

Módulo: Aprendizaje Profundo

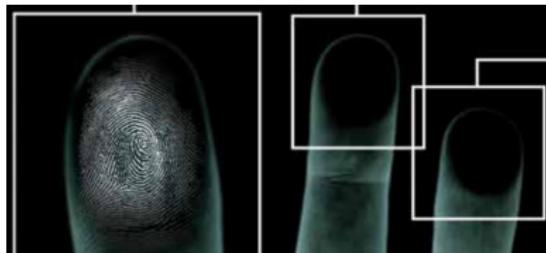
Clase 2: Redes convolucionales. Arquitecturas para deep learning

Temario de la clase 2

1. Localización en una red neuronal
2. Invariancia de traslación.
3. Capas convolucionales y filtros.
4. Alexnet
5. Inception red
6. Redes residuales.
7. Autoencoders

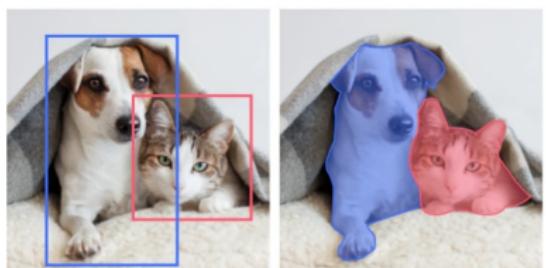
Aprendizaje automático en imágenes

1. Identificación de objetos.
2. Seguimiento de objetos.
Predicción de videos.
3. Segmentación de imágenes.
Clasificación de los pixeles.
4. Generación de nuevas imágenes
5. Reconstrucción 3D a partir de varias imágenes.
6. Rellenado de parte de la imagen (borrado de objetos/caras/etc)
7. Downscaling o super-resolución.
8. Descripción de la imagen.



Object Detection

Instance Segmentation



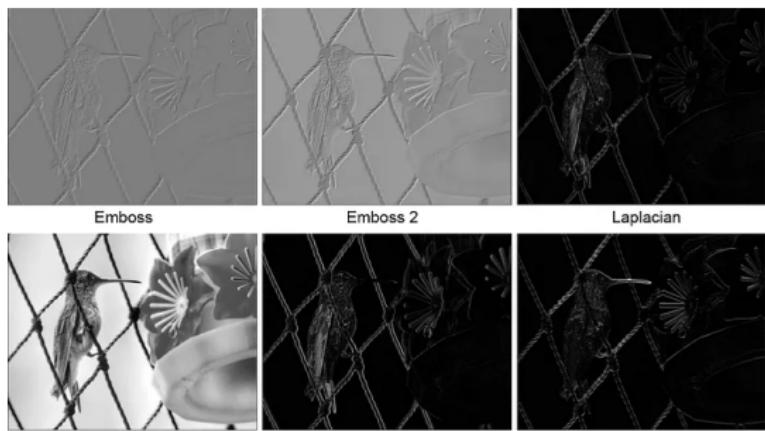
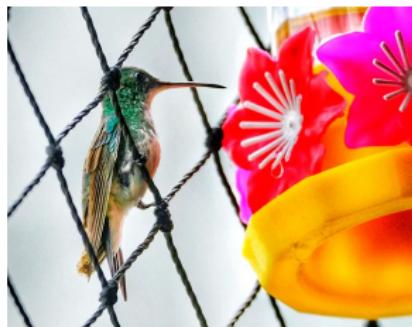
Cat, Dog

Cat, Dog

Multiple objects

Aplicaciones redes FCNN en imágenes?

- ▶ Si la entrada es una imagen tenemos $N_p \times N_p$ en escala de grises.
- ▶ Si fuera color son 3 canales $3 \times N_p \times N_p$. Cada intensidad/variable es un entero de 8 bits, 0-255.



La imagen está estructurada con pixeles cercanos correlacionados entre sí en color y textura.

Aplicación de FCNN en imágenes?

- ▶ Cuantos parámetros necesitamos para una fully connected? N_p^4 (una capa interna)
- ▶ Si $N_p = 100$ y $N_1 = 1000$ (neuronas capa interna) requerimos 10^7 !!!!
- ▶ Peor aun, para trabajar con una imagen hay que convertirla en un vector (flattening) perdiendo toda la estructura 2D que la imagen tiene.

La FCNN no esta teniendo en cuenta las fuertes correlaciones entre pixeles cercanos.

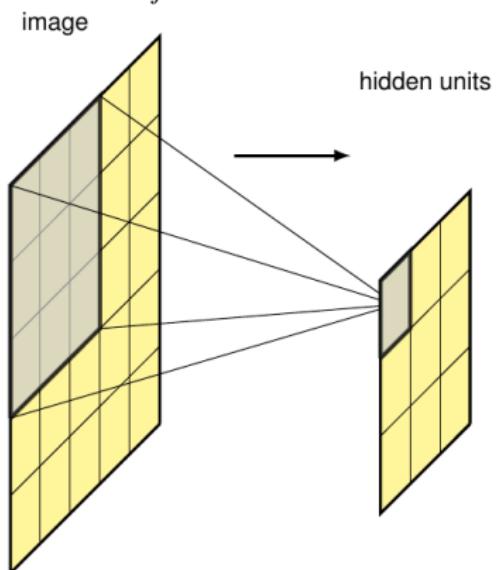
Localidad: Campo receptivo de un pixel

Hipótesis de localidad: Los pixels estan correlacionados entre si pero solo con los vecinos mas cercanos.

Los pixels lejanos son independientes del pixel en cuestión:

$$p(x_j|\mathbf{x}) = p(x_j|\mathbf{x}_{\Gamma_j})$$

donde Γ_j es el **Markov blanket** o la vecindad alrededor de j .



- ▶ No tiene sentido usar una red fully connected. Parámetros igual a 0.
- ▶ **Capa local o red local:** cuando cada nodo de la oculta recibe entradas de solo una vecindad restringida de la capa anterior.
- ▶ Esta vecindad es llamada el **campo receptivo del nodo**.
- ▶ Estos campos receptivos suelen ser de 3x3 o de 5x5 etc. Porque impar?.

Locally connected neural network

Local network: para calcular un nodo de la capa oculta solo usamos los nodos vecinos (de la capa previa).

Para un dado pixel i, j en la capa l la operación que hacemos es:

$$x_{ij}^{(l+1)} = \mathbf{W}_{ij}^{(l)} \mathbf{x}_{\Gamma_{ij}}^{(l)}$$

donde $\mathbf{W}_{ij}^{(l)}$ es la matriz local de 3x3.

Estos pesos o matriz local la vamos a llamar filtro o kernel (nucleo).

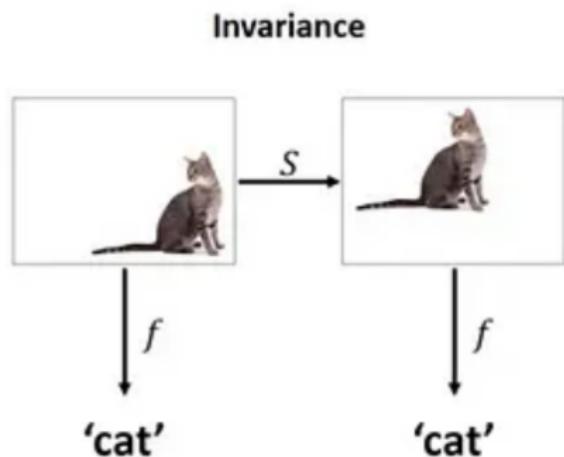
Para cada pixel de la imagen estamos haciendo una doble suma alrededor de ésta:

$$x_{ij}^{(l+1)} = \sum_{m=-N_k/2}^{N_k/2} \sum_{n=-N_k/2}^{N_k/2} W_{ij,mn}^{(l)} x_{i+n,j+m}^{(l)}$$

Debido a la hipótesis de localidad la cantidad de parámetros es: $N_p^2 N_k^2$ si $N_p = 100$ y $N_k = 3$ tenemos $9 \cdot 10^4$. Se disminuyó 1000 veces la cantidad de parámetros.

Cada pixel tiene N_k^2 parámetros.

