

Redes Convolucionales

Métodos Avanzados de Procesamiento y Síntesis de Imágenes

3er bimestre 2025

Plan de la presentación

Introducción

Filtrado de Imágenes

Redes Neuronales Convolucionales

Principales Arquitecturas de Redes Neuronales Convolucionales

Introducción

Introducción

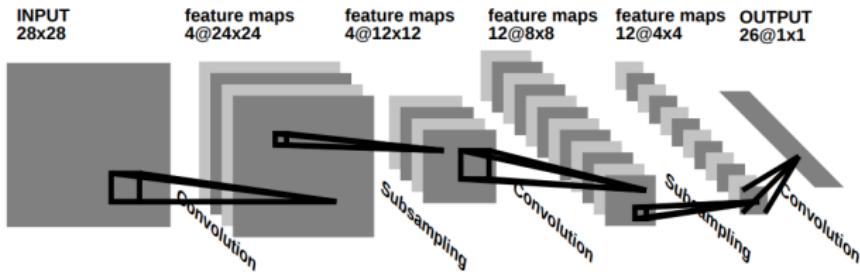
El modelo de Visión por Ordenador se puede dividir en tres niveles:
Procesamiento, Análisis e Interpretación.



Introducción

En 1995 LeCun y Bengio [1] proponen una arquitectura de red neuronal basado en 3 pilares:

1. Campos receptivos locales,
2. Pesos compartidos (o pesos replicados),
3. Subsampling espacial o temporal.

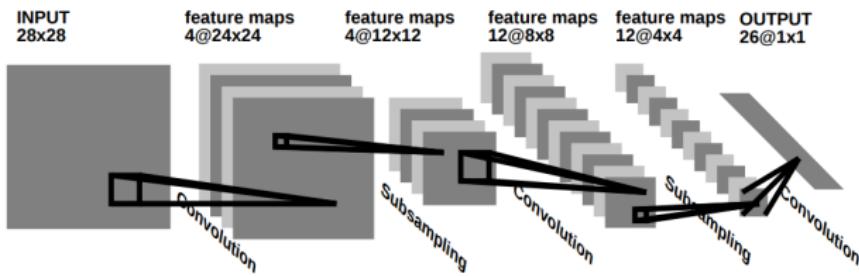


¹Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

Introducción

En 1995 LeCun y Bengio [1] proponen una arquitectura de red neuronal basado en 3 pilares:

1. Campos receptivos locales,
2. Pesos compartidos (o pesos replicados),
3. Subsampling espacial o temporal.



Esta red posee en su primera capa un cálculo de descriptores, que es entrenado por backpropagation.

¹Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.



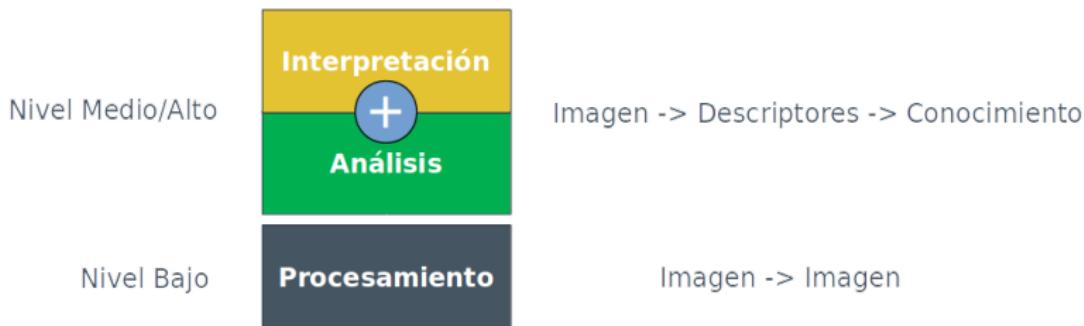
Introducción

Las Redes Neuronales Convolucionadas integran los dos niveles superiores en un mismo proceso.



Introducción

Las Redes Neuronales Convolucionadas integran los dos niveles superiores en un mismo proceso.



Esto significa que la etapa de extracción de descriptores está fuertemente relacionada con el problema a resolver.

Introducción

Las Redes Neuronales Convolucionadas integran los dos niveles superiores en un mismo proceso.



Esto significa que la etapa de extracción de descriptores está fuertemente relacionada con el problema a resolver.

DEPENDE DE LA BASE DE ENTRENAMIENTO!!!

Filtrado de Imágenes

Filtrado de Imágenes

Volvamos a tomar el modelo de Visión por Ordenador dividido en tres niveles: Procesamiento, Análisis e Interpretación.



Filtrado de Imágenes

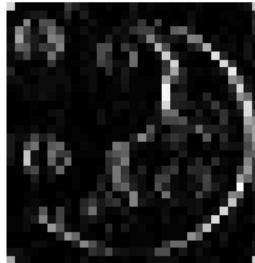
Segmentación

La segmentación convierte a la información de vecindad de los pixeles una representación matemática.



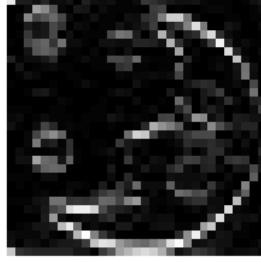
$$\begin{matrix} -1 & 1 \\ \rightarrow & \end{matrix}$$

Dx



$$\begin{matrix} 1 \\ -1 \end{matrix}$$

Dy



Filtrado de Imágenes

Máscaras

-1
1

-1	1
----	---

-1	0
0	1

0	-1
1	0

Roberts

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Prewitt

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Sobel

Filtrado de Imágenes

Convolución

Para realizar el filtrado, usamos la operación de CONVOLUCIÓN de tipo matricial.

La Convolución entre una matriz de entrada \mathbf{I} y un filtro \mathbf{w} se define como:

$$\begin{aligned}\mathbf{I}' &= \mathbf{w} * \mathbf{I} \\ I'[u, v] &= \sum_{q=0}^a \sum_{p=0}^b w[q, p] I[u - q, v - p]\end{aligned}$$

Convolución

Para realizar el filtrado, usamos la operación de CONVOLUCIÓN de tipo matricial.

