

Deep Learning Specialization

by DeepLearning.AI

Course #4: Convolutional Neural Networks



[Course Site](#)

Made By: [Matias Borghi](#)

Table of Contents

Summary	3
Week 1: Foundations of Convolutional Neural Networks	5
Convolutional Neural Networks (CNN)	5
Computer Vision	5
Problemas en computer vision	5
Ejemplo de detección de bordes	6
Detección de bordes verticales	6
Más sobre detección de bordes	8
Detección de ejes horizontales	8
Padding	9
Strided Convolutions	11
Cross-correlación vs convolución	12
Convolución sobre volúmenes (o imágenes RGB)	12
Múltiples filtros	13
Resumen de la notación	15
Ejemplo de una red convolucional	16
Capas tipo Pooling	17
Max Pooling	17
Pooling promedio	17
Ejemplo red neuronal basado en el trabajo de Yann LeCunn: LeNet-5	18
Características que se notan al realizar ejemplos	18
Ejemplo entrenando una CNN con fotos de gatos	20
Week 2: Deep convolutional models	21
Case studies	21
LeNet-5	21
AlexNet	22
VGG-16	22
ResNet	23
Bloques residuales	23
Red neuronal plana (plain network)	25
¿Por qué funcionan tan bien las ResNets?	26
Ejemplo aplicado a una imagen (obtenido del paper referenciado)	27
Convoluciones 1x1	27
Usando una red de 1x1	28
Motivación para la red tipo Inception	29
Módulo de la red tipo Inception	30
Consejos prácticos sobre cómo utilizar redes convolucionales (Conv-Nets)	32
Implementar distorsiones durante el entrenamiento	35
Estado de la visión computacional	36
Datos vs ingeniería dura	36

Consejos para obtener buenos resultados en benchmarking/ganar competiciones
36

Week 3: Detección de objetos	37
Localización de objetos	37
Detección de puntos importantes (landmarks)	40
Detección de objetos	41
Algoritmo de las ventanas deslizantes	41
Implementación convolucional de las ventanas deslizantes	42
Predicciones de los cuadrados (bounding box)	44
Algoritmo YOLO (You Only Look Once)	44
Intersección sobre unión (IoU)	46
¿Cómo evitar que un objeto se detecte una vez en lugar de detectar varias veces un solo objeto?	47
Anchor boxes	47
Juntando todo lo visto en el algoritmo YOLO	48
Entrenamiento	48
Predicciones	49
Salida del algoritmo non-max suppressed	50
Región de propuestas (R-CNN, Region proposal)	50
Algoritmos más rápidos	50
Week 4: Aplicaciones especiales: Reconocimiento de caras & Neural Style transfer	52
Reconocimiento facial	52
¿Qué es reconocimiento facial?	52
Verificación facial vs Reconocimiento facial	52
One Shot Learning	53
Siamese Network	54
Triplet Loss	55
Verificación facial y clasificación binaria	57
Neural Style Transfer	58
Función de coste	59
Función de coste de contenido	60
Función de coste de estilo	61
Generalización a 1D y 3D	63

Summary

Week 1 Foundations of Convolutional Neural Networks

Learn to implement the foundational layers of CNNs (pooling, convolutions) and to stack them properly in a deep network to solve multi-class image classification problems.

Learning Objectives

- Explain the convolution operation
- Apply two different types of pooling operations
- Identify the components used in a convolutional neural network (padding, stride, filter, ...) and their purpose
- Build and train a ConvNet in TensorFlow for a classification problem

Week 2 Deep convolutional models: case studies

Learn about the practical tricks and methods used in deep CNNs straight from the research papers.

Learning Objectives

- Discuss multiple foundational papers written about convolutional neural networks
- Analyze the dimensionality reduction of a volume in a very deep network
- Implement the basic building blocks of ResNets in a deep neural network using Keras
- Train a state-of-the-art neural network for image classification
- Implement a skip connection in your network
- Clone a repository from github and use transfer learning

Week 3 Object detection

Learn how to apply your knowledge of CNNs to one of the toughest but hottest field of computer vision: Object detection.

Learning Objectives

- Describe the challenges of Object Localization, Object Detection and Landmark Finding
- Implement non-max suppression to increase accuracy
- Implement intersection over union

- Label a dataset for an object detection application
- Identify the components used for object detection (landmark, anchor, bounding box, grid, ...) and their purpose

Week 4 Special applications: Face recognition & Neural style transfer

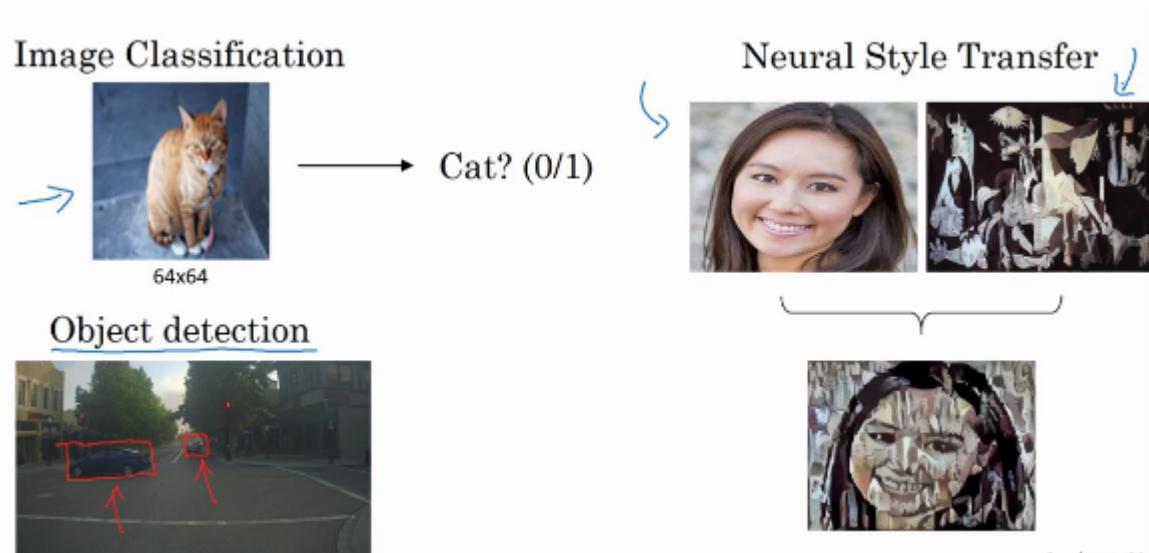
Discover how CNNs can be applied to multiple fields, including art generation and face recognition. Implement your own algorithm to generate art and recognize faces!

Week 1: Foundations of Convolutional Neural Networks

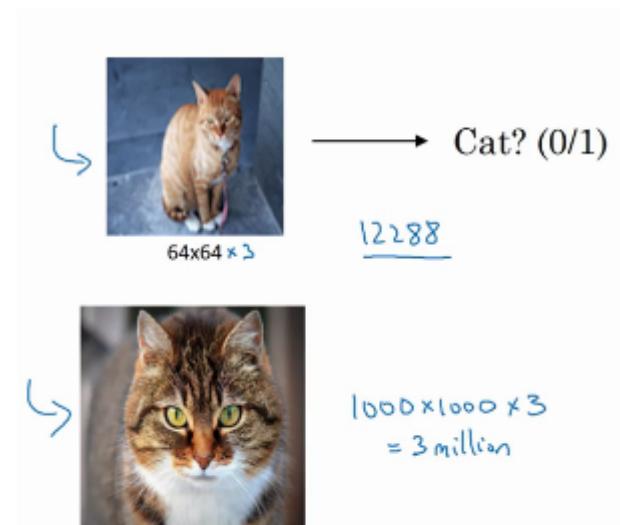
Convolutional Neural Networks (CNN)

Computer Vision

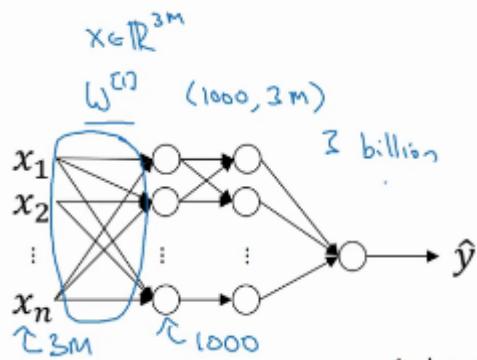
Problemas en computer vision



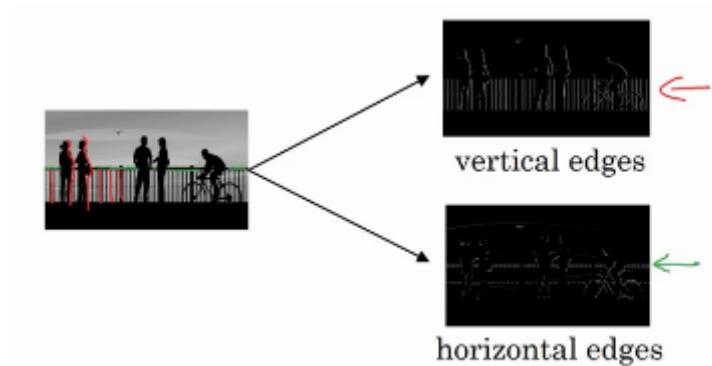
Deep Learning en imágenes grandes



Una red neuronal estándar para una imagen de alta resolución tendría alrededor de 3 mil millones (billions, en inglés) de parámetros. Es por ello que se implementará un nuevo estilo de red neuronal, la red neuronal convolucionada.



Ejemplo de detección de bordes



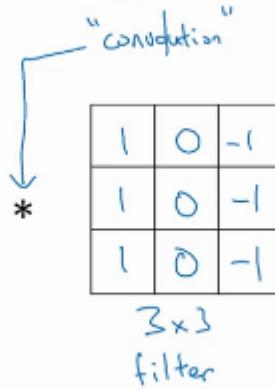
Detección de bordes verticales

Dado a modo de ejemplo el input de $6 \times 6 \times 1$ se puede convolucionar multiplicando por el filtro indicado en la imagen. ¿Cómo se aplica la convolución? Se elige el cuadrado de 3×3 y se multiplica elemento por elemento por el filtro sumando todos sus elementos, obteniendo el primer elemento como resultado. Para obtener el segundo elemento se mueve la matriz cuadrada celeste un paso a la derecha y así sucesivamente.

