0 W_0) = g(Y_0)$$

$$X_2 = g(X_1 W_a) = g(Y_1)$$

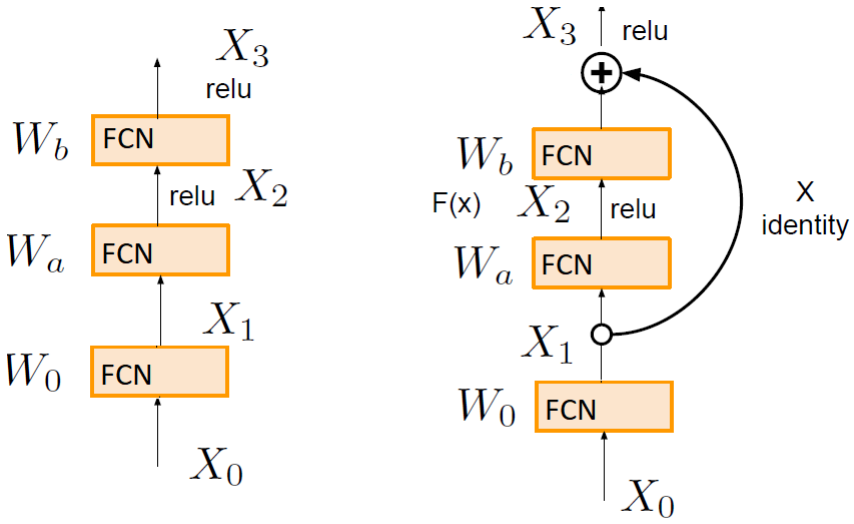


Figura 20: Conexión de salto, Resnet.

y de forma similar

$$X_3 = g\left(X_2 W_b + X_1\right) = g\left(Y_2\right)$$

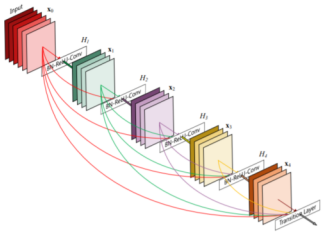
$$L = \left(X_3 - T\right)^2$$

$$\frac{\partial L}{\partial W_0} = \frac{\partial L}{\partial X_3}\frac{\partial X_3}{\partial Y_2}\left(\frac{\partial Y_2}{\partial X_2}\frac{\partial X_2}{\partial Y_1}\frac{\partial Y_1}{\partial X_1}\frac{\partial X_1}{\partial Y_0}\frac{\partial Y_0}{\partial W_0} + \frac{\partial X_1}{\partial Y_0}\frac{\partial Y_0}{\partial W_0}\right)$$

Observe que al adicionar la X_1 en la entrada de la función de activación, el gradiente $\frac{\partial Y_2}{\partial X_2}$ se reemplaza por el término $\left(\frac{\partial Y_2}{\partial X_2} + \frac{\partial X_1}{\partial Y_0}\frac{\partial Y_0}{\partial W_0}\right)$, lo cual permite al gradiente calculado anteriormente $(\frac{\partial L}{\partial X_3}\frac{\partial X_3}{\partial Y_2})$, **fluir más rápidamente**, prescindiendo de multiplicaciones por W_i que pueden hacer el gradiente desvanecerse o explotar. Además, la conexión de salto permite a la red **aprender la función identidad más fácilmente** (equivalente a ignorar la necesidad de crear otros mapas de activacion), y también, en caso de no necesitar capas de unidades neuronales, descartarlas en el aprendizaje, haciendo en el caso de

$$X_3 = g\left(X_2 W_b + X_1\right) = g\left(Y_2\right)$$

$W_b \to 0$, con lo que entonces simplemente $X_3 = g\left(X_1\right)$, lo cual permite en términos generales agregar más capas a al red sin degenerar la precisión e incluso permitiendo mejorarla en caso de ser necesarios más parámetros para aproximar la función.



Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 36
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 24
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

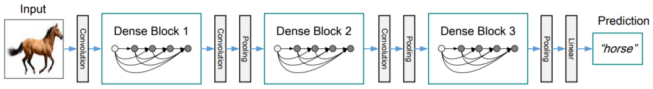


Figura 21: Variantes de DenseNet, tomado de *Densely Connected Convolutional Networks, 2016.*

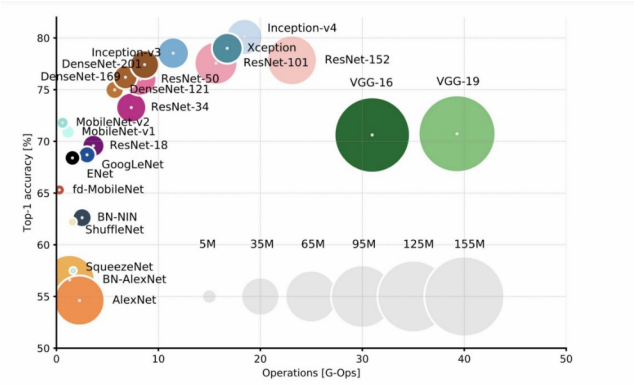


Figura 22: Arquitecturas más populares de aprendizaje profundo.

3.7 La arquitectura DenseNet

Densenet se propuso en el artículo *Densely Connected Convolutional Networks, 2016, inspirada en ResNet*, propone conexiones de salto entre las demás capas, para mejorar el flujo del gradiente. Para **cada capa, los mapas de activación de las capas anteriores son usados como entradas**, usando un coeficiente θ para comprimir tales mapas de activación. Además, como *top model* usa el promediado global y una sola capa completamente conectada, similar a ResNet. La Figura 21 muestra las variantes DenseNet 121, 161, 169 y 201.

La Figura 22 muestra una comparativa de la precisión *top-1 en ImageNet, respecto* a la cantidad de operaciones y la cantidad de parámetros de los modelos.

Figura 23: Generación de mapas de activación por clase, tomado de *Learning Deep Features for Discriminative Localization.*

4 Explicabilidad de los modelos basados en redes profundas

Una debilidad de los modelos basados en redes neuronales es su baja *explicabilidad.* Con ello nos referimos a la dificultad de saber el porqué de una decisión o estimación realizada por un modelo basado en una red neuronal. En el ámbito de uso del modelo, tal explicación puede tener el mismo o incluso más valor que la respuesta en si misma del modelo. Una propuesta para *explicar a nivel de pixel* la realizan los autores del artículo *Learning Deep Features for Discriminative Localization* publicado en la *Computer Vision and Pattern Recognition Conference 2016,* y consiste en crear lo que los autores denominan *mapas de calor* o *mapas de activación por clase.* La Figura 23 muestra el resultado de localizar los pixeles o regiones que más influyen en la discriminación de dos clases específicas (en el caso específico de tal figura, el reconocimiento de acciones), generando una escala de azul a rojo superpuesta en la imagen original.

Para generar tales mapas de activación, los autores proponen cambiar el *top model* de las arquitecturas más conocidas como *GoogleNet,* para clasificar los mapas de características de forma más sencilla en la última capa del modelo $\mathbf{A} = \{A_1, A_2, \ldots, A_N\}$, y facilitar la localización espacial. Tal modificación consiste en eliminar el *top model* e implementar una capa de promediado global, la cual básicamente toma el promedio μ_i de cada mapa de activación $A_i \in \mathbb{R}^{M \times M}$, generando un vector de promedios

$$\overrightarrow{\mu} = \left[\mu_1 = \frac{1}{M^2}\Sigma_{x,y}A_1\left[x,y\right], \mu_2 = \frac{1}{M^2}\Sigma_{x,y}A_2\left[x,y\right], \ldots, \mu_N = \frac{1}{M^2}\Sigma_{x,y}A_N\left[x,y\right]\right].$$

Esos promedios son usados como entrada para una capa completamente conectada, con n unidades a la entrada y K unidades a la salida, con usualmente

una función de activación softmax, por lo que entonces el puntaje p_k para una clase $k$está dado:

$$s_k = \text{softmax}\left(\sum_n^N W_{n,k}\mu_n\right).$$

Los pesos contenidos en la matriz de tal modelo completamente conectado $W \in \mathbb{R}^{N\times K}$ indican entonces la contribución de un mapa de activación n en una clase k. Esto se ilustra en la Figura 24, donde se construye el mapa de activación para la clase *Australian terrier,* la cual podríamos definir como la clase k para construir su *mapa de activación de clase.* Es necesario antes, recordar que los mapas de activación son el resultado de pasar $\mathbf{A} = \{A_1, A_2, \ldots, A_N\}$ banco de filtros $\mathbf{F} = \{F_1, F_2, \ldots, F_N\}$ y una función de activación f, por ejemplo la función ReLU, la cual pone en cero los resultados negativos del filtrado, con lo que :