La Convolución entre una matriz de entrada \mathbf{I} y un filtro \mathbf{w} se define como:

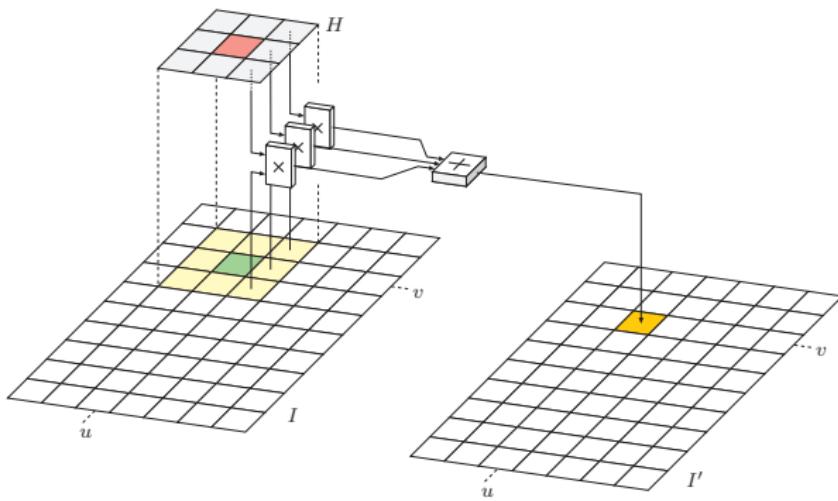
$$\begin{aligned}\mathbf{I}' &= \mathbf{w} * \mathbf{I} \\ I'[u, v] &= \sum_{q=0}^a \sum_{p=0}^b w[q, p] I[u - q, v - p]\end{aligned}$$

y teniendo mucho cuidado tal que p y q tengan valores para que los índices de $w[q, p]$ y $I[u - q, v - p]$ sean válidos.

Filtrado de Imágenes

Convolución

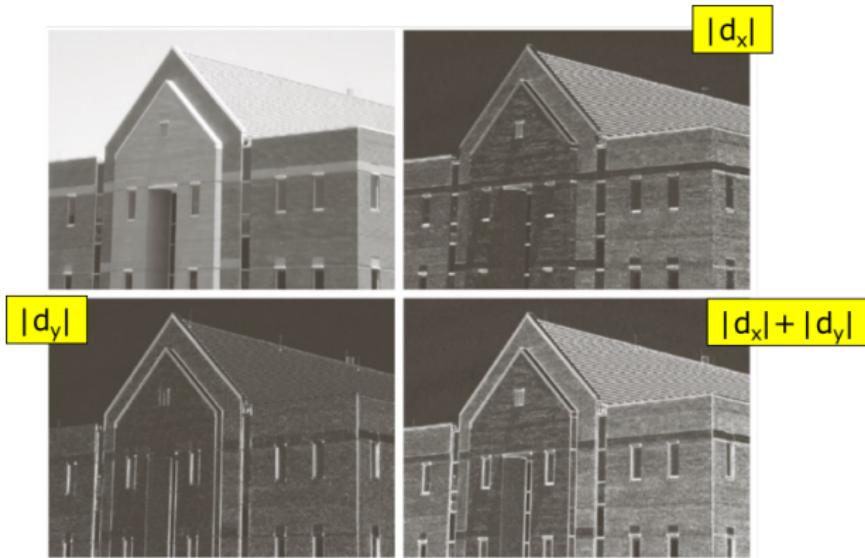
La convolución es una operación local, que aplica el mismo cálculo de relación entre píxeles vecinos a todo el plano de la imagen. El resultado individual de cada operación genera un plano de convolución resultado.



Filtrado de Imágenes

Convolución

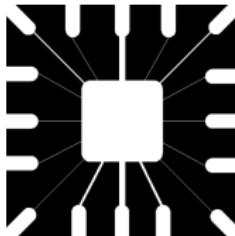
El resultado del filtrado va a depender de los valores de la matriz w .



Filtrado de Imágenes

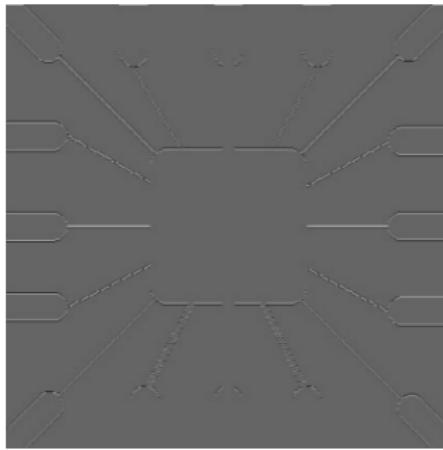
Convolución

El resultado del filtrado va a depender de los valores de la matriz w.



-1	-1	-1
2	2	2
-1	-1	-1

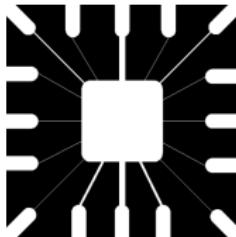
Horizontal



Filtrado de Imágenes

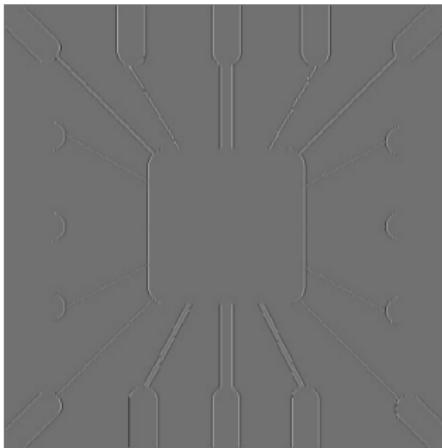
Convolución

El resultado del filtrado va a depender de los valores de la matriz w.



-1	2	-1
-1	2	-1
-1	2	-1

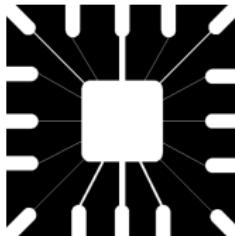
Vertical



Filtrado de Imágenes

Convolución

El resultado del filtrado va a depender de los valores de la matriz w .



2	-1	-1
-1	2	-1
-1	-1	2

+45°



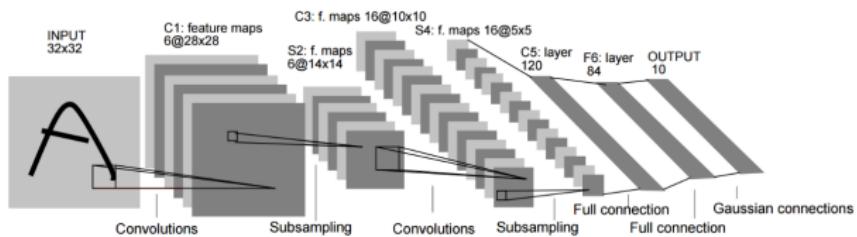
Convolución

Se conoce entonces de antemano la salida esperada del filtrado.
Los $w(p, q)$ de los filtros se eligieron expresamente!!!