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 3 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6×6



=

-5			

4×4

El resultado de la convolución es entonces

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4×4

Esto en Python se implementa con `conv_forward`. En TensorFlow `tf.nn.conv2d`. En Keras `conv2d`, etc.

¿Porque esto realiza una detección de bordes verticales? Veamos un ejemplo más simple primero.

The diagram illustrates the convolution process. An input image of size 6×6 is multiplied by a 3×3 kernel. The result is a smaller output image of size 3×3 . The diagram shows the receptive fields of each output unit and the stride of the kernel.

El borde blanco indica una presencia de borde sobre el resultado. Parece grueso el borde vertical por el hecho de que la imagen es pequeña.

Más sobre detección de bordes

¿Qué sucede si la imagen inicial está invertida, es decir, si es oscura en la izquierda y brillante en la derecha?

The diagram illustrates the convolution process with an inverted input image. The input image has values 0, 10, 10, 10, 10, 10 in its first row. The result is a smaller output image where the central value is -30, indicating the filter is sensitive to dark-to-light transitions.

En este caso se ve que el filtro diferencia los bordes oscuros \rightarrow claros de claros \rightarrow oscuros. Si no se está interesado se podría tomar valor absoluto del resultado y listo.

Detección de ejes horizontales

Como es de esperarse el filtro para la detección de bordes horizontales viene dado por

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

Horizontal

Veamos un ejemplo

The diagram shows a 6x6 input image with values ranging from 0 to 10. A 3x3 kernel is applied horizontally across the image. The result is a 4x4 output image where the central element is 30, indicating a strong horizontal edge detection.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

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

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Otros ejemplos de filtros para bordes verticales

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

→

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

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

Generalizando, los filtros se pueden tomar como parámetros w_i . Detectar bordes para grados que no sean ni horizontales ni verticales.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Padding

Una modificación a la técnica de convolución que se vio previamente.

$\begin{matrix} & \\ \end{matrix} * \begin{matrix} & & \\ & & \\ & & \end{matrix} = \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \end{matrix}$

6×6
 $n \times n$

$n-f+1 \times n-f+1$
 $6-3+1=4$

Los principales **beneficios** del padding son los siguientes:

- Permite utilizar una capa convolucional sin achicar necesariamente el alto y ancho de los volúmenes. Esto es importante para construir redes neuronales profundas, dado que, de otro modo, la altura/ancho se achicaría a medida que uno se mueve hacia capas más profundas. Un caso importante especial es la convolución “same” en el cual el ancho/alto se preserva exactamente luego de una capa.
 - Ayuda a preservar la información del borde de la imagen. Sin padding, muy pocos valores en la próxima capa se verían afectados por píxeles del borde de la imagen.

Desventajas de la convolución:

- Aplicar continuamente las técnicas de convolución hace que la imagen original se reduzca.
 - Los bordes son utilizados una única vez.

Para evitar esto se agrega una borde exterior llamado padding variando el tamaño de la salida.

Considerando el ejemplo anterior, la imagen inicial pasa a ser de 8x8, mientras que la salida de 6x6.

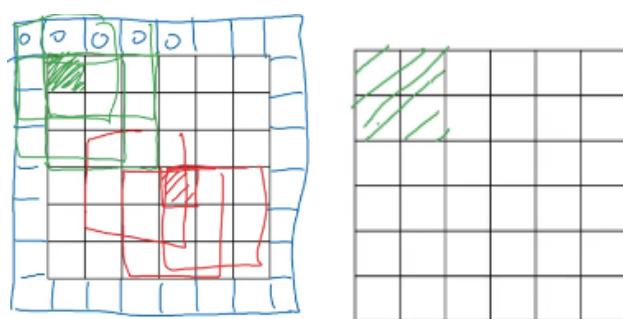


Figure 1. (izquierda) Input

Figure 2. (derecha) Output

Considerando a 'p' (o padding) como el número de capas agregadas, en este caso, $p = 1$. Mientras que ahora se satisface la ecuación:

$$\begin{aligned} n+2p-f+1 &\times n+2p-f+1 \\ 6+2-3+1 \times \underline{\quad} &= 6 \times 6 \end{aligned}$$

¿Cuántas capas conviene agregar? Convoluciones del tipo Valid y Same:

- Valid: no hay padding.
- Same: padding tal que el tamaño de salida sea el mismo que el tamaño de entrada.

Esto es

$$\begin{aligned} n+2p-f+1 &\times n+2p-f+1 \\ p+2p-f+1 = p &\Rightarrow p = \frac{f-1}{2} \end{aligned}$$

El tamaño de los filtros generalmente es impar.

Strided Convolutions

Empecemos con un ejemplo

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ \hline 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ \hline 3^3 & 4^4 & 8^4 & 3 & 8 & 9 & 7 \\ \hline 7^1 & 8^0 & 3^2 & 6 & 6 & 3 & 4 \\ \hline 4^{-1} & 2^0 & 1^3 & 8 & 3 & 4 & 6 \\ \hline 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ \hline 0 & 1 & 3 & 9 & 2 & 1 & 4 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|} \hline 3 & 4 & 4 \\ \hline 1 & 0 & 2 \\ \hline -1 & 0 & 3 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|} \hline 91 & 100 & 83 \\ \hline 69 & & \\ \hline & & \\ \hline \end{array}$$

3×3
stride = 2

Esto quiere decir que dado un stride (paso) = 2, para calcular los siguientes elementos el bloque celeste se mueve de a 2 pasos en lugar de 1.

Summary of convolutions

$n \times n$ image $f \times f$ filter

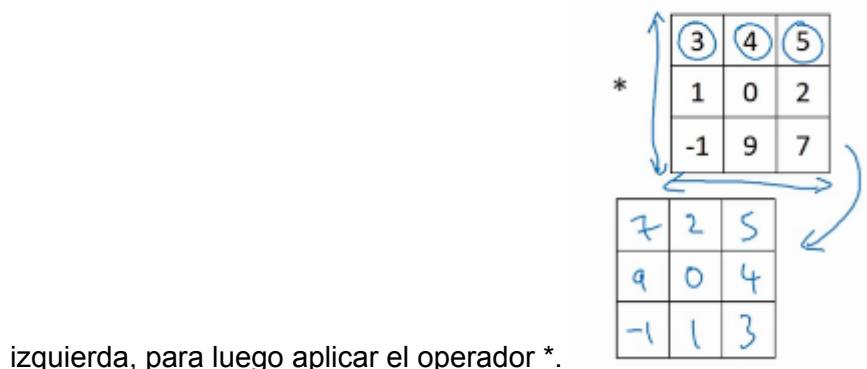
padding p stride s

$$\left[\frac{n+2p-f}{s} + 1 \right] \quad \times \quad \underbrace{\left[\frac{n+2p-f}{s} + 1 \right]}_{\text{ }}$$

$$\lfloor z \rfloor = \text{floor}(z)$$

Cross-correlación vs convolución

Resulta que en la literatura de Machine Learning se suele decir que la operación recientemente explicada y realizada es llamada convolución, pero realmente es una cross-correlación. En los libros de matemática una convolución viene dada por una operación previa que viene de espejar el filtro de la manera mostrada en la figura de la

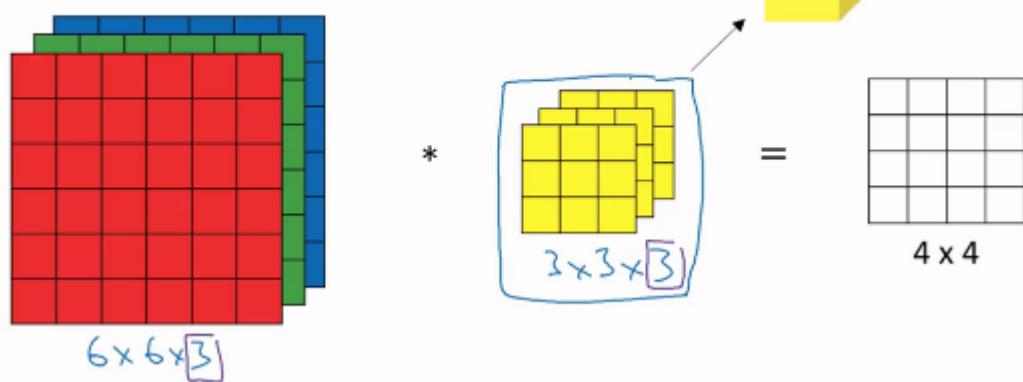


izquierda, para luego aplicar el operador $*$.

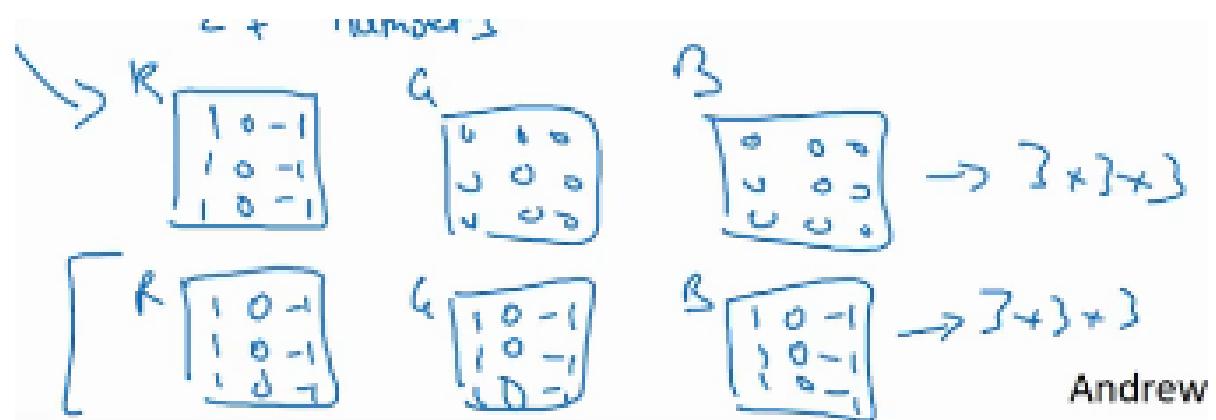
Convolución sobre volúmenes (o imágenes RGB)

El canal correspondiente al tercer número en el input y filtro deben coincidir.