$$\mathbf{A} = \{A_1 = f(F_1), A_2 = f(F_2), \ldots, A_N = f(F_n)\}$$

Observe que los mapas de activación no son nada más que imágenes filtradas por su filtro correspondiente, por lo que la correlación espacial se conserva, incluso luego de realizar operaciones de agrupación o *pooling* como el *max pooling*, que no hacen más que submuestrear o reducir la dimensionalidad de la imagen, sin alterar tal correlación espacial. De esta forma, para un mapa activación $A_i[x, y]$ podemos indexar por pixel cada mapa.

Retomando la construcción del mapa de activación de la clase k *Australian terrier* $M_k \in \mathbb{R}^{M\times M}$, con las mismas dimensiones del mapa de activación por característica, se utilizan los pesos de la capa completamente conectada, de forma que:

$$M_k[x, y] = W_{1,k}A_1[x, y] + W_{2,k}A_2[x, y] + \ldots + W_{N,k}A_N[x, y].$$

Lo anterior se muestra en la Figura 24.

Es usual que la dimensionalidad de los mapas de activación $A_i \in \mathbb{R}^{M\times M}$ sea notablemente más reducida que la dimensionalidad de la imagen de entrada $U \in \mathbb{R}^{A\times A}$, lo que hace necesario aumentar de tamaño usando técnicas de interpolación de los mapas de activación por clase para superponerlos en la imagen original.

5 Herramientas y ejemplos

Algunos frameworks populares para el desarrollo de redes profundas:

1. Caffe, desarrollado inicialmente por la UC de Berkeley, y Caffe2, extendido por Facebook.
2. Torch, desarrollado inicialmente por la Universidad de Nueva York, y Pytorch, extendido por Facebook.
3. Theano, iniciado por la Universidad de Montreal, y extendido en TensorFlow por Google.

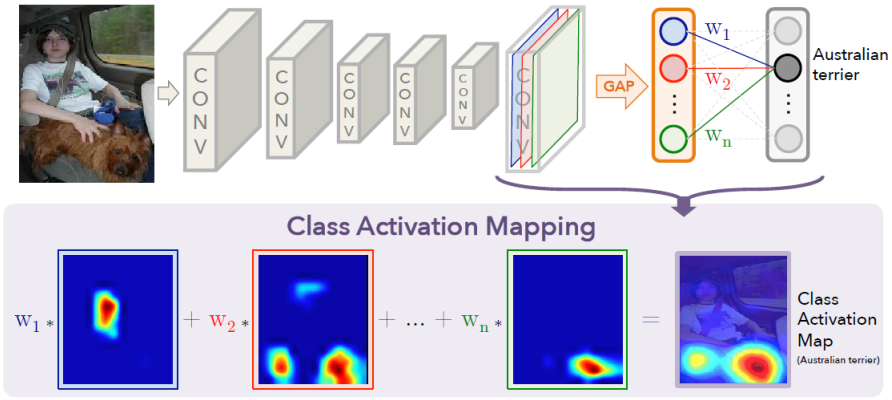


Figura 24: Construcción de mapas de activación.

4. Otros marcos de trabajo: Paddle (Baidu), CNTK (Microsoft), MXNet (Amazon), entre otros.

Las facilidades que usualmente proveen estos marcos de trabajo comprenden la construcción de grafos computacionales, la simplificación para calcular los gradientes en tales grafos computacionales, y su ejecución en plataformas de alto rendimiento.

5.1 Redes convolucionales con Keras

Keras es un API de tensor-flow, CNTK, o Theano, que permite en generar redes neuronales y convolucionales de forma muy sencilla. El siguiente es un ejemplo sencillo de una red diseñada para clasificar imágenes digitales de perros y gatos. La Figura 25 muestra los detalles de la arquitectura.