Redes Neuronales Convolucionales

Introducción

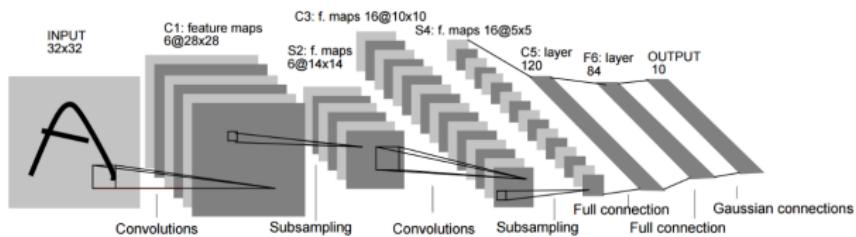
La propuesta de LeCun y Bengio [2] es que estos pesos sean *entrenados* en una arquitectura de red neuronal:



²Y. LeCun, P. Haffner, L. Bottou, Y. Bengio. Object recognition with gradient-based learning. In Shape, Contour and Grouping in Computer Vision, pp. 319-345, 1999.

Introducción

La propuesta de LeCun y Bengio [2] es que estos pesos sean *entrenados* en una arquitectura de red neuronal:



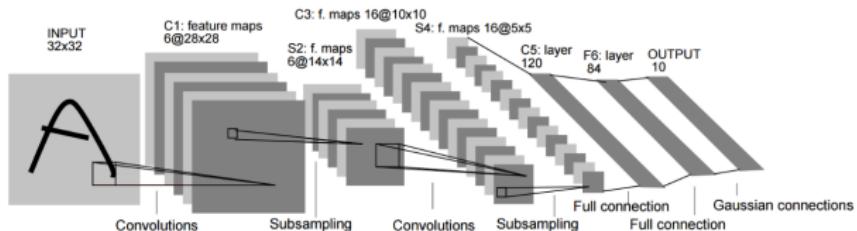
Entonces tenemos una arquitectura con:

1. Campos receptivos locales,
2. Pesos compartidos (o pesos replicados),
3. Subsampling espacial o temporal,
4. Capas de salida Densas.

²Y. LeCun, P. Haffner, L. Bottou, Y. Bengio. Object recognition with gradient-based learning. In Shape, Contour and Grouping in Computer Vision, pp. 319-345, 1999.

Arquitectura de la Red

Retomamos la arquitectura LeNet5, de Y. Lecun.

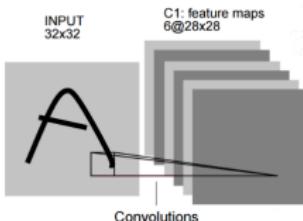


El detalle es el siguiente:

Tipo	Entrada	CONV	POOL	CONV	POOL	CONV	FC	OUTPUT
Nombre		C1	S2	C3	S4	C5	F6	
Dimension	32x32	6x28x28	6x14x14	16x10x10	16x5x5	120	84	10

La entrada consiste en una matriz de un solo plano de 32x32 pixels. Esto significa que es una imagen en niveles de gris. La salida es un vector de 10 unidades, donde cada una da la probabilidad de la pertenencia de la imagen de entrada a una de las 10 clases (en el MNIST son dígitos manuscritos).

Arquitectura de la Red : CONV:C1



Esta capa se obtiene de la convolución de 6 filtros de 5x5 pixeles. La función de activación es:

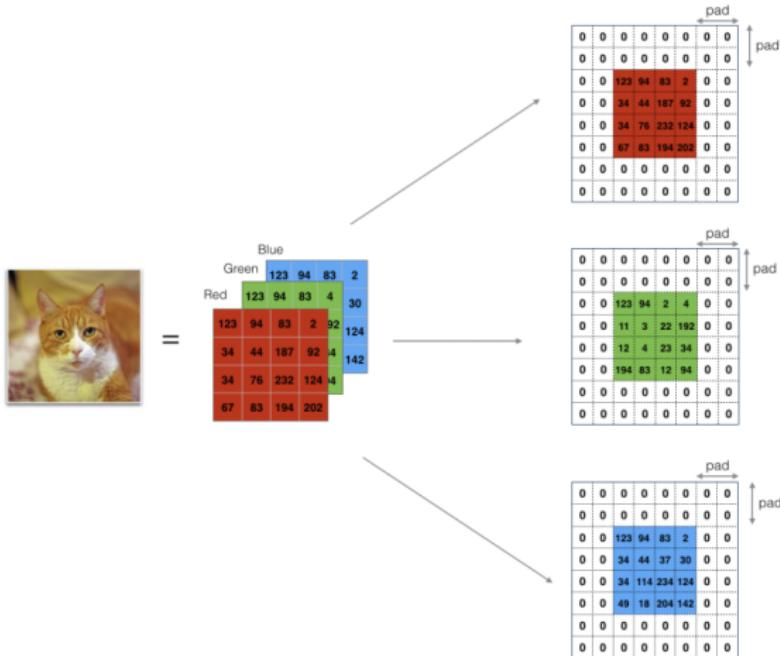
$$A_{1i} = \mathbf{w}_{1i} * \mathbf{I} + b_{1i}, \quad i = 1, \dots, 6$$

\mathbf{w}_{1i} es el i -ésimo filtro, y b_{1i} es el bias. El tamaño de A_{1i} es de $28 = 32 - 5 + 1$. Los parámetros a ser entrenados en cada capa son 25, más el bias. En total son 156 parámetros en esta capa ($25 * 6 + 6$).

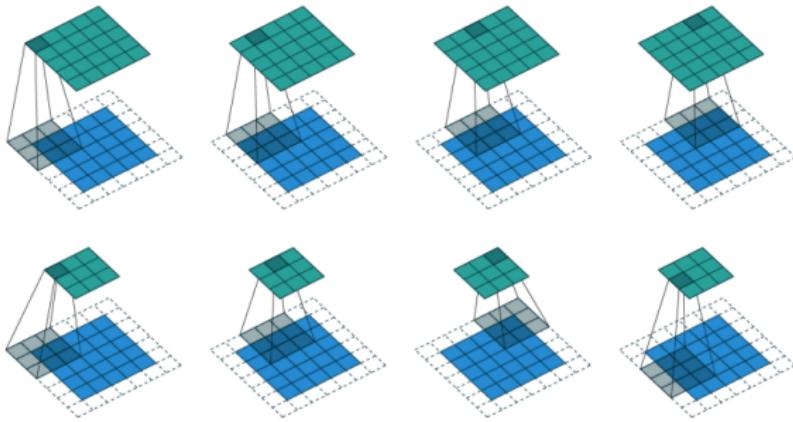
Para el cálculo de la convolución se incorporan dos conceptos:

- ▶ *padding*: es la cantidad de pixels con valor 0 que se agregan por fuera de la imagen de entrada ($p = 0$)
- ▶ *stride*: es el paso de convolución, o sea, la cantidad de pixeles que se desplaza el filtro entre una convolución y la siguiente ($s = 1$).