Invarianza de traslación



- ▶ Para la clasificación no importa la posición en la que se encuentra el objeto.
- ▶ Invarianza: La salida permanece sin cambios cuando se realizan transformaciones de traslación.
- ▶ Los parámetros deberían ser independientes de la posición.

Parámetros compartidos, $\mathbf{W}^{(l)}$ es independiente de i, j :

$$x_{ij}^{(l+1)} = \mathbf{W}^{(l)} \mathbf{x}_{\Gamma_{ij}}^{(l)}$$
$$x_{ij}^{(l+1)} = \sum_{m=-N_k/2}^{N_k/2} \sum_{n=-N_k/2}^{N_k/2} W_{mn}^{(l)} x_{i+n, j+m}^{(l)}$$

Los parámetros no dependen de la posición y el mismo kernel se utiliza en todas las posiciones.

Hipótesis de las redes convolucionales

Son dos hipótesis totalmente independientes:

- ▶ **Localidad.** Los pixels solo se relacionan con sus vecinos.
- ▶ **Equivarianza de traslación.** Los parámetros son compartidos.

Multichannels convolutions

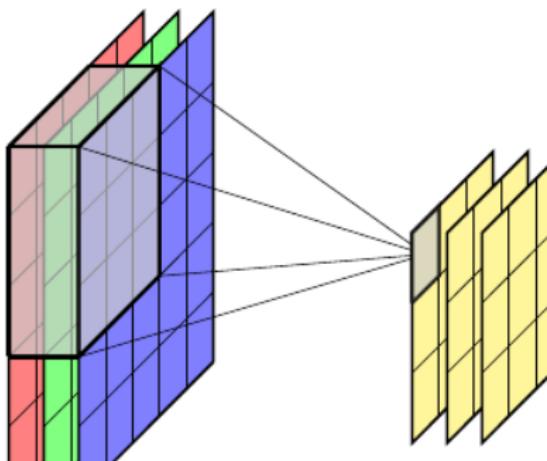
Hasta el momento tenemos un solo mapa de características.

Si queremos aprender múltiples mapas de características ponemos múltiples kernels en “canales” $N_k \times N_k \times N_C$ sería la cantidad parámetros.

El nro de canales internos corresponde al número de filtros que vamos a aplicar (y optimizar).

$$x_{ijc}^{(l+1)} = \sum_{d=0}^{N_c^{(l)}} \sum_{m=-k/2}^{k/2} \sum_{n=-k/2}^{k/2} W_{mnc}^{(l)} x_{i+n, j+m, d}^{(l)}$$

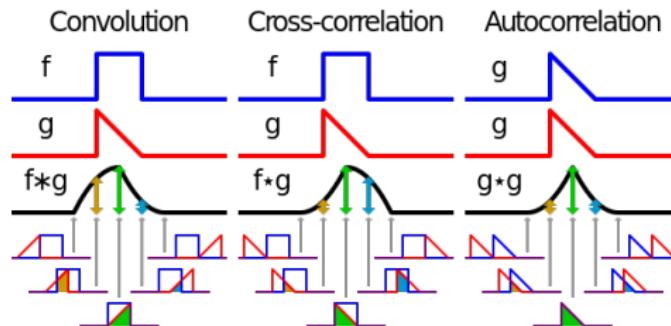
donde $x_{ijc}^{(l)}$ tiene $N_C^{(l)}$ y $x_{ijc}^{(l+1)}, N_C^{(l+1)}$.



Definición del operador convolución. Matemáticamente

La convolución de dos funciones f y g es denotada por $*$ y está definida por la integral del producto de los dos funciones después de que una de ellas se revierte y se desfase:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau = \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau$$



En el caso discreto con un filtro/kernel de dimension (K, K) ,

$$x_{ij}^{(k+1)} = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x_{i-m, j-n}^{(k)} K_{nm}^{(k)}$$

Padding



1	2	1
2	4	2
1	2	1

1	2	3	2
2	4	6	4
3	6	9	6
2	4	6	4

Como mantengo la misma dimensión de la imagen?

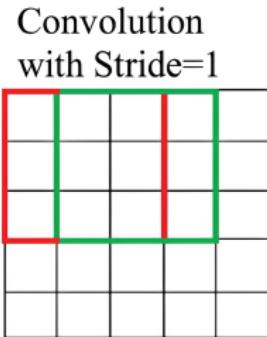
Dado un vector de datos cuando comenzamos y cuando terminamos no tenemos datos para generar la convolución de la primera componente del vector:

$$y_j = \sum_{i=-N_k//2}^{N_k//2} w_{i+N_k//2} x_{j+i}$$

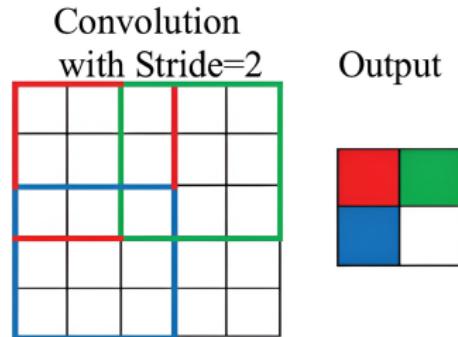
Que pasa para $j = 0, N_k = 5$?

- ▶ Agrego $N_k//2$ puntos con 0s: $x_{-2} = x_{-1} = 0, x_{N_x} = x_{N_x+1} = 0$.
- ▶ Asumo dominio cíclico o periódico $x_{-1} = x_{N_x-1}, x_{-2} = x_{N_x-2}, x_{N_x} = x_0$
- ▶ Asumo dominio espejo: $x_{-1} = x_1, x_{-2} = x_2$
- ▶ Reduzco componentes del vector de salida en $2 \times N_k//2$.

Convoluciones con salto o paso: Stride



Output



Output

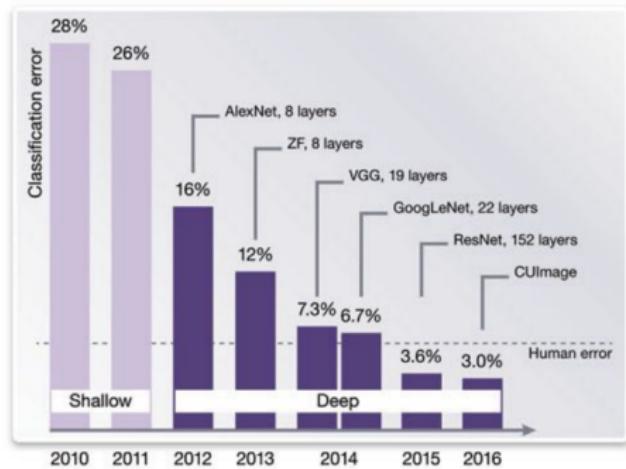
- ▶ Cuando movemos el filtro para calcular el próximo nodo podemos saltar de a 1, `stride=1`. Panel derecho.
- ▶ Esto significa si el centro del filtro es i, j el próximo centro será $i + 1, j$ etc.
- ▶ Una alternativa es que hagamos un salto. Supongamos un `stride=2`. Panel izquierdo.
- ▶ Si el centro del filtro es i, j el próximo centro será $i + 2, j, i + 4, j$, etc. Cuando recorramos en j , lo mismo $i, j + 2, i, j + 4$.

Convolución en pytorch

```
self.capa1= torch.nn.Conv2d(in_chan, out_chan,  
kernel_size=3, stride=1, padding=kernel_size//2)
```

- ▶ Esto va a crear out_chan kernels.
- ▶ Cada kernel es de 3er orden: (3, 3, in_ch).
- ▶ El total de parámetros de la capa es: $\text{in_chan} \times \text{out_chan} \times 3 \times 3$.

Papers mas relevantes de Image Recognition



- ▶ Todos los papers basados en la base ImageNet.
- ▶ Importantes avances en la performance en reconocimiento de objetos.
- ▶ La performance se correlaciona con el aumento de capas (mas profundo mas performance).
- ▶ ResNet tiene menor error que un humano!

Fuente Reddit.