Convolutions on RGB image



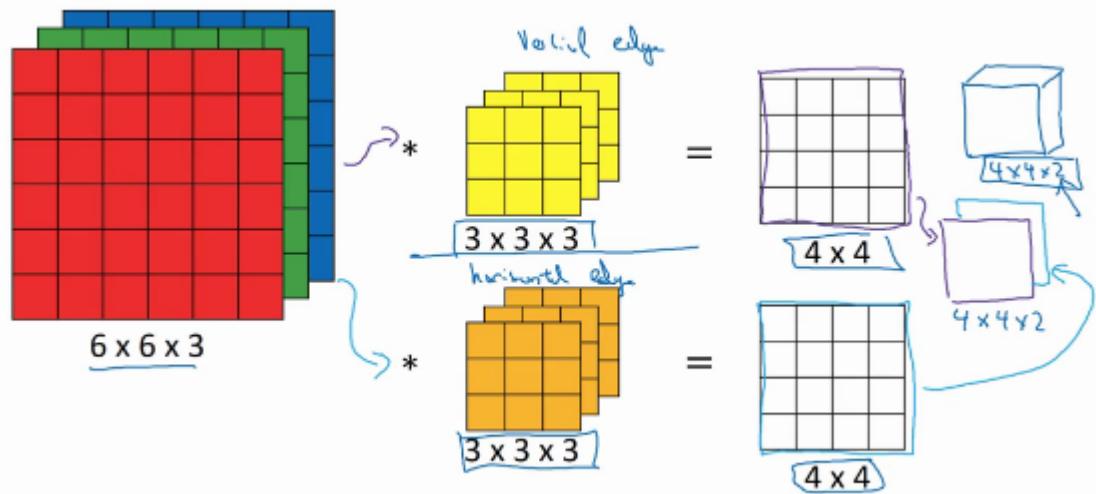
La elección del filtro puede ser la siguiente



donde la primera busca bordes únicamente para el color rojo, mientras que la segunda para cualquier color.

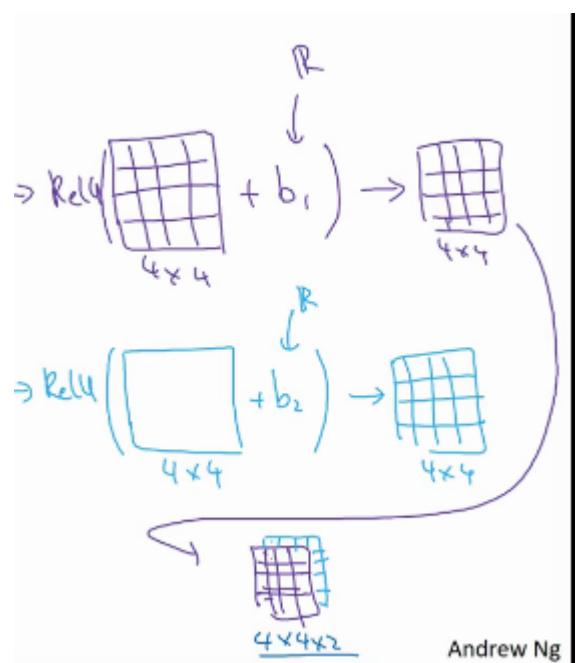
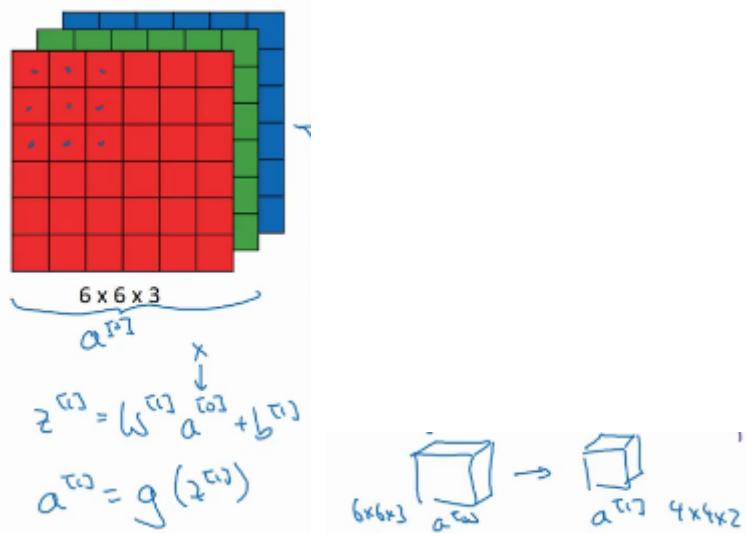
Múltiples filtros

También se pueden aplicar múltiples filtros por separado. Por ejemplo, para bordes verticales y otro para horizontales y juntar los resultados de ambos como muestra la figura.



Una capa de red convolucional

Claramente, la matriz de 4×4 juega el rol de w^*a



Andrew Ng

Resumen de la notación

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l]}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$.

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l]} \times n_c^{[l]}$

bias: $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$ ← #filters in layer l .

Input: $\frac{n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}}{n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}}$

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$$A^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$$

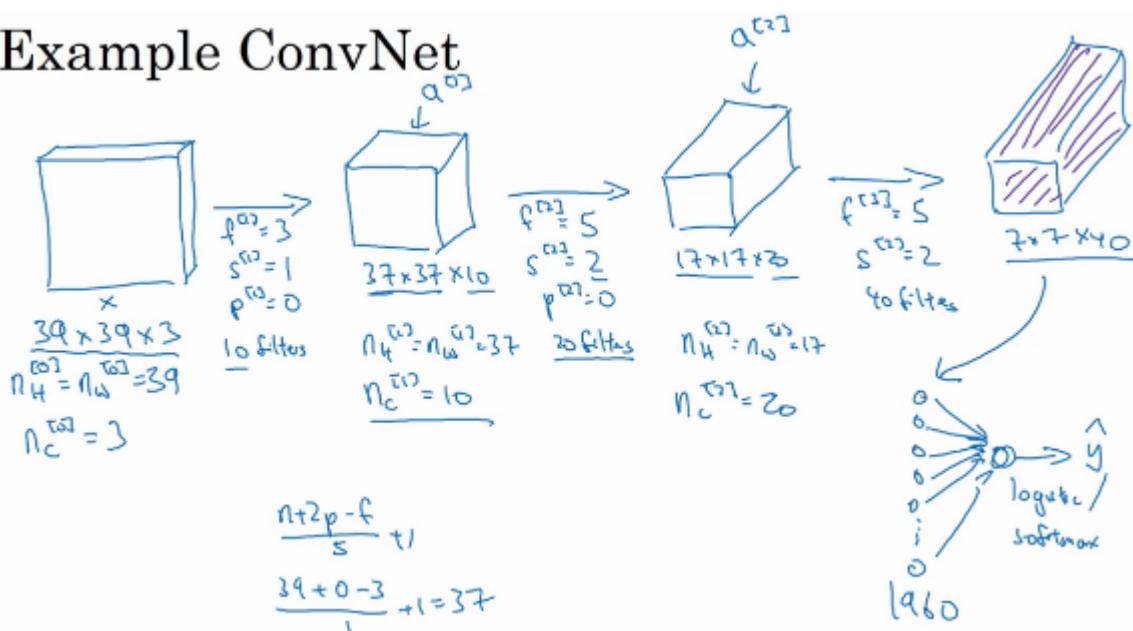
Ejemplo de una red convolucional

Veamos cómo aplicar una red convolucional en varias capas de redes neuronales.

Los parámetros correspondientes al tamaño y número de los filtros, stride y padding los decidimos nosotros previa a cada capa. Los resultados se obtienen con las ecuaciones mencionadas arriba. Por ejemplo, para la primera capa el tamaño de la imagen viene dado por $(39 + 0 - 3) / 1 + 1 = 37$.

Uno de los mayores desafíos en CNN es elegir estos hiper parámetros.

Example ConvNet



Tipos de capas en redes convolucionales:

- Convolución (CONV)

- Pooling (POOL)
- Completamente conectada (FC)

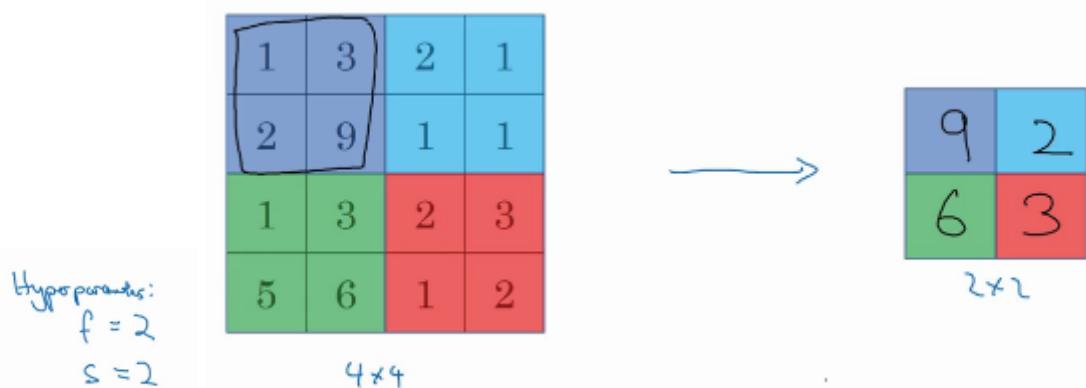
Esto se realiza para mejorar el tiempo de implementación de las CNN principalmente.