El siguiente código de Keras implementa la arquitectura propuesta en la Figura 25, y detalla los parámetros de cada función.

```
# Convolutional Neural Network
# Installing Theano # pip install --upgrade --no-deps git+git://github.
    com/Theano/Theano.git
# Installing Tensorflow # pip install tensorflow
# Installing Keras # pip install --upgrade keras
# Part 1 - Building the CNN
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
# Initialising the CNN classifier = Sequential()
# Step 1 - Convolution, input image has 64x64 pixels #32 filters of 3x3
    , by default keras does not do zero padding, changing dimensions
```

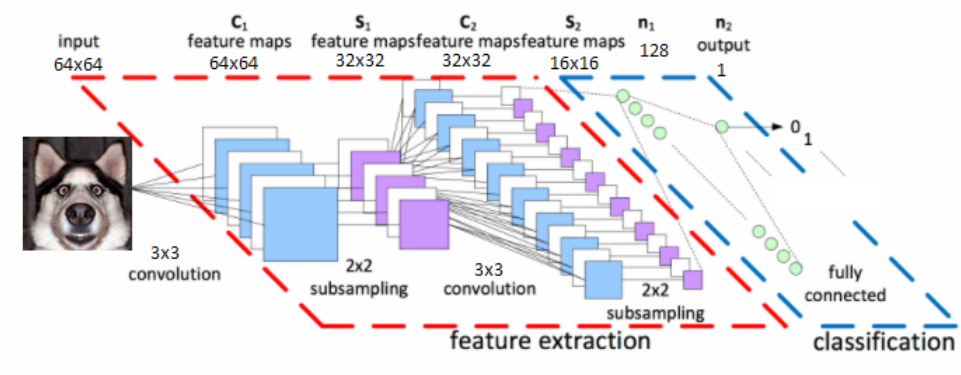


Figura 25: Ilustración de red convolucional simple para la clasificación de imágenes con perros y gatos, en dos clases.

```
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation
    = 'relu'))
# Step 2 - Pooling classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a second convolutional layer
#RELU activation function
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Step 3 - Flattening (vectorization)
classifier.add(Flatten())
# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))
# Compiling the CNN
#loss function: error function to be minimized, usually is the MSE, but
    the crossentropy converges faster
#for weight optimization ADAM is used, a mod. of stochastic gradient
    descent
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
    metrics = ['accuracy'])
# Part 2 - Data augmentation and model training
from keras.preprocessing.image import ImageDataGenerator
# Data augmentation: performs pixel wise normalization (0-1), shear,
    size and flipping
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,  zoom_range =
    0.2, horizontal_flip = True)
#Test images are also normalized
test_datagen = ImageDataGenerator(rescale = 1./255)
#We specify the training data set path, a folder per class
#Batch size: how many samples are randomly taken before doing the back
    propagation (32 samples are used to calculate the error and
    recalculate the weights)
training_set = train_datagen.flow_from_directory('dataset/training_set'
    , target_size = (64, 64), batch_size = 32, class_mode = 'binary')

                                   #We specify the test data set
```

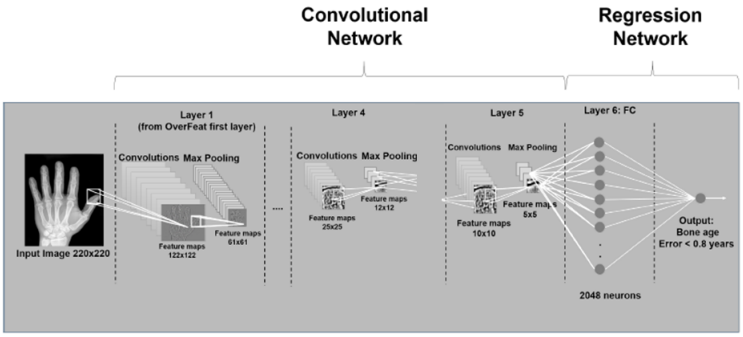


Figura 26: Red BoNET.

Figura 27: Vgg16

```
    path, random samples are taken from this folder
test_set = test_datagen.flow_from_directory('dataset/test_set',
    target_size = (64, 64), batch_size = 32, class_mode = 'binary')
#We train the model
#steps per epoch: number of samples taken per epoch, if the number is
    higher than the samples available, data augmentation is performed
#epochs: one epoch consists in a complete iteration over the whole data
     set
classifier.fit_generator(training_set, steps_per_epoch = 8000,   epochs
    = 25,   validation_data = test_set, validation_steps = 2000)
```

5.2 BoNET simple

Para el siguiente tutorial corto, se implementará una simplificación de la red propuesta en el artículo *Deep learning for automated skeletal bone age assessment in X-ray images* , diseñada para estimar la edad en meses de un sujeto, a partir de la imagen digital de rayos-x de su mano.