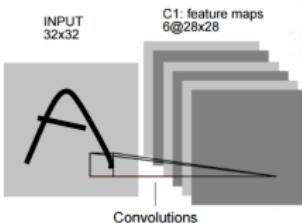
Arquitectura de la Red : CONV:C1



Arquitectura de la Red : CONV:C1



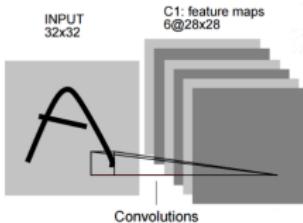
Arquitectura de la Red : CONV:C1



Luego de la convolución se aplica la función *sigmoide*:

$$C_{1i} = \frac{1}{1 + e^{-A_{1i}}}, i = 1, \dots, 6$$

Arquitectura de la Red : CONV:C1



Tips de implementación.

Dado que la definición matemática de la convolución involucra un flipping de la matriz de entrada, invirtiendo sus líneas y columnas, al utilizar la convolución, es preciso hacer un flip de la máscara o filtro W antes de la convolución. Si W es una máscara de 3x3 pixels:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \xrightarrow{\text{flip}} \quad \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix}$$

Esto se puede realizar con

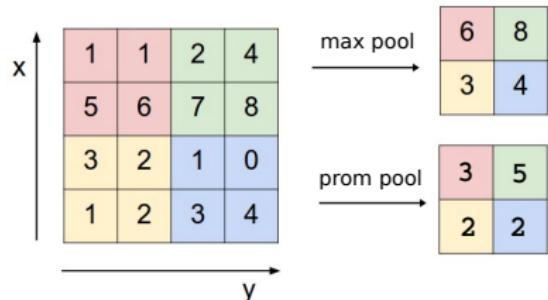
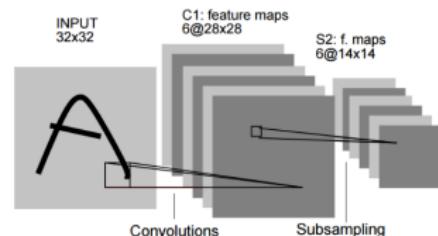
```
> W = rot90(W,2);
```

Arquitectura de la Red : POOL:S2

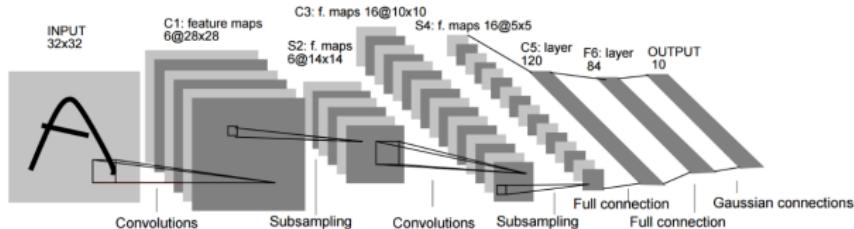
La salida de C1 se propaga a la capa de subsampleo o pooling S2.
El tamaño de pooling es de 2x2 pixeles
sin overlapping ($stride = 2$),
reduciendo las matrices de C1 a la
mitad. De esta manera, la capa S2
termina con un tamaño de $6 \times 14 \times 14$.
Hay dos tipos de pooling: máximo y
promedio. En la gráfica vemos el
resultado de aplicarlos a una matriz de
 4×4 usando un $poolDim = 2$ sin
overlapping.

El pooling promedio puede resolverse
eficientemente con la función
convolve^a.

^a<https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve.html>

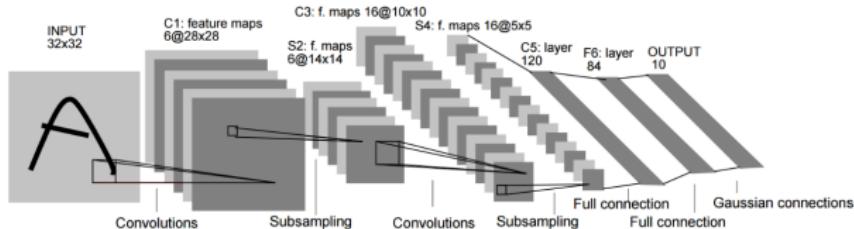


Arquitectura de la Red



- ▶ Las capas C3 y S4 son una capa convolucional seguida de un pooling.
- ▶ Las capas FC:F5 están completamente conectadas entre las salidas de S4 (400 neuronas) y las neuronas escondidas de F5 (120). Las funciones de activación son también *sigmoide*.
- ▶ Entre F5 y F6 es también una conexión completa entre las neuronas.
- ▶ Finalmente, el valor de cada neurona de salida mide la probabilidad de que la imagen de entrada pertenezca a esta clase.

Arquitectura de la Red



- ▶ Esto se calcula mediante la función softmax.

$$P(y^{(i)} = k | \mathbf{z}^{(i)}; W^{F6}, b^{F6}) = \frac{e^{W_k^{F6} \mathbf{z} + b_k^{F6}}}{\sum_j^K e^{W_j^{F6} \mathbf{z} + b_j^{F6}}} \quad (1)$$

donde W_i^{F6} son los pesos conectando la capa F6 con la neurona de salida i , y b_i es el bias correspondiente, \mathbf{z} es la salida de la capa FC precedente, y $K = 10$.

Aprendizaje: Backpropagation

Llamamos ahora:

- ▶ δ^{l+1} es el término de sensibilidad (o error) de la capa $(l + 1)$
- ▶ la red posee una función de costo $J(W, b; x, y)$.

Si la capa l es de tipo FC, y se conecta con $(l + 1)$, la sensibilidad o error se propaga como: $\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \bullet f'(z^{(l)})$.

$$\begin{aligned}\nabla_{W^{(l)}} J(W, b; x, y) &= \delta^{(l+1)} (a^{(l)})^T, \\ \nabla_{b^{(l)}} J(W, b; x, y) &= \delta^{(l+1)}\end{aligned}$$

donde $a^{(l)}$ es la entrada de la capa l .

Aprendizaje: Backpropagation

En el caso que la capa l sea convolucional/pooling, el error se propaga como

$$\delta_k^{(l)} = \text{upsample} \left((W_k^{(l)})^T \delta_k^{(l+1)} \right) \bullet f'(z_k^{(l)}) \quad (2)$$

k es el número de filtro convolucional de esta capa, y $f'(z_k^{(l)})$ es la derivada de la función de activación. La operación `upsample` propaga el error a través de la capa pooling hacia la capa convolucional. El proceso inverso al pooling significa volver al tamaño original (que espera para hacer la convolución). Para el caso del pooling promedio, se puede distribuir el valor del error en todo el parche del pooling.