ImageNet

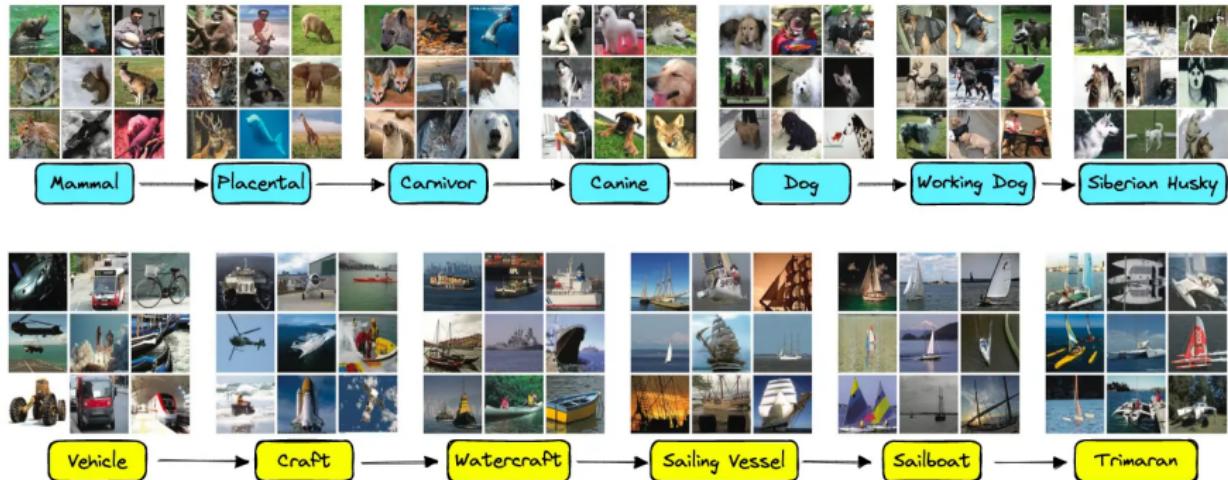
<https://image-net.org/>

- ▶ Base de datos con mas de 14 millones de images de alta resolución.
- ▶ Los objetos estan clasificados en 22.000 clases.
- ▶ ImageNet marca un antes y un despues en el avance del machine learning por la sola razon que era una gran base de datos y era abierta
- ▶ Armada a través de web scrapping.

Deng J, W Dong, et al., ImageNet: A Large-Scale Hierarchical Image Database.
IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

Russakovsky O, J Deng, et al. ImageNet Large Scale Visual Recognition Challenge.
IJCV, 2015.

ImageNet



Desafío: **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** desde 2010. Detección de múltiples objetos en imágenes y su clasificación. Las imágenes tienen resoluciones variables. Ref 400x400. Se transforman a 256x256 o 224x224.

Historia de la clasificación de imágenes

- ▶ **LeNet-5 (1998)** LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
- ▶ **AlexNet (2012)** Kizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in NIPS, 1097-1105.
- ▶ **ZFNet (2013)** Zeiler, M.D. and Fergus, R., 2014. Visualizing and understanding convolutional networks. In Computer Vision-ECCV 2014
- ▶ **VGGNet (2014)** Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- ▶ **GoogLeNet (2014)** Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE
- ▶ **ResNet (2015)** He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE 770-778. 170.000 citas!

Downsampling o subsampling

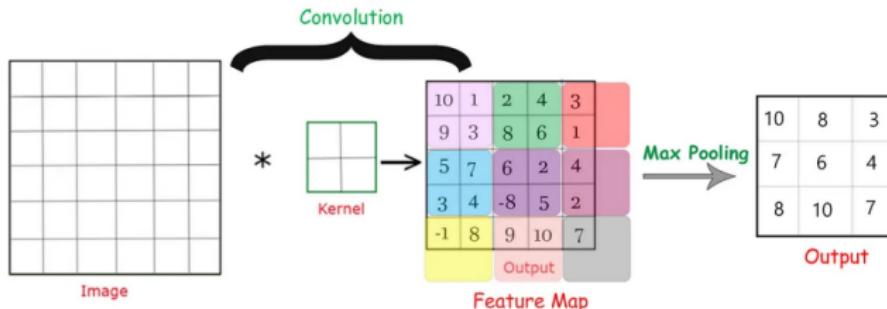
¿Como disminuyo la resolución de las imágenes?

Supongamos que quiero ir de 256 pixels a 128. Manteniendo el mismo dominio/apertura.

- ▶ Puedo ir saltando de dos pixels y me quedo entonces con los pares (**stride**). En dos dimensiones son rectángulos.
- ▶ Tomo el máximo entre dos pixels o en el rectángulo de dos por dos (**max-pooling**).
- ▶ Promedio los pixels del rectángulo: **Average pooling**

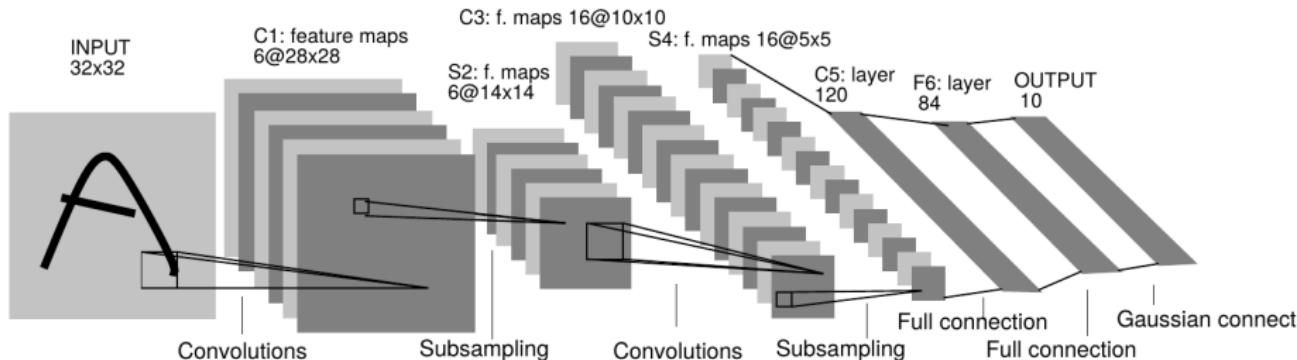
Lo debo realizar para cada canal por separado.

Downsampling: Max-pooling



- ▶ Invariancia en la posición.
- ▶ Invariancia en la rotación.
- ▶ Invariancia a la escala (chico o grande)
- ▶ Selecciona las características mas importantes de la entrada.
- ▶ Pierde información. El average pooling conserva mas información.
- ▶ Puede causar sobre-suavizado de las características, con pérdida de detalles de peq. escala.

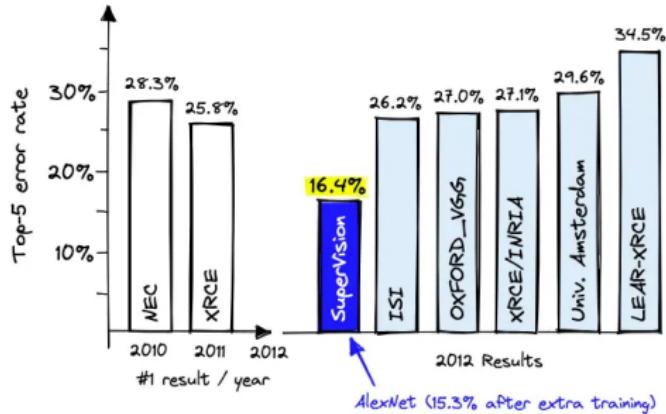
Precursor: LeNet-5



Desarrollada por varios años para digitalizar los códigos postales a partir de las cartas. También se utiliza en cajeros.

- Kernels 5x5
- El subsampling se hacia con la suma de los 4 vecinos.
 $\sigma[(\alpha \sum_{n=1}^4 C_n + \beta)]$. Luego se aplica una sigmoide.
- No se aplican todas las conexiones entre S2 y C3.
- Entre S4 y C5 hay una FC. Tangente hiperbólica como función de activación.
- La última capa se compone de unidades de RBFs con 84 inputs c/u. La salida de la RBF es $y_i = \sum_j (x_j - w_{ij})^2$.