La arquitectura BoNET simple implementa los siguientes parámetros:

- **Entrada de la imagen** con 220×220 pixeles.

- **Múltiples capas**: la red implementa 2 capas, donde cada capa está formada por una capa de convolución y otra de *max pooling*. La cantidad de filtros implementada en la capa convolucional
  - La primera capa implementa filtros de tamaño de 7×7 y en la segunda el núcleo es de 5×5 pixeles.
  - La primera capa, la cual funciona como extractor de características de más bajo nivel, debe tener sus pesos congelados, para evitar el problema del gradiente que se desvanece.
  - Por capa, la cantidad de filtros o *mapas* de características es: $96, y\, 1024$.
  - No se cargarán los pesos de otras redes (no habrá *transfer learning*).
  - La capa de max pooling toma el valor máximo cada dos pixeles, reduciendo la dimensionalidad a la mitad.
  - Según lo reportado en artículo, el MAE promedio para los distintos conjuntos es de 1.1 años.
- **Capa completamente conectada:** O también conocida como la *fully connected layer.* Esta red está diseñada para hacer regresión, por lo que a la salida se fija una sola neurona, con una función de activación lineal en su defecto la función identidad. La cantidad de neuronas en la capa oculta es de 2048, según la experiencia de los autores del artículo.
- La métrica para el error es la *Mean Absolute Distance* o *Mean Absolute Error,* correspondiente a $|y - \tilde{y}|$.
- El coeficiente de aprendizaje α está fijado inicialmente con $\alpha = 0{,}002$ con un decaimiento de $0{,}0002$.
- El factor de momentum es $0{,}9$.
- La cantidad de iteraciones o *epochs* es 150.

5.3 BoNET completa

La arquitectura de BoNET implementa los siguientes parámetros:

- **Entrada de la imagen** con 220×220 pixeles.
- **Múltiples capas**: la red implementa 5 capas, donde cada capa está formada por una capa de convolución y otra de *max pooling*. La cantidad de filtros implementada en la capa convolucional
  - La primera capa implementa filtros de tamaño de 7×7, la segunda el núcleo es de 5×5 pixeles y en las siguientes de 3×3 pixeles.
  - La primera capa, la cual funciona como extractor de características de más bajo nivel, debe tener sus pesos congelados, para evitar el problema del gradiente que se desvanece.

  - Por capa, la cantidad de filtros o *mapas* de características es: $96, 2048, 1024, 1024\, y\, 1024$.
  - La primera capa corresponde a la capa de la red *OverFeat*.
  - La capa de max pooling toma el valor máximo cada dos pixeles, reduciendo la dimensionalidad a la mitad.

- **Capa completamente conectada:** O también conocida como la *fully connected layer*. Esta red está diseñada para hacer regresión, por lo que a la salida se fija una sola neurona, con una función de activación lineal en su defecto la función identidad. La cantidad de neuronas en la capa oculta es de 2048, según la experiencia de los autores del artículo.
- La arquitectura *fuera-del-paquete* que mejor funcionó fue GoogLeNet.
- La métrica para el error es la *Mean Absolute Distance* o *Mean Absolute Error,* correspondiente a $|y - \tilde{y}|$.
- El coeficiente de aprendizaje α está fijado inicialmente con $\alpha = 0{,}002$ con un decaimiento de $0{,}0002$.
- El factor de momentum es $0{,}9$.
- La cantidad de iteraciones o *epochs* es 150.

5.3.1 Código

Los siguientes son los paquetes a importar.

```
from keras.models import Sequential, Model, load_model
from keras import applications
from keras import optimizers
from keras.layers import Dropout, Flatten, Dense, Convolution2D,
    MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
```

Generación de semilla y carga de los datos

```
qsub -l nodes=tule-00.cnca vgg16.pbs
```

Para matar un proceso:

```
qdel 16672
```

Referencias

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.