Los gradientes se calculan como:

$$\begin{aligned}\nabla_{W_k^{(l)}} J(W, b; x, y) &= \sum_{i=1}^m (a_i^{(l)}) * \text{rot90}(\delta_k^{(l+1)}, 2), \\ \nabla_{b_k^{(l)}} J(W, b; x, y) &= \sum_{a,b} (\delta_k^{(l+1)})_{a,b}.\end{aligned}$$

Aprendizaje: Backpropagation

Tips para realizar el upsampling.

La función kron puede ser utilizada para el upsampling del error proveniente de la capa siguiente, para poder propagarlo en la capa de pooling de tipo promedio. Esta función calcula el producto de tipo delta Kronecker entre dos matrices. Suponiendo se recibe una matriz delta de 2x2, y se debe hacer el upsampling a una de 4x4, la operación es algo como:

$$\text{kron} \left(\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \right) \xrightarrow{\text{flip}} \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{pmatrix}$$

Luego de que el error ha sido repartido en la nueva matriz, lo que queda dividirlo por el tamaño de la región de pooling y propagarlo a la capa de convolución. La implementación sería:

Esto se puede realizar con

```
> deltaPool = (1/(poolDim*poolDim)) *  
np.kron(delta,ones(poolDim));
```

Aprendizaje: Backpropagation

La función de salida se calcula mediante la ecuación 1 que implementa el softmax. Agrupamos los parámetros $\theta = (W, b)$ de manera de simplificar la notación.

La función de costo asociada::

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K \mathbf{1}\{y^{(i)} = k\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)\top} x^{(i)})} \right] \quad (3)$$

donde m es la cantidad de imágenes en el set. $\mathbf{1}\{\cdot\}$ es la función que devuelve 1 si el argumento es verdadero o 0 en caso contrario.

Calculando derivadas, se obtiene:

$$\begin{aligned} \nabla_{\theta^{(k)}} J(\theta) &= -\sum_{i=1}^m \left[x^{(i)} \left(\mathbf{1}\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right] \\ &= -\sum_{i=1}^m \left[x^{(i)} E^{(i)}(\theta; x^{(i)}, y^{(i)}) \right] \end{aligned} \quad (4)$$

Aprendizaje: Stochastic Gradient Descend

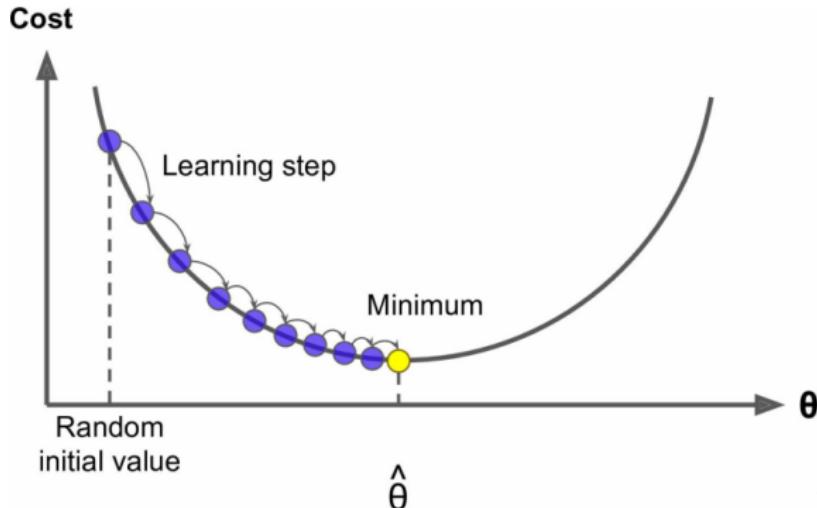
El SGD es un método de entrenamiento basado en el descenso del gradiente a partir de un set limitado de ejemplos de entrenamiento. De esta manera posee dos ventajas frente a las metodologías de batch training: permite trabajar con bases de entrenamiento muy grandes, y, al mismo tiempo, acelera la convergencia.

Dado que el update de los parámetros de la red se realiza sobre un set pequeño (o incluso un solo ejemplo), podemos escribirlo como:

$$\theta := \theta - \eta \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)})$$

η , el ratio de aprendizaje, toma valores pequeños y en general disminuye a lo largo del entrenamiento, a cada epoch.

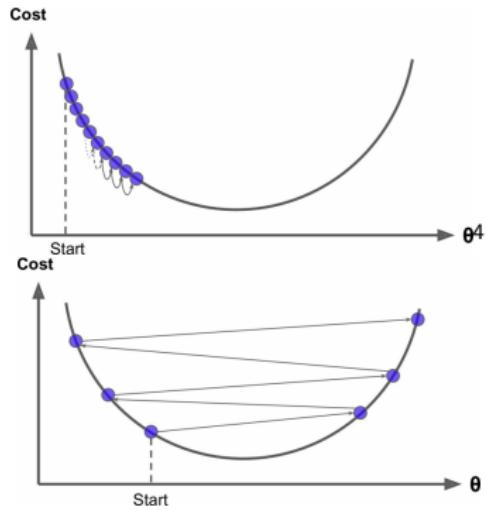
Aprendizaje: Stochastic Gradient Descend



3

³Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, 2019.

Aprendizaje: Stochastic Gradient Descend



⁴Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, 2019.

Aprendizaje: Stochastic Gradient Descend

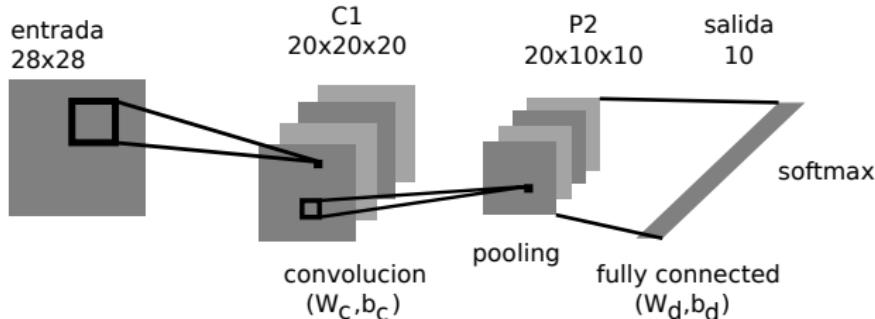
En las cercanías del mínimo, métodos como el SGD tienen a oscilar, retrasando la convergencia. Una metodología para acelerar el proceso es el llamado *momentum*. El update se define ahora como

$$\begin{aligned}v &:= \gamma v + \eta \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)}) \\ \theta &:= \theta - v\end{aligned}$$

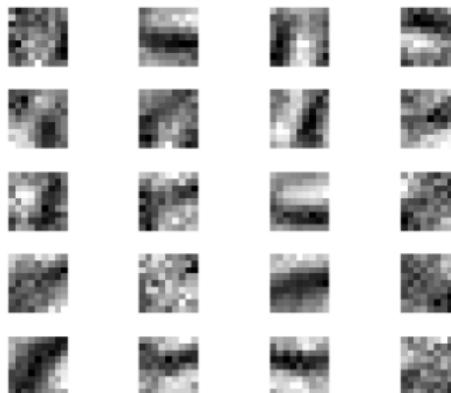
donde v es la velocidad, del mismo largo que el θ , y γ determina la cantidad de iteraciones del gradiente son incorporadas al update.