Capas tipo Pooling

Max Pooling

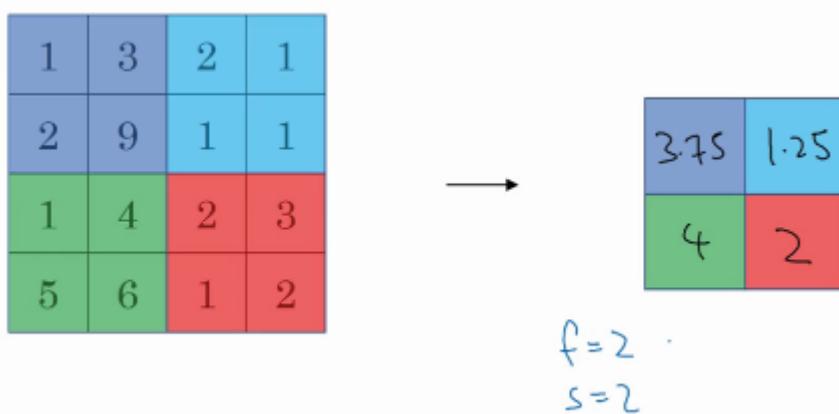
Tomando el máximo de cada cuadrado coloreado.

Hiper parámetros correspondientes a la técnica de max pooling.



Pooling promedio

Al igual que lo anterior se toma el promedio de cada cuadrado coloreado.



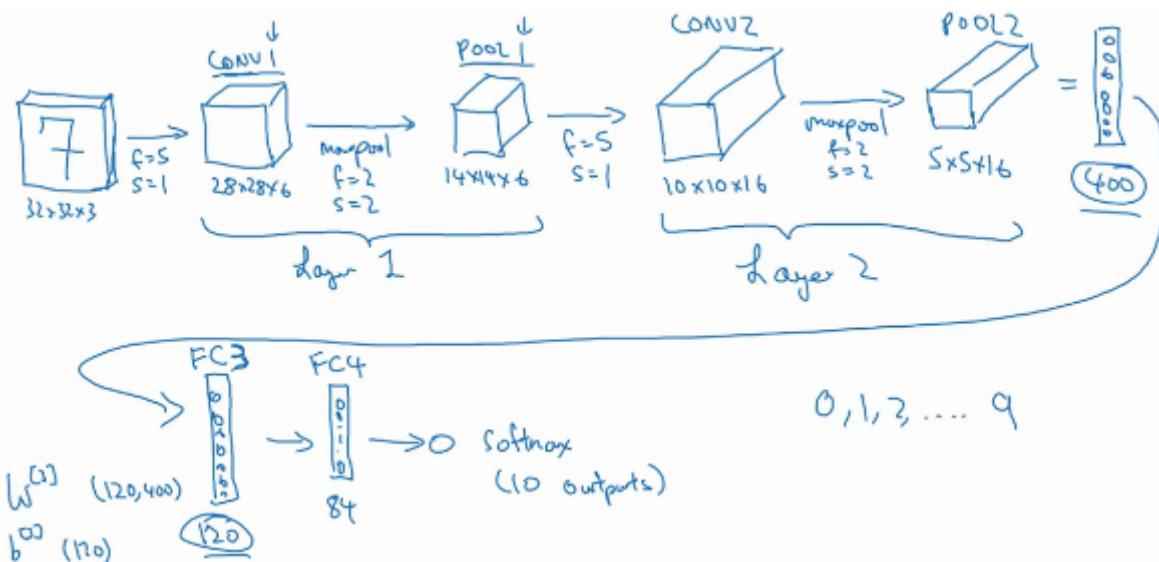
En redes muy profundas suele ser mas común utilizar esta última técnica antes que el pooling máximo.

El tema con pooling es que no hay parámetros que aprender en backpropagation.

Ejemplo red neuronal basado en el trabajo de Yann LeCunn: LeNet-5

Una convención es decir que una capa es como la que figura la imagen. Otros definen a las capas tipo pooling como una capa de por si, pero no se suele hacer por el hecho de que no poseen hiper parámetros.

La capa completamente conectada utilizada en la red neuronal es simplemente una capa utilizada como se hizo previamente donde los 400 inputs se conectan completamente con los 120, de aquí que los parámetros w y b tengan los tamaños que se muestran en la figura.



Características que se notan al realizar ejemplos

El tamaño de las imágenes tiende a disminuir al aumentar el número de capas, mientras

$$\begin{array}{l} n_h, n_w \downarrow \\ n_c \uparrow \end{array}$$

que el número de filtros tiende a aumentar.

Esquema de la red neuronal aplicada a CNN

CONV - POOL - CONV - POOL - FC - FC - FC - SOFTMAX

Algunos temas importantes que se notan mirando la tabla inferior y el ejemplo previo:

- El número de hiper parámetros de las capas tipo pooling es cero, mientras que de las capas convolucionales es bastante pequeño comparados con las FC.
- El tamaño de las funciones de activación disminuyen a medida que nos movemos a capas más profundas.

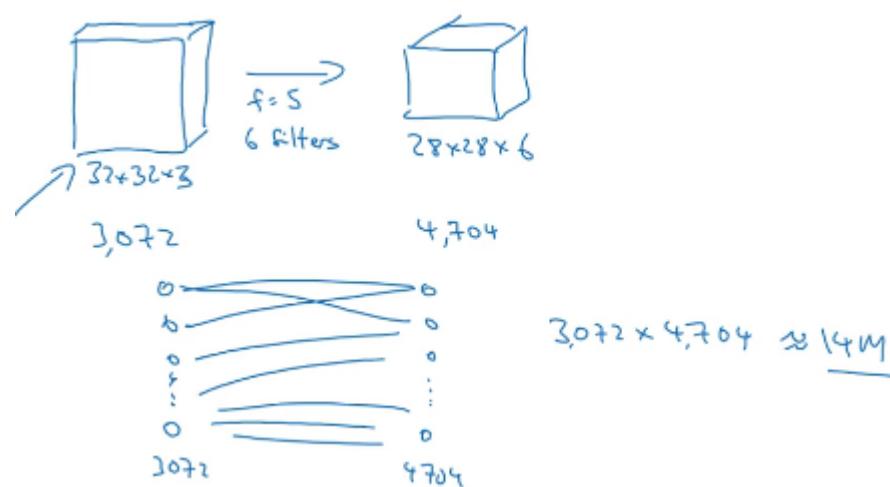
	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{32 \times 3}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 ←
POOL1	(14,14,8)	1,568	0 ←
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 ←
POOL2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48,001 {
FC4	(84,1)	84	10,081 {
Softmax	(10,1)	10	841

¿Por qué usar capas convolucionales en lugar de únicamente FC?

Ventajas por sobre utilizar únicamente las redes totalmente conectadas (FC).

- Parámetros compartidos
- Sparsity de las conexiones.

Veamos esto con un ejemplo, como se dijo previamente, conectar estas dos imágenes requeriría entrenar en una capa 14M de parámetros. A diferencia en una red convolucional es 156 considerando un filtro de 5 ($5 \times 5 \times 6 = 156$), mucho menor.



Parámetros compartidos: una característica a detectar sobre que una parte útil de una imagen es probablemente útil en otra parte de la imagen.

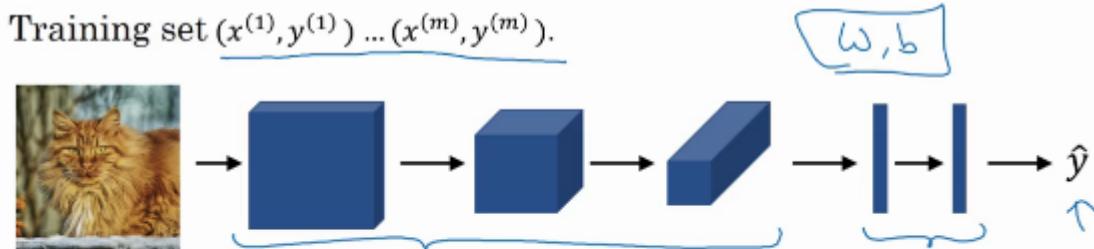
$$\begin{array}{|c|c|c|c|c|c|} \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline \end{array}$$

Sparcity de las conexiones: En cada capa, cada valor de salida depende únicamente de un número pequeño de entradas. Ej., el valor 0 del extremo izquierdo de la salida depende del cuadro superior izquierdo de 3x3 de la entrada con valor 10s.

Ejemplo entrenando una CNN con fotos de gatos

x: imágenes de felinos

y: etiqueta identificando gatos o no.



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce J .

Supongamos que se eligió una estructura para la CNN empezando por una secuencia de CONV-POOL seguida por FC y SOFTMAX al final.

Con esto se calculará una función de coste como figura en la ecuación superior y luego se usará gradiente descendiente para optimizar los parámetros y reducir esta función.

Week 2: Deep convolutional models

Case studies

Outline

Classic networks:

- LeNet-5 [←](#)
- AlexNet [←](#)
- VGG [←](#)

ResNet [\(152\)](#)

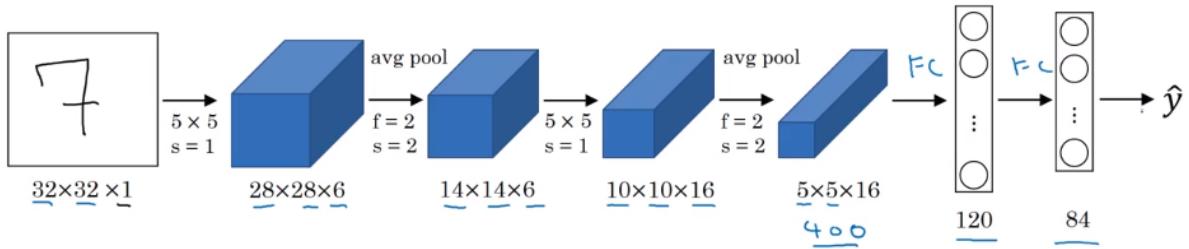
Inception

Estudiar otros ejemplos son útiles para entender muchos conceptos de CNN. Los temas a abordar incluyen CNNs clásicas como LeNet-5, AlexNet y VGG, como así también ResNet una red neuronal muy profunda de 152 capas y las redes del tipo Inception.