La bisagra: Alexnet



Aplicación: ImageNet Large Scale Visual Recognition Challenge

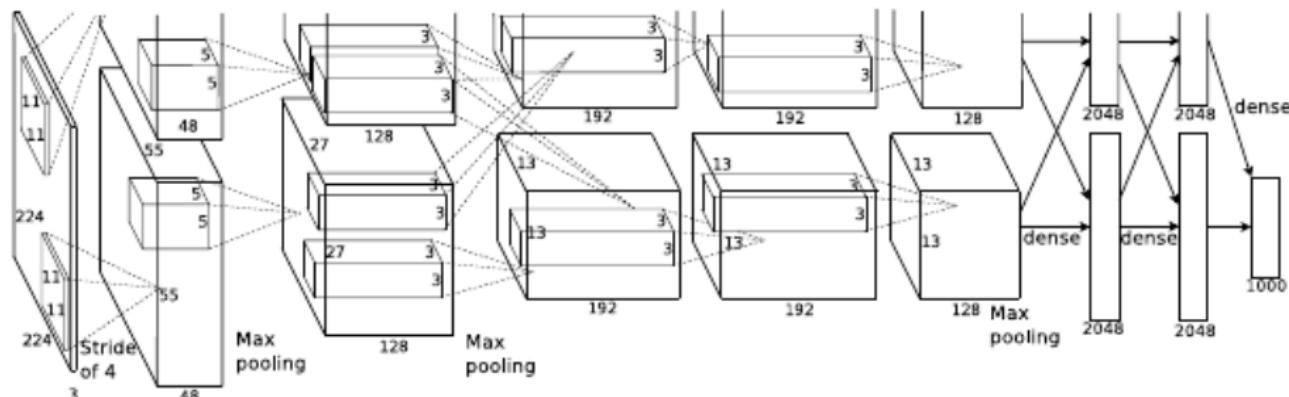
Implementada en GPUs (2 x GTX 580).

La profundidad del modelo es esencial para obtener alta performance.

Tremenda performance!!!, comparado a las tecnologías del momento.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in NIPS, 1097-1105.

Arquitectura Alexnet



8 capas. La división en dos es por las GPUs.

5 primeras con convolucionales. Algunas con maxpooling.

Las últimas tres son FC.

$(CNN \rightarrow RN \rightarrow MP)^2 \rightarrow (CNN^3 \rightarrow MP) \rightarrow (FC \rightarrow DO)^2 \rightarrow Linear \rightarrow softmax$

CNN = capa convolucional (con funcion de activacion ReLU)

RN = normalización local

MP = max-pooling

FC = capa fully connected (con funcion de activacion ReLU)

Linear = capa fully connected (sin activacion)

DO = dropout

¿Porque Alexnet fue un landmark?

Resultados: AlexNet ganó la competición de ImageNet 2012 con un error top-5 de 15.3%, comparado al 2do lugar que obtuvo 26.2% (por afano).

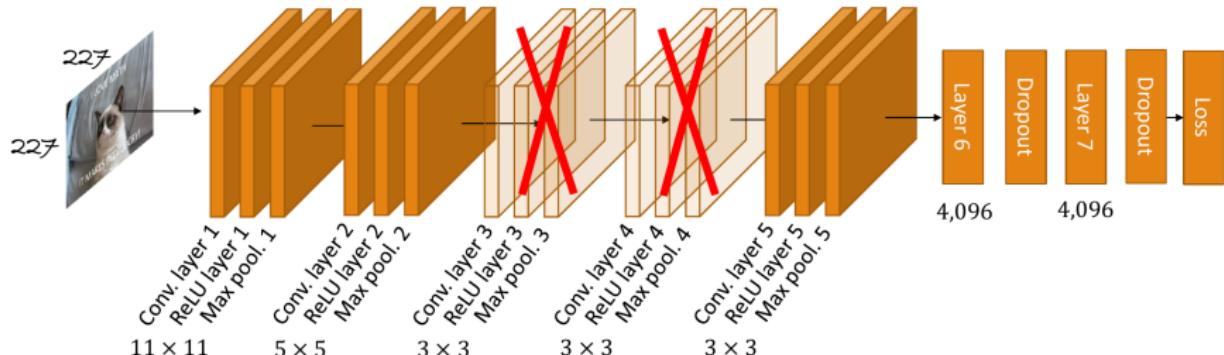
- ▶ **Activación con ReLU.** AlexNet usaba ReLU en lugar de la tanh. Las ReLUs se pueden entrenar mucho mas eficientemente.
- ▶ **Overlapping Pooling.** Evaluan el overlap en la aplicación de las CNNs disminuye el error y encuentran menos overfit.
- ▶ Batch size of 128. Algoritmo de aprendizaje: SGD Momentum
- ▶ **Problema de overfitting:** AlexNet tenía 60 millones de parámetros (gracias a las 2 GPUs de 3gb), para disminuir el overfitting se uso:
 - ▶ **Data Augmentation.** Se generaron imágenes a través de reflecciones horizontales y traslaciones.
 - ▶ **Dropout.** Uno de los primeros modelos de producción en aplicar el dropout.