LABORATORIO

Vamos a implementar un aprendizaje de una red CNN como muestra la figura



Los datos de entrada de la red son imágenes de dígitos (LPR) en niveles de gris con un tamaño 28×28 pixeles.



Principales Arquitecturas de Redes Neuronales Convolucionales

Arquitecturas de redes convolucionales

Resumen de principales arquitecturas

- ▶ LeNet (1989-1999)
- ▶ AlexNet (2012)
- ▶ VGGNet (2014)
- ▶ GoogLeNet (Inception, 2014)
- ▶ ResNet (2015)
- ▶ DenseNet (2016)
- ▶ MobileNet (2017)
- ▶ ShuffleNet (2018)
- ▶ EfficientNet (2019)

Arquitecturas de redes convolucionales

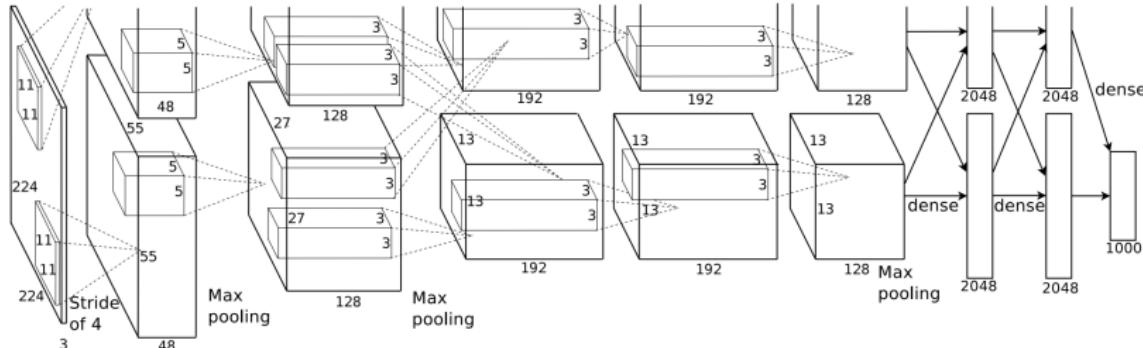
AlexNet

AlexNet: características e innovaciones

- ▶ Paper: Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems (NeurIPS), 2012.
- ▶ Ganadora de la competencia ImageNet 2012.
 - ▶ 15M de imágenes etiquetadas de la web en alta resolución.
 - ▶ 22K categorías.
 - ▶ Challenge: clasificar 1K categorías de un test set de 150K
 - ▶ Top 5 error: hasta 2011:28% - 2012: AlexNet 16%
- ▶ Se entrena en dos GPUs permitiendo un modelo de más parámetros.
- ▶ Dropout: Técnica de regularización que evita overfitting. Se desactivan aleatoriamente un porcentaje de neuronas durante el entrenamiento.
"Dropout consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are 'dropped out' in this way do not contribute to the forward pass and do not participate in backpropagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights."
- ▶ ReLu (Rectified Linear Units) como función de activación.
- ▶ Data Augmentation

Arquitecturas de redes convolucionales

AlexNet:



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL 1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 266 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3 : 384 3x3 filters at stride 1, pad 1

[13x13x334] CONV4: 334 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

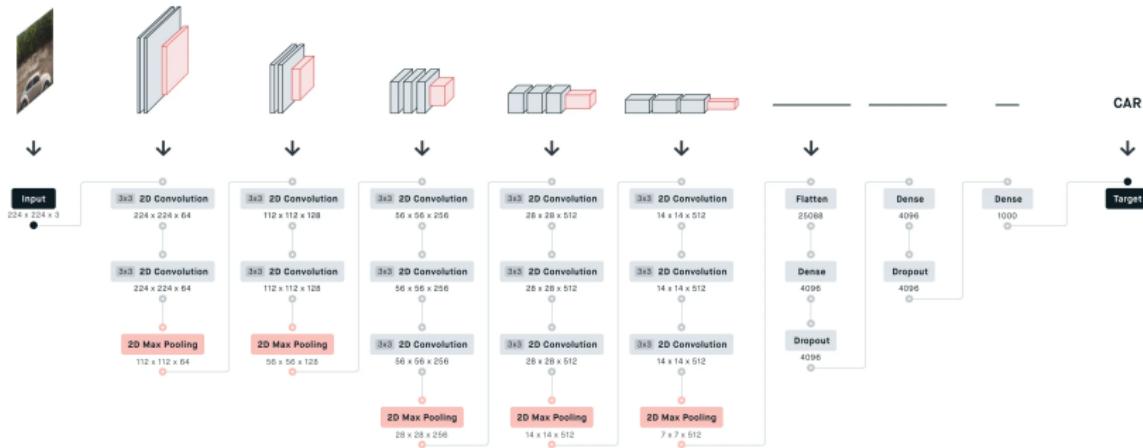
[4096] FC7: 4096 neurons with F=1

[1000] FC8: 1000 neurons (class scores)

Arquitecturas de redes convolucionales

VGG

VGG

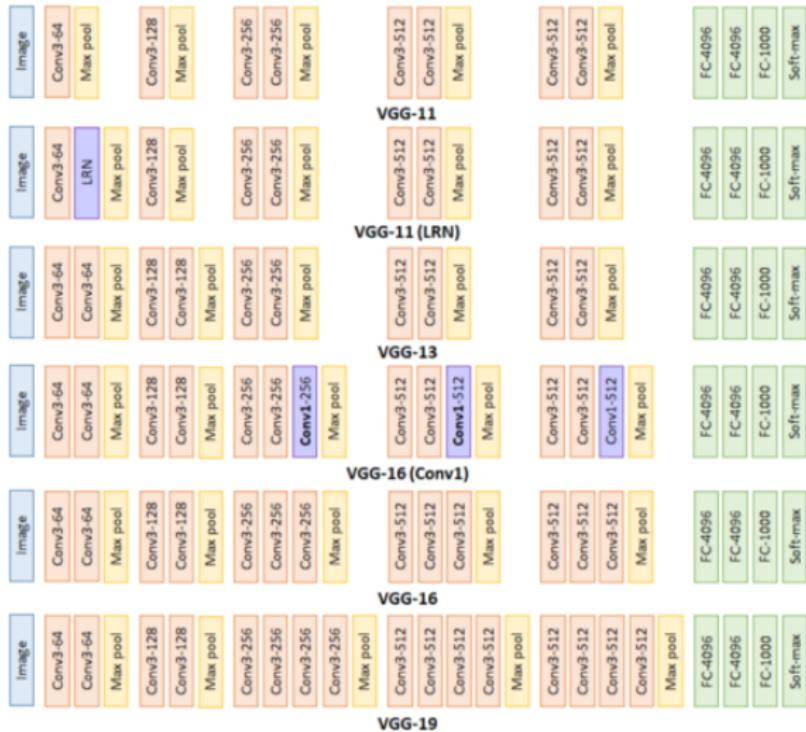


- ▶ Paper: Karen Simonyan, Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556, 2014.
- ▶ Capas más profundas y filtros pequeños (3x3).
- ▶ Modularidad: se repiten bloques convolución(3x3)-pooling(2x2) para construir diferentes versiones: VGG-11, VGG-13, VGG-16, VGG-19.