LeNet-5

La arquitectura de esta red se esquematiza en la imagen siguiente. Obtenido del documento: [[Le Cun et al., 1998. Gradient-based learning applied to document recognition](#)] (si se quiere leer mirar principalmente secciones II y III).

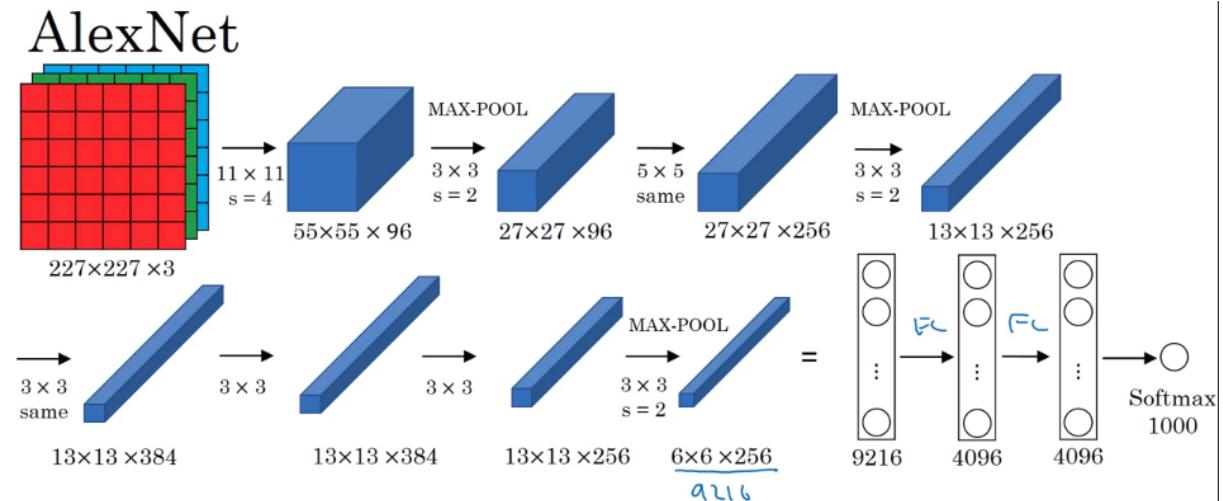
Inicialmente la entrada (imágenes en escala de grises) de tamaño 32x32x1 se procesa mediante 6 filtros de 5x5 con stride = 1 y no padding. Es por ello que la salida es una red de 28x28x6. Luego se aplica a pooling. Pooling promedio era utilizado más en 1998, ahora no tanto. Luego se aplica otra capa de red convolucional y un pooling promedio. Luego dos redes completamente conectadas y por ultimo un último nodo que puede tomar 10 valores, uno para cada una de las posibles soluciones: 0,1, 2, ...,9. Una versión moderna usaría Softmax al final.



Esta red posee alrededor de 60k parámetros. A día de hoy se encuentran redes con más de 10M.

AlexNet

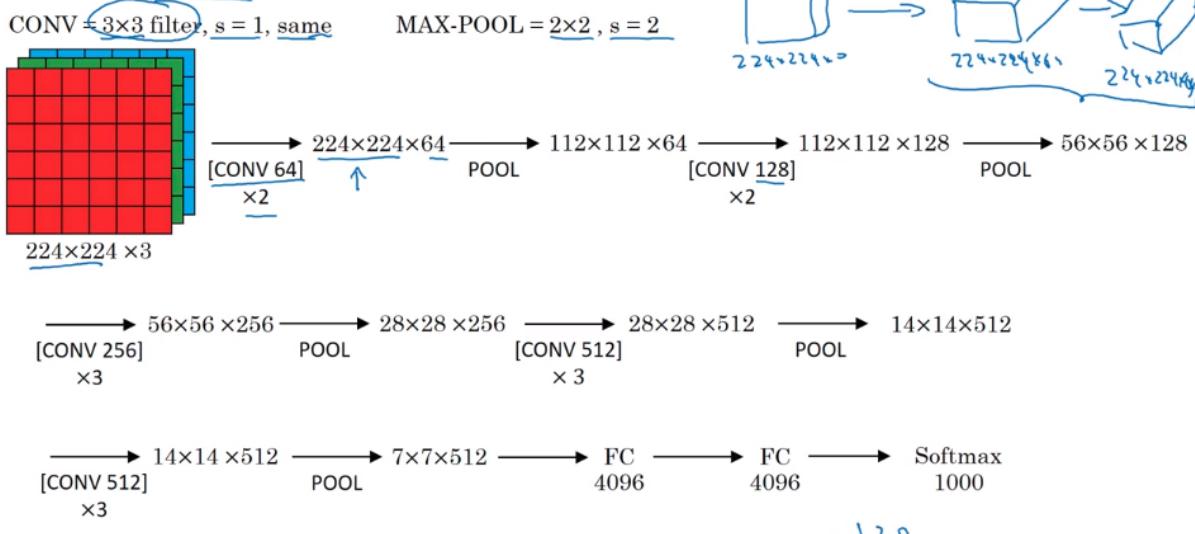
Esta red está basada en el siguiente documento: [[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks](#)]. La misma tiene una arquitectura similar a la anterior, pero es mucho más grande, tiene alrededor de 60M de parámetros.



VGG-16

Esta red está basada en el siguiente documento: [[Simonyan & Zisserman, 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition](#)]. Posee 138M de parámetros. Se llama VGG-16 por el número de capas que posee, 16.

VGG - 16



Veamos ahora algunas redes más poderosas y más importantes.

ResNet

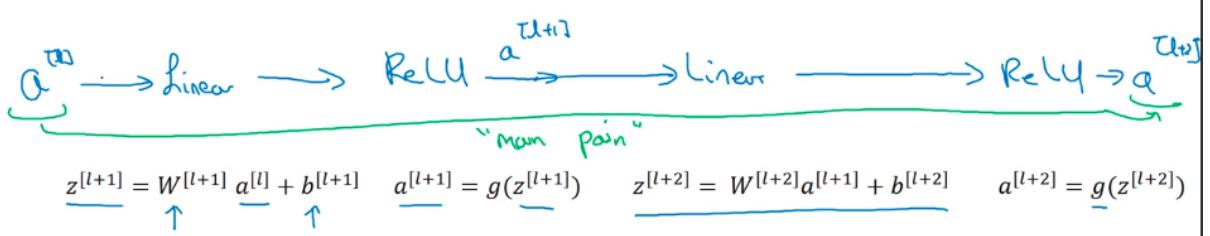
Vanishing and exploding gradients son un problema con las redes neuronales muy profundas. Para entrenar redes con hasta 100 capas se va a explicar un concepto en el que la activación de una capa y alimentarla en una capa más profunda que la primera.

Estas redes están basadas en el paper: [[Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.](#)].

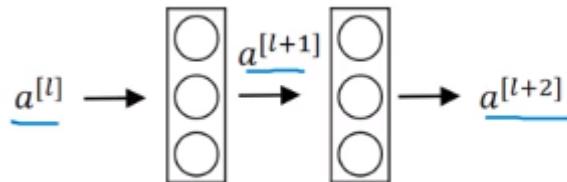
Bloques residuales

Dado una función de activación de orden l se puede obtener una de orden $l+2$ pasando por dos capas de la siguiente manera:

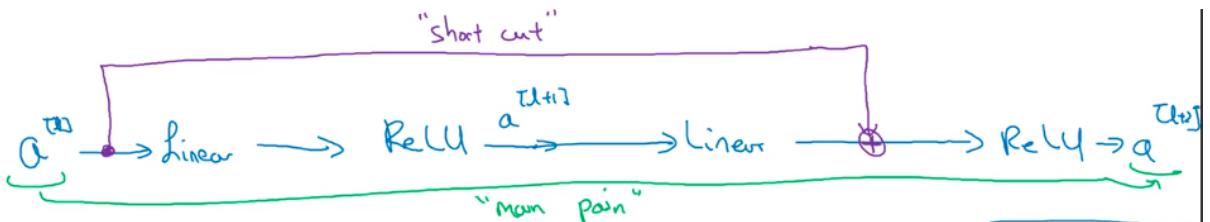
Donde primero se aplica una función lineal para obtener $Z^{[l+1]}$. Luego se aplica una función no lineal como ReLU obteniendo la activación $a^{[l+1]}$. De manera análoga se obtiene $a^{[l+2]}$. A esto se lo llama “main path”.



En redes residuales se hace un cambio, esto es, tomar $a^{[l]}$ y sumarlo antes de calcular la función no lineal ReLU para obtener $a^{[l+2]}$, como muestra la figura llamado “short cut” (skip connection o atajo).



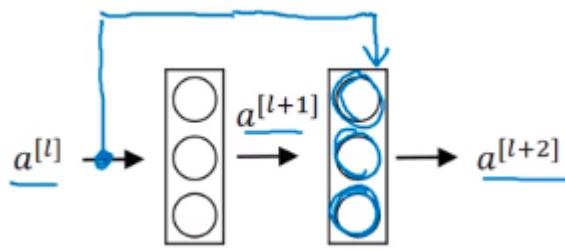
La información de $a^{[l]}$ va más adentro de la red neuronal.