Alexnet in pytorch

```
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            [[VARIAS MAS]]... )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(in_features=256 * 6 * 6, out_features=4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes) )
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

¿Que rol tienen las capas?

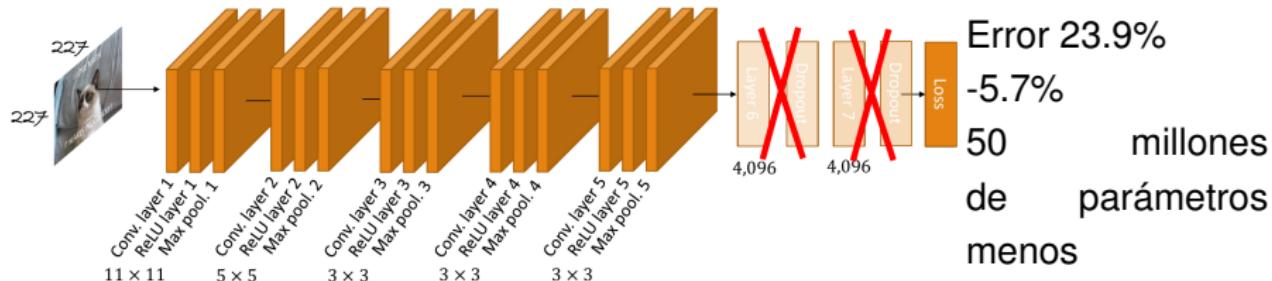
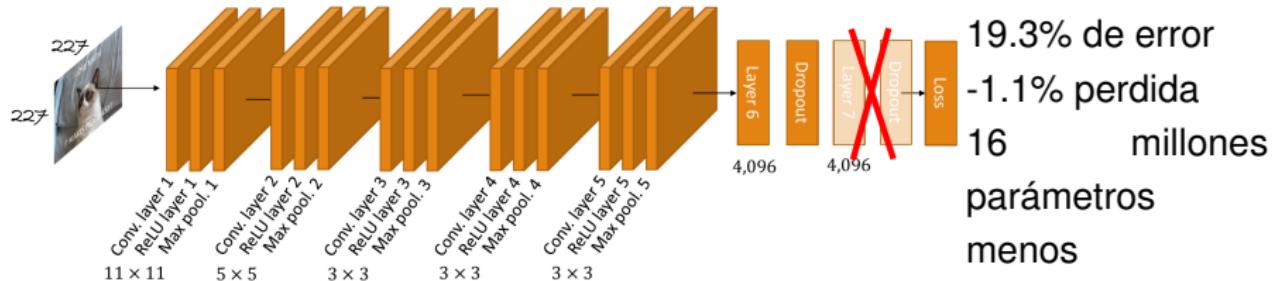
¿Que pasa si disminuimos la cantidad de capas convolucionales?



No mucho pasamos de 18.2% a 21.2% de error (3% de perdida de performance).

Hemos disminuido en 1 millón de parametros.

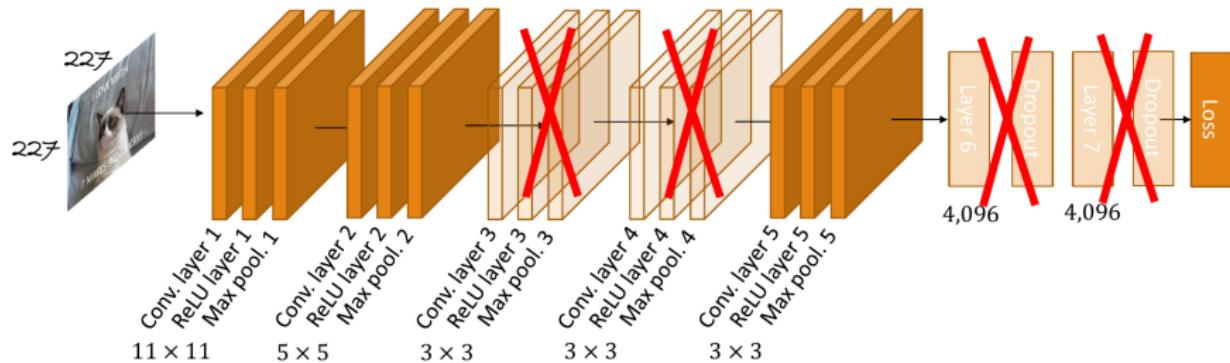
¿Que pasa si disminuimos la cantidad de capas fully connected?



En términos de memoria es conveniente recortar capas FC en lugar de Conv.

Que rol tienen las capas

¿Que pasa si disminuimos ambas capas convolucionales y capas fully connected?



Pasamos de 18.2% a 51.7% de error. **Se nos derrumbó la performance!**

¿Porque? **La profundidad es esencial para representar las características de las imágenes.**

Interpretación de las distintas capas

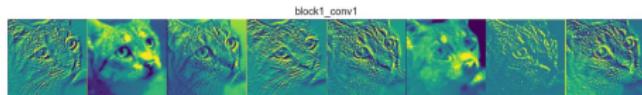
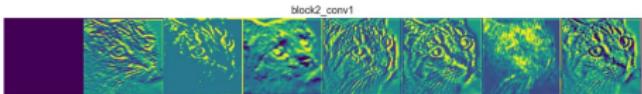
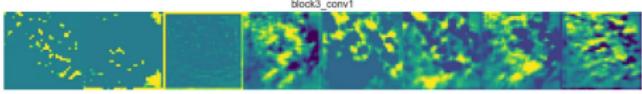


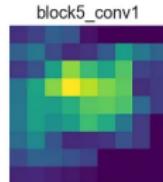
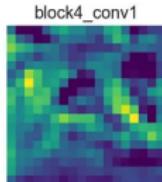
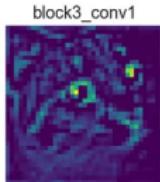
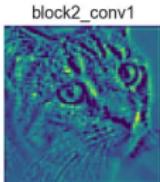
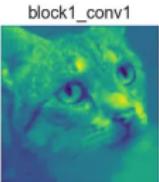
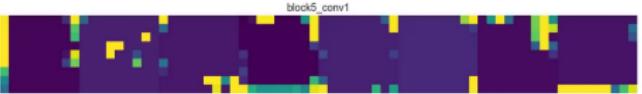
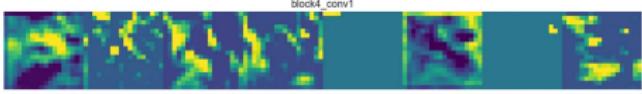
Imagen original



En las primeras capas se concentra en las características de pequeña escala de los objetos.
Bordes. Ojos. Nariz.



A medida que profundiza es capaz de capturar las características de mas largo alcance de la imagen (el big picture).

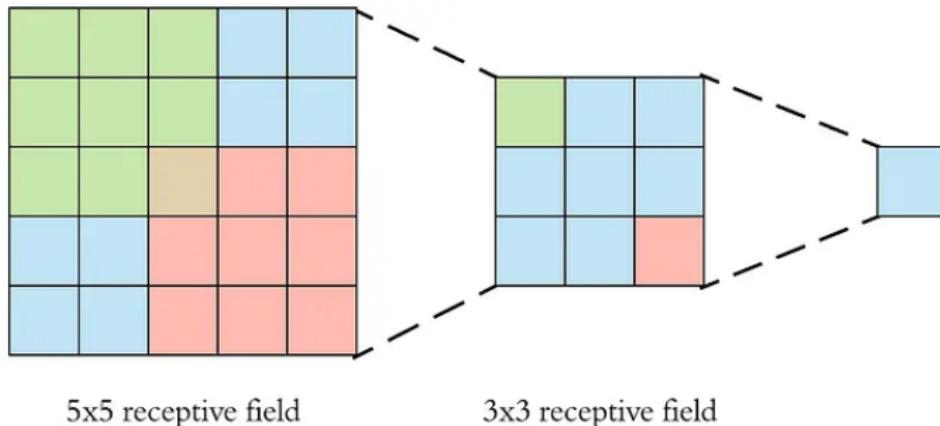


CNN explainer: <https://poloclub.github.io/cnn-explainer/>

¿Qué tamaño deben tener los filtros?

El tamaño mas utilizado actualmente es 3×3 . Porque?

- ▶ Es el kernel mas pequeño con estructura.



- ▶ Se puede representar campos receptivos mas largos “estaqueando” filtros 3×3 :
 - ▶ Dos filtros 3×3 tienen un campo receptivo de 5×5 .
 - ▶ Tres filtros 3×3 tienen un campo receptivo de 7×7 .

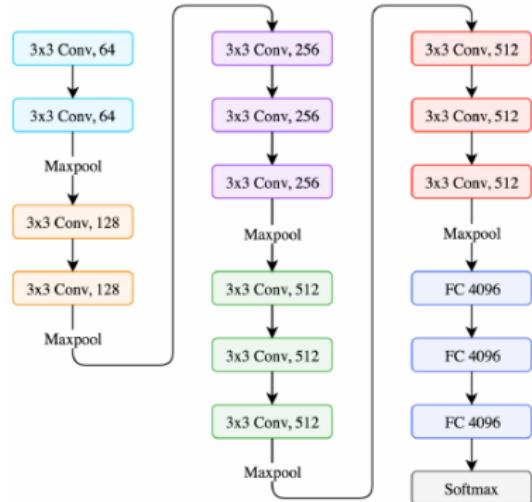
¿Qué tamaño deben tener los filtros?

El tamaño mas utilizado actualmente es 3×3 . Porque?

- ▶ Es el kernel mas pequeño con estructura.
- ▶ Si tenemos un stack de varios filtros pequeños tenemos mas composiciones y no-linealidades (al mismo precio).
- ▶ Los parametros para aprender son menos.

Conclusión: un kernel con gran campo receptivo puede ser reemplazado por un stack de kernels pequeños con mejor performance.

Red VGG



- ▶ VGG error en ImageNet 7.3%.
- ▶ Mejora significativa frente a la Alexnet 16%.
- ▶ VGG-16 (13 Conv y 3 FC) tiene 138M de parametros
- ▶ Requerimiento de memoria de 48.6 Mb.
- ▶ AlexNet que ocupa 1.9 Mb.

Visual geometry group (**VGG**)!. Oxford University

Simonyan, K. and Zisserman, A., 2014. **Very deep convolutional networks** for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Red VGG

Filosofía de las redes VGG:

- ▶ Aumentar la profundidad de la red para representar mas composiciones no-lineales mejorando la predicción.
- ▶ Campos receptivos pequeños de 3x3.
- ▶ Arquitectura homogénea (bloques de estructuras) → **minimiza nro de hiperparámetros**

La estructura de los bloques de convolucionales de la VGG es la característica usada actualmente en redes convolucionales.

Bloque de la Red VGG

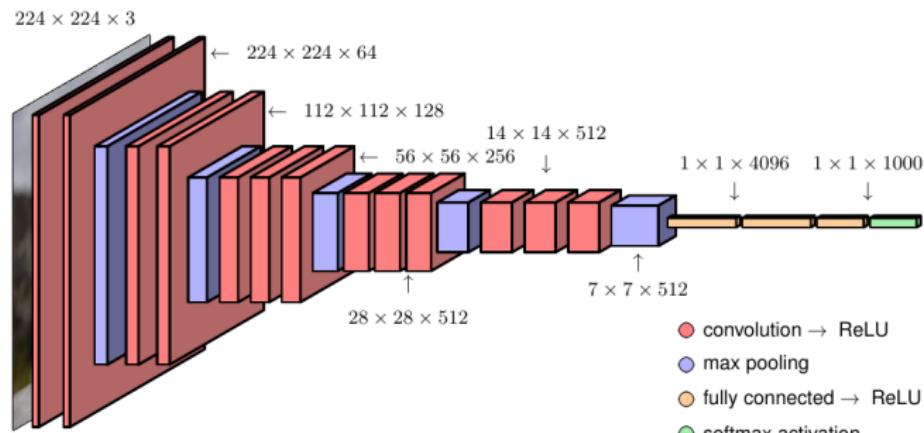
La VGG tiene una estructura homogénea que se repite en las distintas capas.

Cada bloque contiene:

- ▶ 3 convoluciones de kernel 3x3 con padding de 1 y stride de 1.
- ▶ Función de activación ReLU.
- ▶ Max poling con stride de 2 y kernel de 2x2. (las imágenes se achican por 4!).
- ▶ Sin normalizaciones.

A medida que disminuyo dimensión aumento el nro de canales.

Dimensiones del estado escondido de la VGG



Estructura de la red en términos de las dimensiones de $x^{(l)}$.