Arquitecturas de redes convolucionales

VGG

Versiones VGG



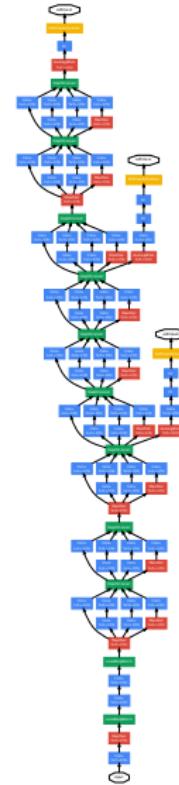
Arquitecturas de redes convolucionales

GoogLeNet - Inception



Inception

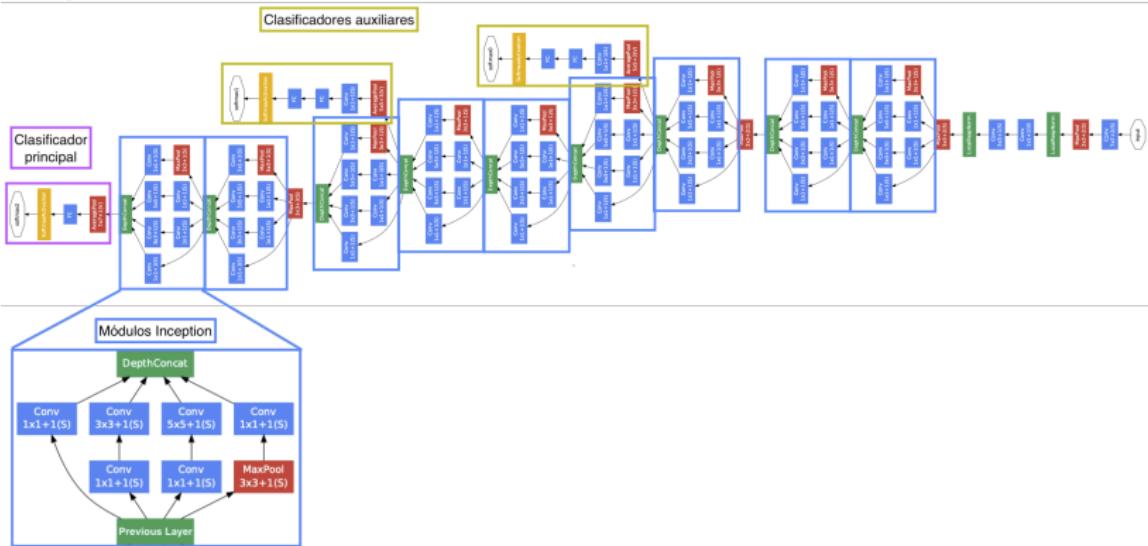
- ▶ Paper: C. Szegedy, et al., *Going deeper with convolutions*, IEEE Conf. on Computer Vision and Pattern Recog. (CVPR), USA, 2015.
- ▶ Módulo Inception: convoluciones en paralelo con diferentes tamaños de filtros (1×1 , 3×3 , 5×5).
- ▶ Convoluciones 1×1 : reducción de la dimensión y agregado de no-linealidad en capas intermedias.
- ▶ Clasificadores auxiliares
 - ▶ Mejora el vanishing gradient problem.
 - ▶ Sólo utilizados durante el entrenamiento.
 - ▶ Peso de 0.3 en la loss total.



Arquitecturas de redes convolucionales

GoogLeNet - Inception

Inception

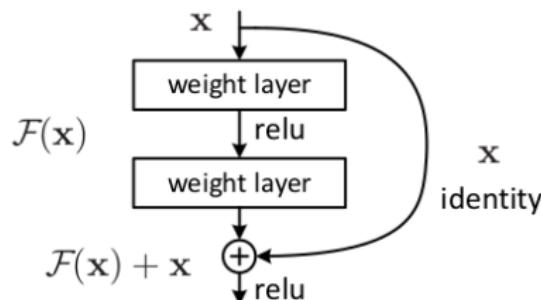


Arquitecturas de redes convolucionales

ResNet

ResNet: Residual Network

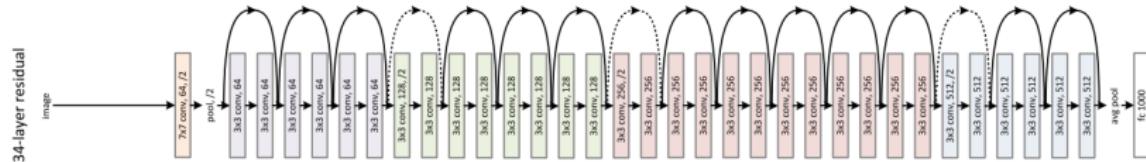
- ▶ Paper: K. He, X. Zhang, S. Ren and J. Sun, *Deep Residual Learning for Image Recognition*, IEEE Conf. on Computer Vision and Pattern Recog. (CVPR), USA, 2016, pp. 770-778.
- ▶ Bloques residuales: se agrega una *skip connection* que conecta directamente entrada con salida de la capa, y se suma al resultado del bloque convolucional.



- ▶ Bloques residuales permiten arquitecturas más profundas y evitan el vanishing gradient problem.

Arquitecturas de redes convolucionales

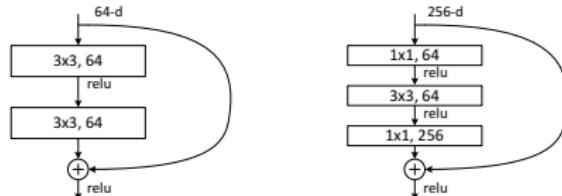
ResNet



ResNet: Residual Network

- Variantes: ResNet-18 / 34 / 50 / 101 / 152.