De esta manera la función de activación es calculada como

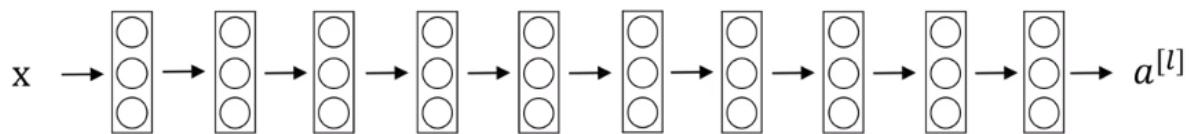
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

EL diagrama de esta nueva red es entonces

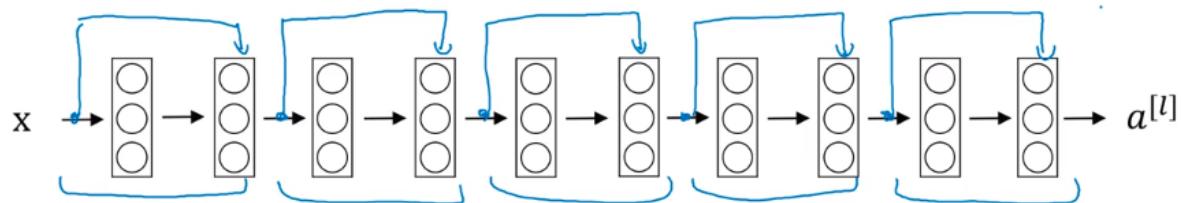


Las Resnets agarran varios de estos bloques residuales para formar una red con varias capas.

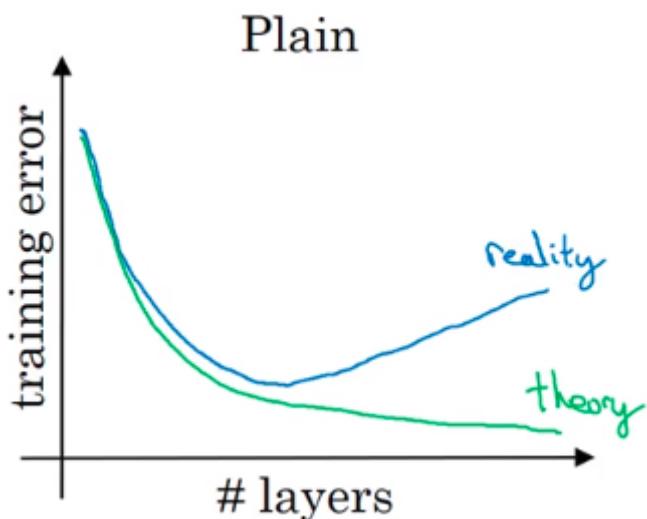
Red neuronal plana (plain network)



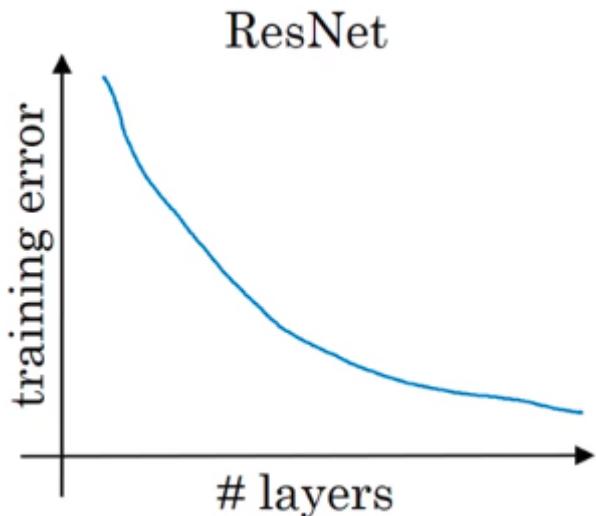
Para convertirla en una red residual agregamos los bloques residuales de la siguiente manera



Veamos ahora cómo se comporta una red neuronal plana a medida que el número de capas aumenta. En teoría el error de entrenamiento debería decrecer, pero llega un punto en el que aumenta.



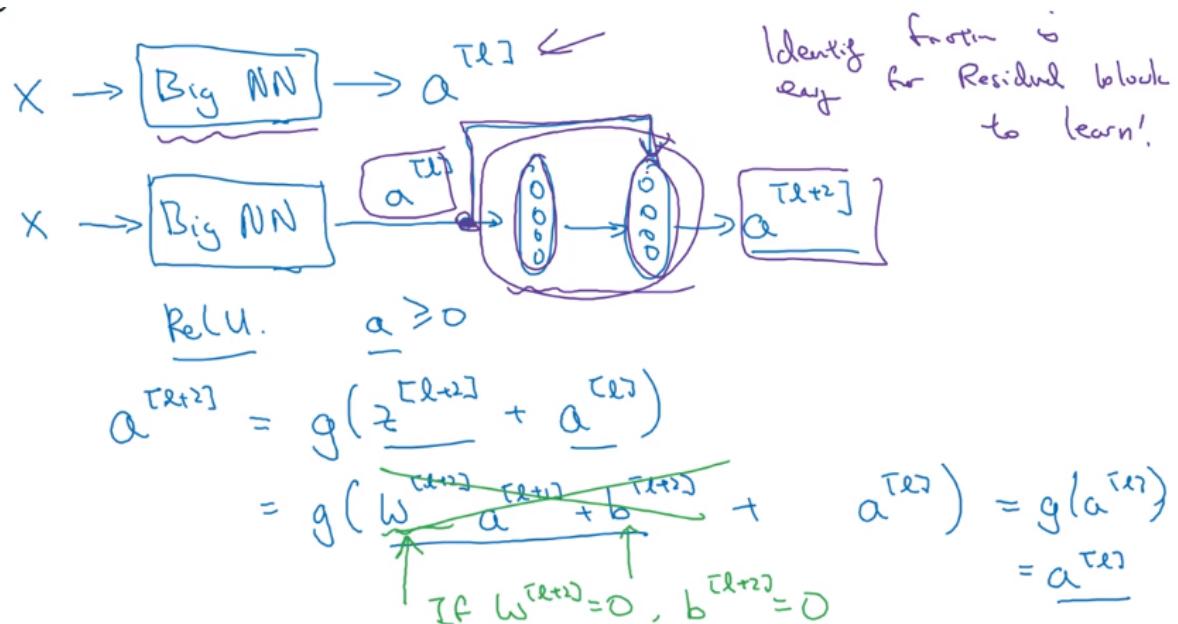
En cambio, con las Resnets se pueden entrenar redes más profundas dado que el error de entrenamiento disminuye al incluir bloques residuales.



¿Por qué funcionan tan bien las ResNets?

Funcionar bien en sets de entrenamiento es un prerequisito para que funcione correctamente en dev o test sets.

En la primera figura se considera el caso de dado un input X , una gran red neuronal actúa sobre la misma y se obtiene una función de activación $a^{[l]}$. Si se agrega un bloque residual, se puede ver que la función identidad es fácil de aprender a pesar de tener más capas que la red anterior.



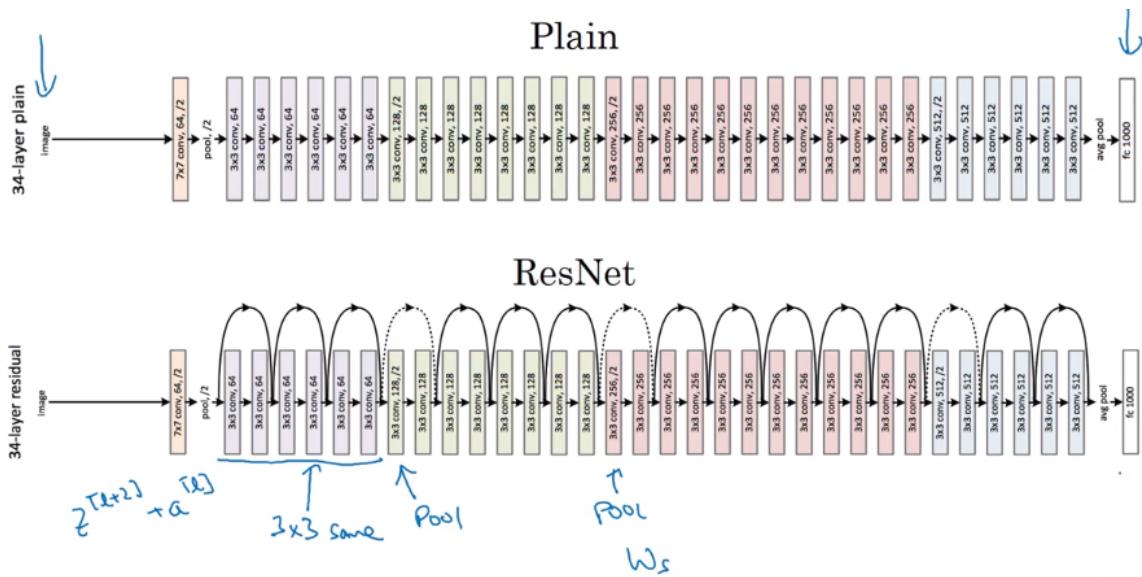
Suposición importante: Tanto $z^{[l+2]}$ como $a^{[l]}$ poseen la misma dimensión. Para lograr esto es por ello que se usan convoluciones del tipo same. En el caso de que posean diferentes

dimensiones, por ejemplo, $a^{[l+2]}$ es de dimensión 256 y $a^{[l]}$ de dimensión 128, se agrega una matriz w que multiplica a esta última de dimensión 256x128. Esta puede ser una matriz de parámetros o una matriz que implementa padding igual a cero.

$$\begin{aligned}
 & a^{[l+2]} = R^{256 \times 128} \\
 & a^{[l+2]} = W_s \cdot \frac{a^{[l]}}{128} \\
 & a^{[l+2]} = 0
 \end{aligned}$$

Ejemplo aplicado a una imagen (obtenido del paper referenciado)