El max-pooling: 224×224 a 112×112 a 56×56 a 28×28 a 14×14 a 7×7

Es decir 5 aplicaciones de max-pooling $224/2^5 = 7$.

La salida del último max-pooling 7×7 se la pasamos a las 3 capas FC

Luego aplicamos la función de activación softmax para clasificar las imágenes. Recordar que tenemos 1000 clases!

Ejercicio: Determinar la cantidad de parámetros de la red VGG-16. Rta 138 millones

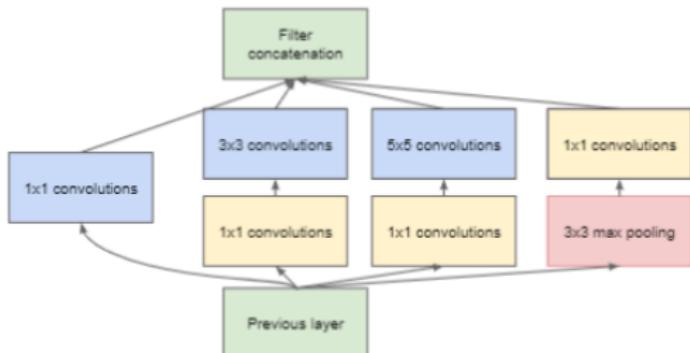
Convoluciones 1x1

- ▶ Es un pooling de canales.
- ▶ Motivación: Cuando quiero cambiar el numero de canales generalmente para reducir utilizo una convolucion 1x1.
- ▶ Si la entrada es N_x, N_y, N_c la salida sera N_x, N_y, N_d donde d es el numero de canales de salida y es arbitrario.
- ▶ Todo lo que estamos haciendo es una combinación lineal de los canales de entrada (pesando cada canal de entrada).

Utilizadas en [GoogleNet](#). Inception blocks. Szegedy, C., et al, 2015. Going deeper with convolutions.

Reduce numero de parametros.

Trabaja con bloques (inception) de kernels de distinto tamaño.



Normalizaciones en las entradas de redes convolucionales

- ▶ Estandarización de la normal: Esta técnica convierte los datos para que tengan una media 0 y una desviación estandar 1.
Se requiere entonces dada las variables de entrada obtener la media μ y la desviación estandar σ . Luego:

$$X_{\text{normalized}} = \frac{X - \mu}{\sigma}$$

Luego de la predicción de la red es necesario anti-transformar.

Muy utilizada en imágenes de escala de grises o RGB donde los pixeles pueden variar mucho.

- ▶ Normalización Min-Max: Transforma los datos a un rango específico segun se requiera entre 0 y 1 o entre -1 y 1.

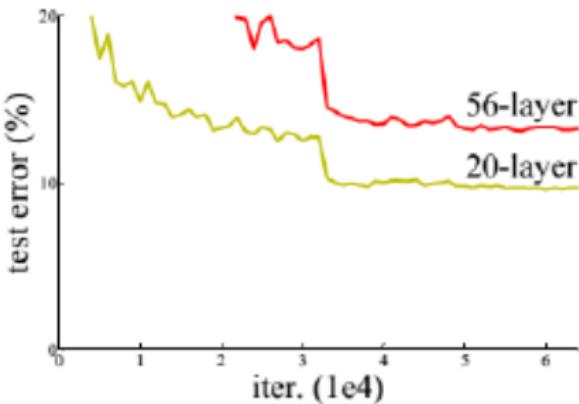
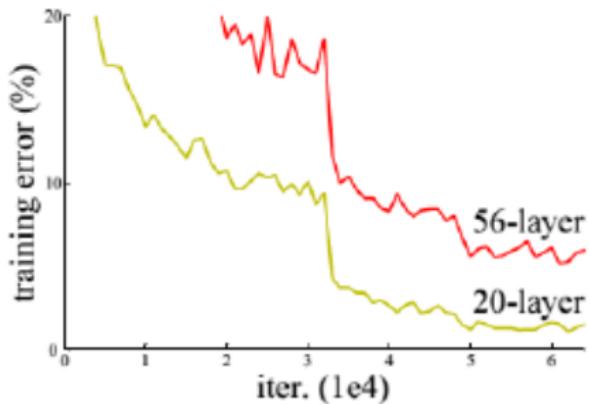
$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}; \quad X_{\text{normalized}} = \frac{2X - (X_{\min} + X_{\max})}{X_{\max} - X_{\min}}$$

- ▶ Transformaciones nolineales para “gaussianizar” a los datos.

Normalizaciones en las capas de las redes convolucionales

- ▶ Normalización por batch (BN): se normalizan las entradas en la capa usando la dimensión del batch. Se requieren batch > 16. Es muy eficiente para las CNNs.
- ▶ Normalización por capa (LN): se normalizan las entradas en la capa usando las dimensiones del estado escondido (la capa). Se utiliza para redes recurrentes y en transformers. No es útil para CNNs.
- ▶ Normalización por grupo (GN): divide los canales en grupos y normaliza cada grupo. Util cuando se trabaja con una tamaño del batch chico <16. Se utiliza en modelos de segmentación semántica.

Sigamos profundizando la red VGG



- ▶ Tanto el error de entrenamiento como el error de testing se deteriora cuando la red se hace mas profunda.
- ▶ Pero porque el error de entrenamiento si tenemos mas grados de libertad!!!!
- ▶ La performance se esta deteriorando por un problema de entrenamiento no por overfitting

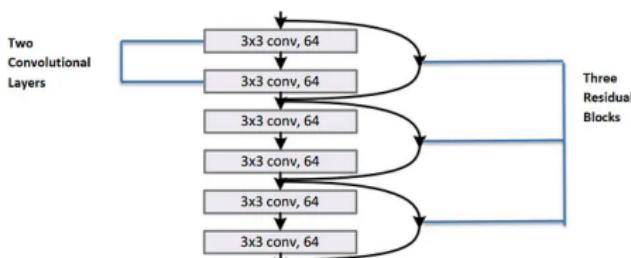
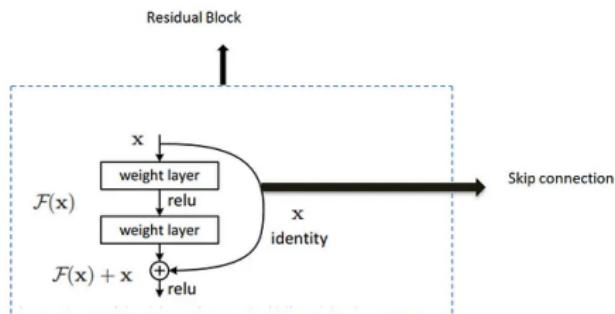
Red residual o salto de conexiones

Considerando que la señal se pierde con gran cantidad de capas.

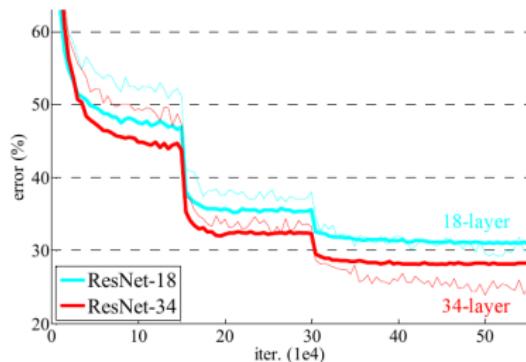
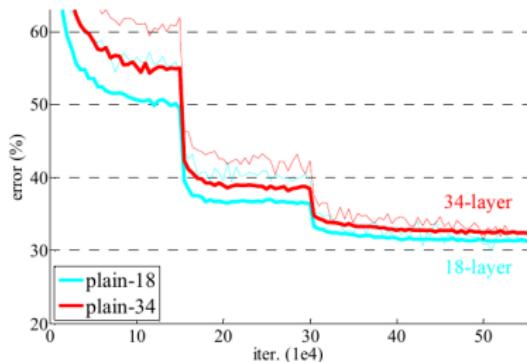
Proponemos modelar las diferencias que los valores absolutos:

$$y = F(x) \rightarrow y = x + G(x)$$

Esto lo podemos ver como que la red deja pasar la información de entrada ergo no perdemos sensibilidad!



Red residual o salto de conecciones



- ▶ La ResNet profunda (34 capas) mejora significativamente a la de 18. Esto no sucede con la estructura de la VGG.
- ▶ La performance de las redes de 18 capas es la misma (Resnet vs VGG).

He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE 770-778. 170.000 citas!

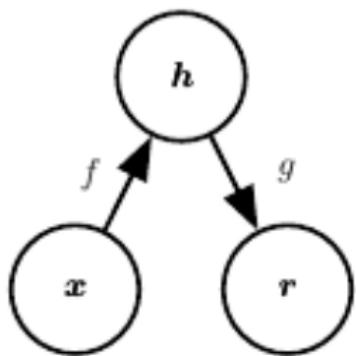
Bloque residual en pytorch

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = conv3x3(in_channels, out_channels, stride)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(out_channels, out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out

model = ResNet(ResidualBlock, [2, 2, 2]).to(device)
```