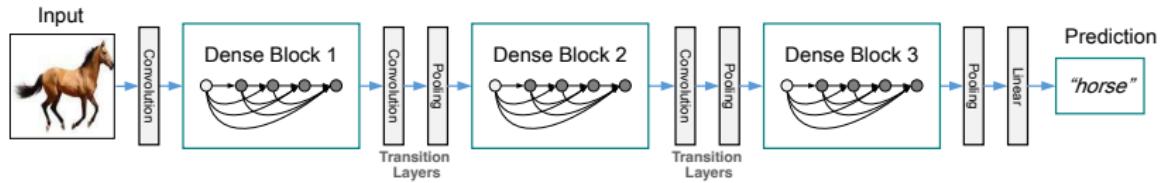
Bottleneck (derecha)



- Diseñado para reducir cantidad de parámetros y cálculos
- Presentes en arquitecturas más profundas (ResNet-50/101/152)
- “bottlenecks”: 1×1 (reduce) + 3×3 (procesa) + 1×1 (restaura)
- Así el bloque 3×3 trabaja con menos dimensiones

Arquitecturas de redes convolucionales

DenseNet



DenseNet: Densely Connected Convolutional Networks

- ▶ Paper: G. Huang, Z. Liu, L. Van Der Maaten and K. Weinberger, Densely Connected Convolutional Networks, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), USA, 2017 pp. 2261-2269.
- ▶ Dense connections: concatenación con todas las salidas de capas anteriores.
- ▶ Redes Tradicionales: $\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1})$
- ▶ ResNet: $\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}$
- ▶ Densenet: $\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}])$
[$\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}$] → concatenación de las capas 0 a la $\ell - 1$.
 H : BN + ReLu + Conv 3×3
- ▶ Transition Layers: Conv 1×1 + Average Pooling para reducir dimensión
- ▶ Variantes: DenseNet 121/169/201

Arquitecturas de redes convolucionales

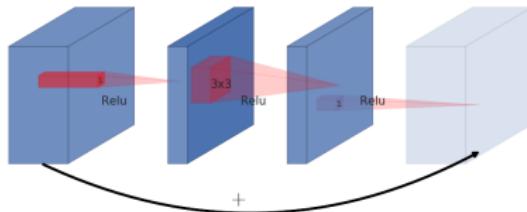
MobileNet

MobileNet

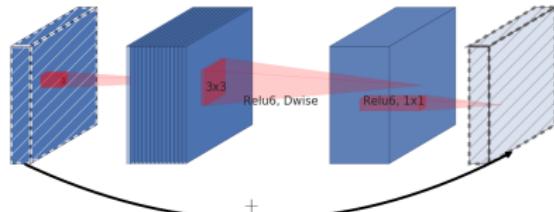
- ▶ Paper: A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv:1704.04861, 2017.
- ▶ Depthwise separable convolutions + Pointwise convolutions.
- ▶ Variantes
 - ▶ MobileNetV2: inverted convolutions
 - ▶ MobileNetV3: NAS (Neural Architecture Search)

Inverted Convolutions

(a) Residual block



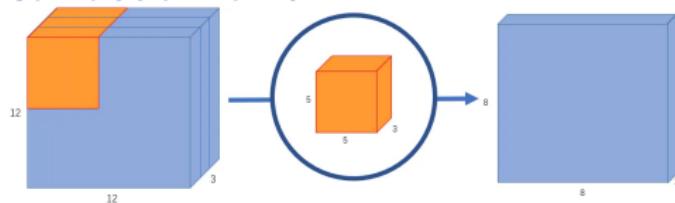
(b) Inverted residual block



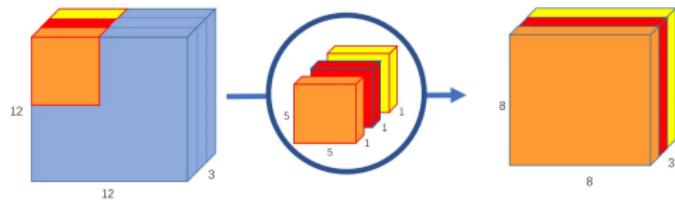
Arquitecturas de redes convolucionales

MobileNet

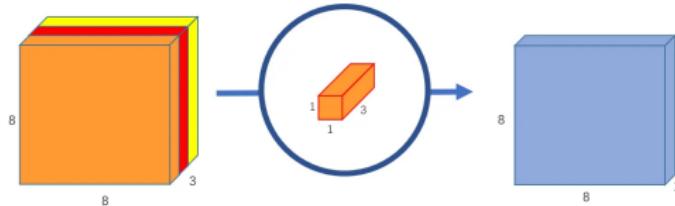
Convolución normal



Depthwise separable convolutions



Pointwise convolutions



Arquitecturas de redes convolucionales

ShuffleNet

ShuffleNet

- ▶ Paper: Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices.* IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2018.
- ▶ Grouped convolutions + channel shuffle.

Arquitecturas de redes convolucionales

EfficientNet

EfficientNet

- ▶ Paper: Mingxing Tan, Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 36th Int. Conf. on Machine Learning (ICML), 2019.
- ▶ Restricciones para obtener la mejor arquitectura:

$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{sujeto a } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Donde ϕ es un parámetro ajustable por el usuario que regula los recursos disponibles y α, β, γ son parámetros que se determinan por una pequeña grid search.

- ▶ Según la convención utilizada por los autores:
 - ▶ $d = 1 \rightarrow 18$ capas convolucionales
 - ▶ $r = 1 \rightarrow$ tamaño de 224×224 de las imágenes.

Arquitecturas de redes convolucionales

EfficientNet

i	Operator $\hat{\mathcal{F}}_i$	Resol. $\hat{H}_i \times \hat{W}_i$	#Chan \hat{C}_i
1	Conv3x3	224×224	32
2	MBConv1, k3x3	112×112	16
3	MBConv6, k3x3	112×112	24
	MBConv6, k3x3	112×112	24
4	MBConv6, k5x5	56×56	40
	MBConv6, k5x5	56×56	40
	MBConv6, k3x3	28×28	80
5	MBConv6, k3x3	28×28	80
	MBConv6, k3x3	28×28	80
6	MBConv6, k5x5	14×14	112
	MBConv6, k5x5	14×14	112
	MBConv6, k5x5	14×14	112
	MBConv6, k5x5	14×14	192
7	MBConv6, k5x5	14×14	192
	MBConv6, k5x5	14×14	192
	MBConv6, k5x5	14×14	192
	MBConv6, k5x5	14×14	192
8	MBConv6, k3x3	7×7	320
9	Conv1x1 & Pooling & FC	7×7	1280

Table: EfficientNet-B0 Arquitecture. Cada fila describe una capa de tipo $\hat{\mathcal{F}}_i$, input de resolución $\langle \hat{H}_i, \hat{W}_i \rangle$ y \hat{C}_i canales de salida.