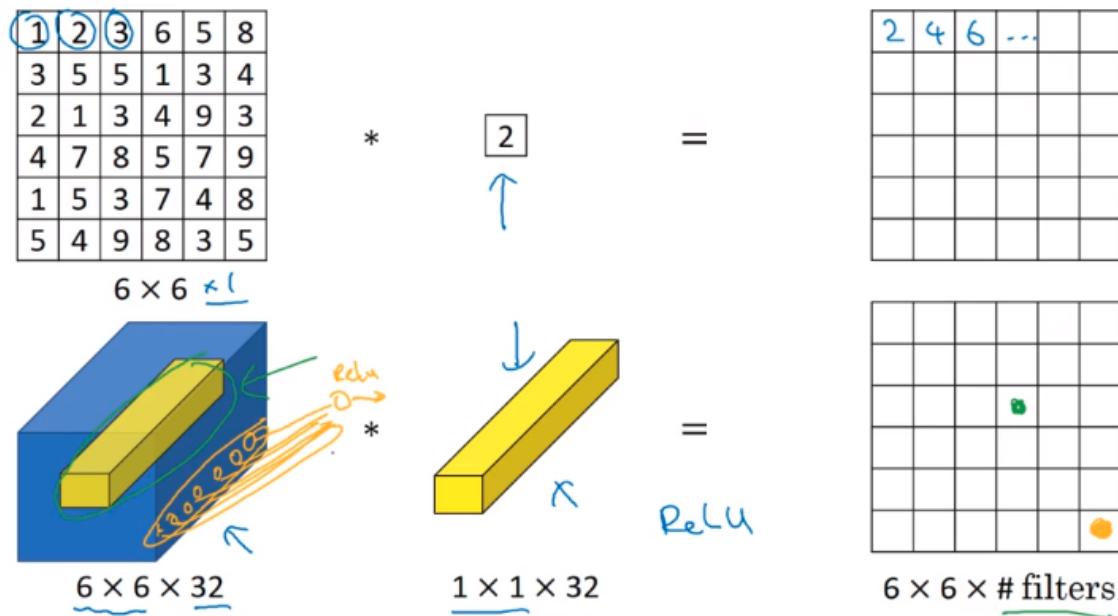
En el primer caso se ve una red neuronal plana de 34 capas que utiliza CNNs con same padding. Para que en la ResNet al aplicar la red se mantienen las dimensiones entre $z^{[l+2]}$ y $a^{[l]}$. Pero también hay incluidas capas tipo pooling por lo que ajustes son necesarios mediante la matriz m .



Convoluciones 1x1

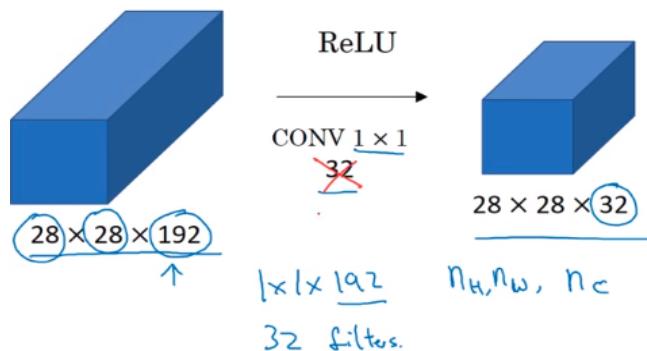
¿Qué es lo que hacen las convoluciones 1x1? Para el caso de una entrada de 6x6x1, multiplicarla por un filtro de tamaño 1x1x1 parece trivial como multiplicar un escalar. Pero si

la entrada posee otros valores la cosa cambia. Por ejemplo, si es de $6 \times 6 \times 32$, multiplicar por un filtro de $1 \times 1 \times 32$ provoca que se obtenga un resultado de dimensión $6 \times 6 \times 32$. Esto es conocido también como Network in Network, nombrado en el paper: [[Lin et al., 2013. Network in network](#)]. Este método ha influenciado otras redes como las del tipo Inception.



Usando una red de 1×1

¿Cuándo son útiles? Nos permite mantener fijo el número de filas y columnas y cambiar únicamente el número de filtros, tanto aumentarlo como disminuirlos. En el caso de la imagen se disminuye la red hasta una con 32 filtros a la salida de la red convolucional. Por otra parte, permite construir redes tipo Inception las cuales veremos a continuación.

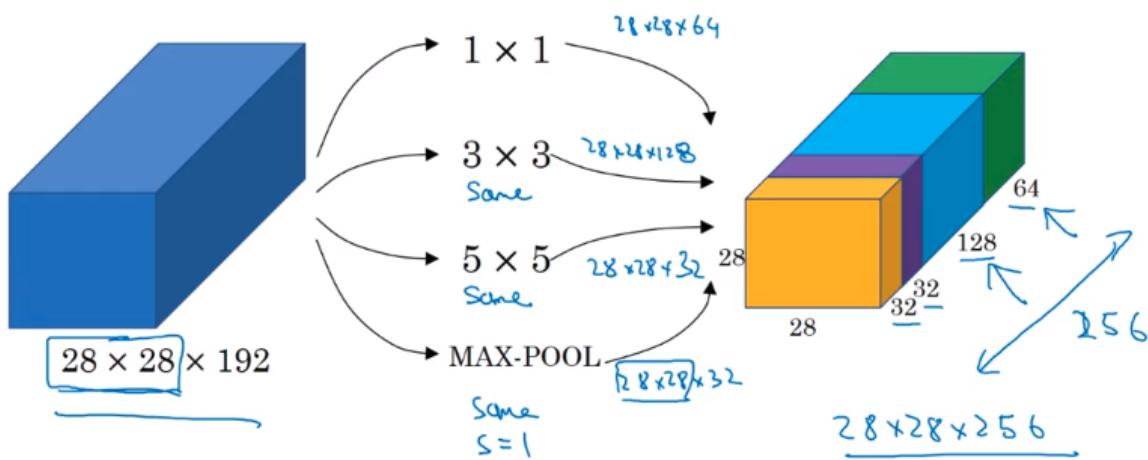


Redes tipo Inception

Al momento de elegir una capa CNN se tiene que elegir de que tamaño se quiere utilizar un filtro o si se usa una capa tipo pooling. Lo que dice la red tipo Inception es por qué no se utilizan todas. Esto hace que la arquitectura sea más complicada pero funciona mucho mejor. Obtenido del paper: [Szegedy, Christian, et al. "Going deeper with convolutions." Cvpr, 2015.]

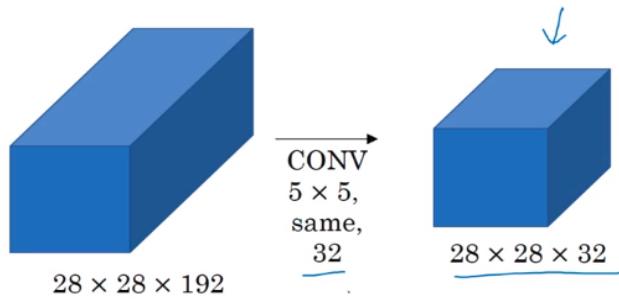
Motivación para la red tipo Inception

Básicamente como se dijo anteriormente, dada una entrada de tamaño $28 \times 28 \times 192$ se aplicara una convolución de, digamos, 1×1 , el resultado será de $28 \times 28 \times 64$, pero tal vez querríamos aplicar una de 3×3 con convolución tipo same para que el tamaño sea el mismo, el resultado sería $28 \times 28 \times 128$. De la misma manera con una capa de 5×5 . Por último, si se aplica max pooling de tamaño $28 \times 28 \times 32$, para que coincidan los tamaños hay que elegir same padding y stride = 1. El resultado total es un volumen concatenado de $28 \times 28 \times 256$.

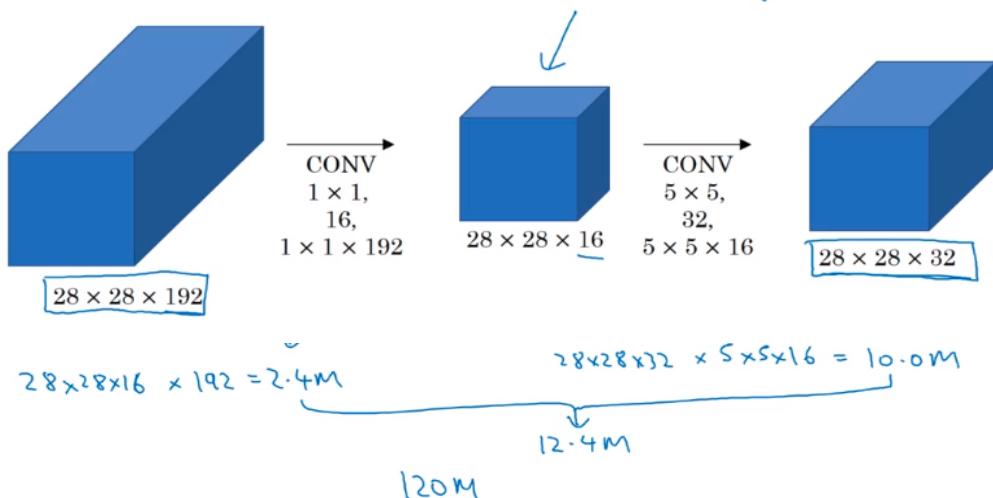


Hay un problema con todo esto con el coste computacional de la red. Veamos, por ejemplo, la de 5×5 .

EL tamaño de los filtros viene dado por $5 \times 5 \times 192$. Mientras que el número de filtros es de $28 \times 28 \times 32$ de esta manera el número de operaciones es $120M \rightarrow$ Operación muy extensa. Esto se puede reducir en un factor de 10.