Autocodificadores



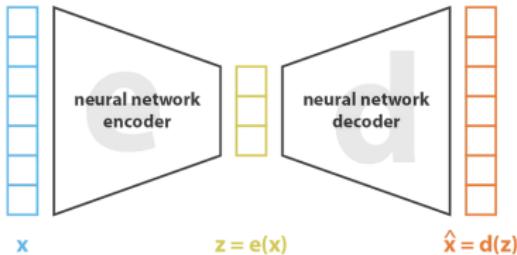
Idea que en principio parece absurda:
Quiero transformar la input \mathbf{x} con f a un
espacio latente, $\mathbf{h} = f(\mathbf{x})$ y luego anti-
transformar con g de vuelta al espacio de
la input $\mathbf{r} = g(\mathbf{x})$.

→ La clave esta en la dimensionalidad: $\mathbf{x} \in \mathbb{R}^{N_x}$, $\mathbf{h} \in \mathbb{R}^{N_h}$ y $\mathbf{r} \in \mathbb{R}^{N_x}$, asumimos que $N_h \ll N_x$

Técnica NO supervisada. No requerimos de “targets”.

Obligo a ir a un **espacio latente pequeño** y a representar todo lo
posible en ese espacio. Si no reduzco estoy copiando.

Autocodificadores



$$\text{loss} = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2$$

- ▶ La complejidad de f también es central para obligar a la codificación.
- ▶ Si f y g son lineales resulta en PCA.
- ▶ El autoencoder es una técnica de reducción de la dimensionalidad no lineal = PCAs no lineales.

Queremos aprender a replicar las inputs, pero aprendiendo de la esencia de x en h/z .

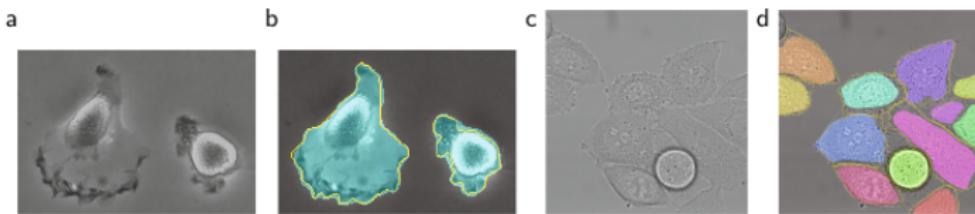
Autocodificadores

Para evitar la memorización de las inputs, $N_h \ll N_x$ con modelos complejos podemos regularizar.

- ▶ Regularización:
 - ▶ representación esparsa $J(\mathbf{x}, g(f(\tilde{\mathbf{x}}))) + \lambda \sum_i \|\nabla_{\mathbf{x}} h_i\|^2$,
 - ▶ derivada pequeña $J(\mathbf{x}, g(f(\tilde{\mathbf{x}}))) + \lambda \sum_i \|\nabla_{\mathbf{x}} h_i\|^2$,
 - ▶ robustocidad al ruido.
- ▶ Modelos generativos: autocodificadores variacionales, redes generativas estocasticas. Aprenden a maximizar la probabilidad (ELBO) evitando la memorización.
- ▶ Autocodificadores para denoising. $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$, función de pérdida: $J(\mathbf{x}, g(f(\tilde{\mathbf{x}})))$.

UNETs

Reconocimiento visual. Clasificación de imágenes.



Aplicaciones: imágenes biomédicas (estructura multiresolución).

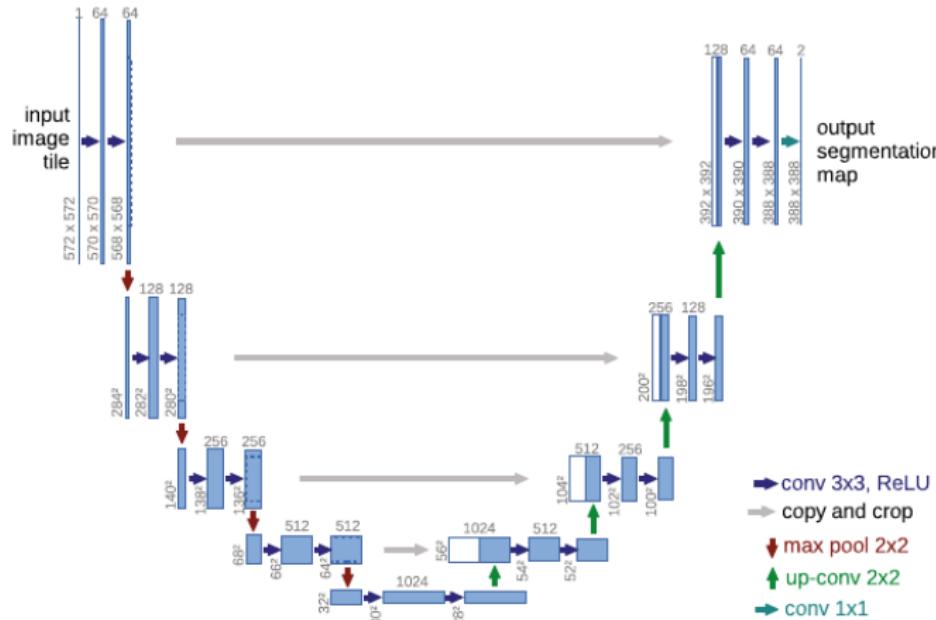
Desafío ISBI (Int. Symp. Biomedical Images) Segmentación automática de estructuras neuronales.

Imágenes de microscopio electrónico.

ISBI seguimiento de células 2015. Segmentación a distintos tiempos.

Objetivo: Requerimos de localización. La etiqueta debe ser asignada a cada pixel de la imagen.

UNETs



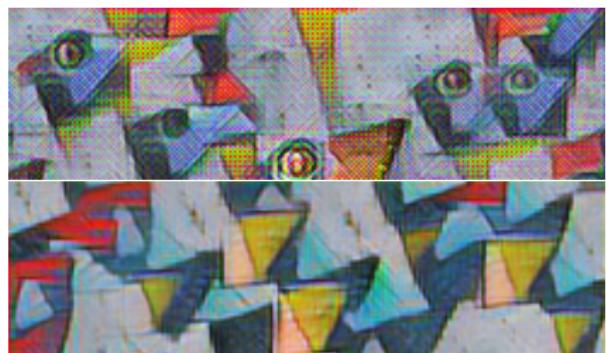
La U es una típica estructura de autocodificación. x es una imagen de 572x572 pixels (bw). La codificamos en un \mathbf{h} de 32x32 pixels (base de la U). Capas residuales (skip connections) mezcladas con down-sampling y up-sampling.

Metodos de upsampling

Para disminuir las dimensiones de la imagen hacemos max-pooling o avg-pooling.

Y para aumentar la resolución?

- ▶ Interpolación de vecinos mas cercano.
- ▶ Interpolación bilineal o bicúbica.
- ▶ Transpose convolution.
Producen tablero de ajedrez.



UNETs

- ▶ Data augmentation: deformaciones elásticas a las imágenes.
- ▶ Función de pérdida: Cross entropy

$$J = \sum_{\mathbf{x}} w(\mathbf{x}) \log(p_{l(\mathbf{x})}(\mathbf{x}))$$

donde l es la etiqueta target de cada pixel.

- ▶ Usan etiquetas de los límites de las celdas. Ver contornos en la figura.
- ▶ Objetos de la misma clase pegados entre ellos. Como distinguirlos?
Mucho peso en la función de costo a la clase de límites entre celdas.

$$w(\mathbf{x}) = w_c(\mathbf{x}) + w_0 \exp \left(-\frac{d_1^2 + d_2^2}{2\sigma^2} \right)$$

donde $w_c(\mathbf{x})$ son los pesos para balancear la frecuencia de los pixels de distintas clases.

$d_1(\mathbf{x})$ y $d_2(\mathbf{x})$ son la primera y la segunda distancia del borde a la celda mas cercana y a la 2da mas cercana.

UNET: Doble convolución

```
class UNetBlock(nn.Module):
    def __init__(self, inCh,midCh,outCh,ksize): # Ch, ksize = 3):
        super().__init__()
        self.block=nn.Sequential(
            nn.Conv2d(inCh, midCh, ksize, padding = ksize // 2),
            nn.BatchNorm2d(midCh),
            nn.ReLU(True),
            nn.Conv2d(midCh, outCh, ksize, padding = ksize // 2),
            nn.BatchNorm2d(outCh),
            nn.ReLU(True) )
    def forward(self, x):
        return self.block(x)
```

UNET: Inicialización de la clase UNET

```
def __init__(self, nets): # Ch, ksize = 3):
    super().__init__()

    Ch = [nets.in_len]+nets.channels
    outCh=nets.out_len
    ksize = nets.ksize
    self.actfn=nets.actfn

    ### Left hand side of UNet
    self.encoding = nn.ModuleList()
    for i in range(len(Ch)-2):
        self.encoding.append(UNetBlock(Ch[i],Ch[i+1],Ch[i+1],ksize) )

    self.center=UNetBlock(Ch[-2], Ch[-1], Ch[-2],ksize) # bottleneck

    ### Right hand side
    self.decoding= nn.ModuleList()
    deCh=Ch[:0:-1];deCh.append(Ch[1])
    for i in range(len(Ch)-2):
        self.decoding.append(UNetBlock(deCh[i],deCh[i+1],deCh[i+2],ksize) )

    self.final      = nn.Conv2d(Ch[1], outCh,      1)
```

UNET: Código de predicción

```
def forward(self, x):

    xskip=[ ]
    for encode in self.encoding:
        x=encode(x)
        xskip.append(x)
        x=F.max_pool2d(x, 2)

    x=self.center(x)

    n=len(self.encoding)-1
    for i,decode in enumerate(self.decoding):
        x = F.interpolate(x, xskip[n-i].shape[-2:], mode="bilinear") # upscaling
        x = decode(torch.cat([x, xskip[n-i]],1)) # agregamos la capa de la rama derecha

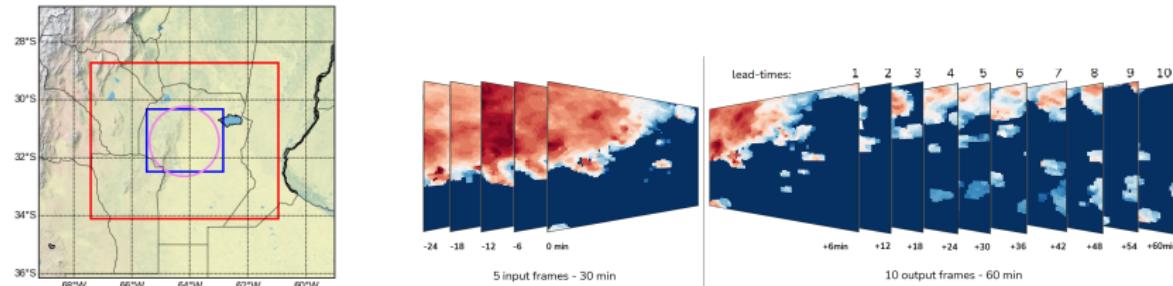
    x = self.final(x)
    return x
```

Es complicado por la necesidad de las conexiones entre ramas (skip connexions)

Predicción temporal con UNETs

Las redes del tipo UNET pueden ser utilizadas para realizar predicciones temporales.

Predicciones de tormentas usando datos de reflectividad de radar.

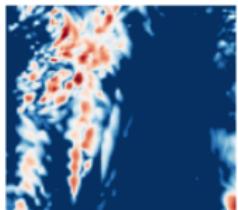


Recuadro azul corresponde al dominio de los datos

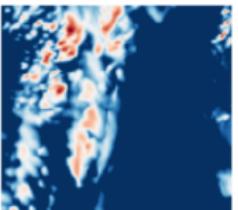
Las imágenes de entrada son $\phi(t - 2), \phi(t - 1), \phi(t)$ las predicciones $\phi(t + 1), \phi(t + 2)$.

Predictión temporal con UNETs

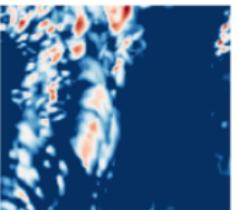
Obs. 6 min



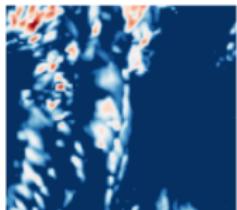
Obs. 24 min



Obs. 42 min

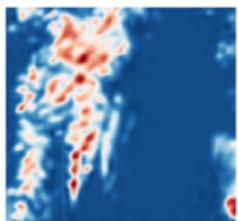


Obs. 60 min

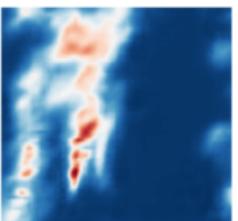


True - Target

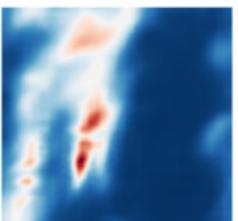
Pred. 6 min



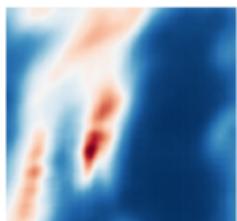
Pred. 24 min



Pred. 42 min

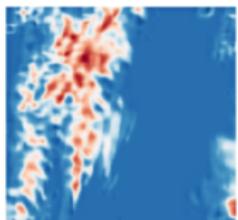


Pred. 60 min

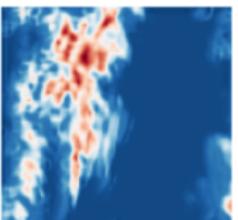


LSTM-Conv

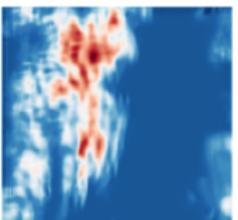
Pred. 6 min



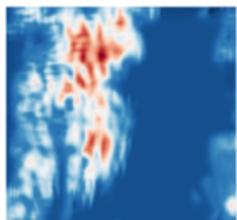
Pred. 24 min



Pred. 42 min



Pred. 60 min



UNET