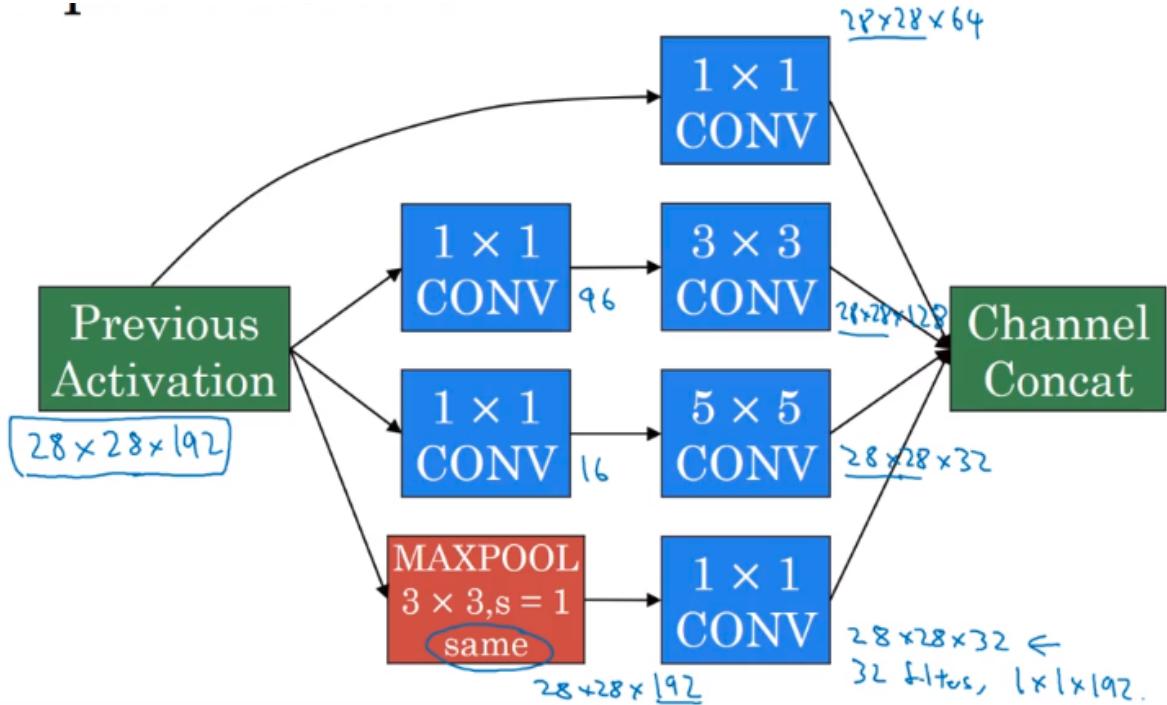


Esto se soluciona agregando una red convolucional de 1×1 luego de la entrada antes de aplicar la red de 5×5 . El tamaño del input cambia a $28 \times 28 \times 16$, siendo este último sobre el que se aplica la red de 5×5 , el tamaño de salida sigue siendo el mismo. Esta capa de 1×1 se llama capa cuello de botella (bottleneck layer). El costo computacional de esta red es mucho menor, la primera capa realiza $28 \times 28 \times 16 \times 192 \sim 2.4$ M de cálculos, mientras que la segunda $28 \times 28 \times 32 \times 5 \times 5 \times 16 \sim 10$ M, lo que es igual a 12.4 M << 120M.

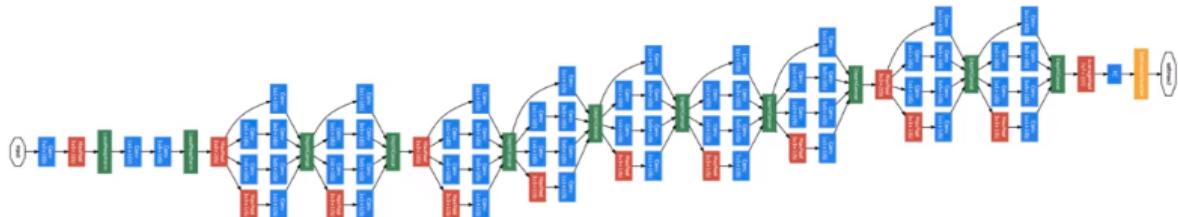


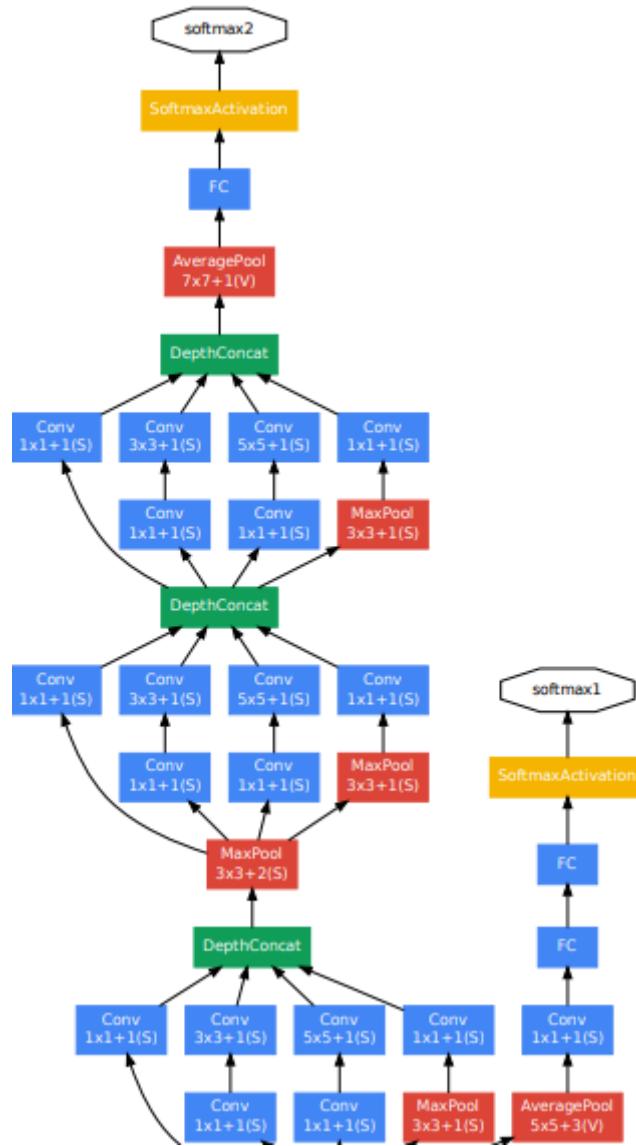
Módulo de la red tipo Inception

Haciendo esto mismo para todas las convoluciones se obtiene el siguiente módulo: para la capa tipo pooling conviene adjuntarle 32 filtros de tamaño $1 \times 1 \times 32$ para que la salida sea de $28 \times 28 \times 32$ como queríamos. Concatenando todos estos resultados se obtiene el valor deseado de $28 \times 28 \times 256$, habiendo realizado muchas menos operaciones que si se hubieran aplicado las convoluciones de 5×5 o 3×3 directamente a la entrada.



Entonces, ¿qué es una red tipo Inception? Es unir estos módulos. También se muestra con un poco de zoom. Puede notarse que el módulo de Inception que se describió anteriormente está repetido continuamente en la red. Una sutil diferencia con el esquema de la red mostrada con aumento es que tiene adicionalmente adjuntada unas capas para realizar predicciones con dos capas completamente conectadas y un Softmax.





Consejos prácticos sobre cómo utilizar redes convolucionales (Conv-Nets)

- Uso de implementaciones en código libre

Implementar código libre para hacer más fácil la replicación de resultados.

También se puede obtener código libre en GitHub de una arquitectura o tema que nos interese, por ejemplo

```
$ git clone https://github.com/tensorflow/models.git
```

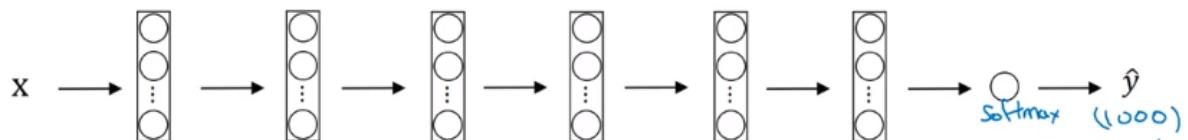
- Transferencia de aprendizaje (Transfer learning)

En lugar de hacer que una red aprenda los parámetros de la misma, se pueden obtener los resultados de alguien más y utilizarlos para realizar nuevas predicciones.

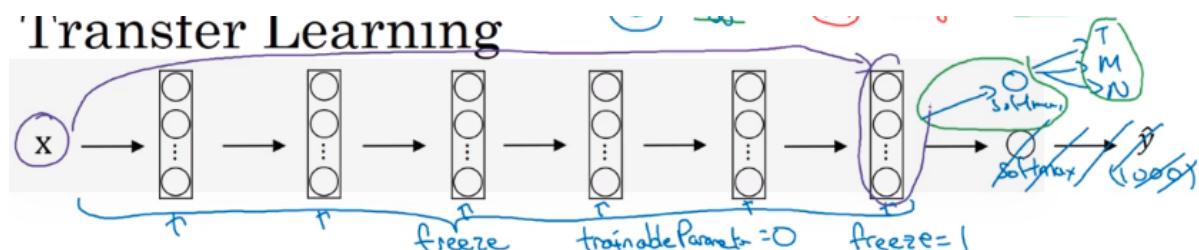
Veamos un ejemplo: si se quiere implementar una red para identificar si un gato es Tiger, Misty o ninguno de los dos (ignorando las fotos en las que aparezcan ambos). Dado que el set de datos seguramente sea pequeño, ¿qué se puede hacer?



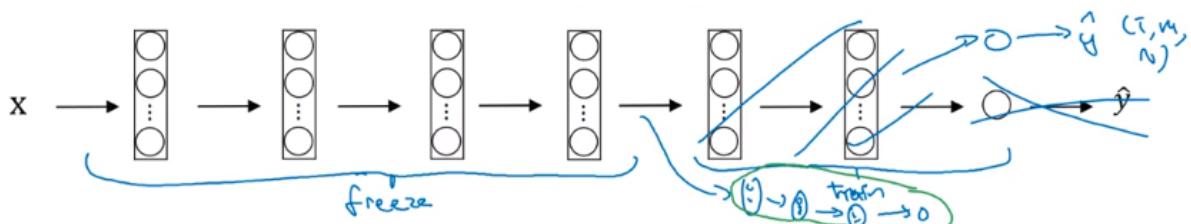
Para ello conviene conseguir una red neuronal open source que ya haya sido entrenada (source code y los pesos), por ejemplo una red que haya sido entrada sobre el ImageNet data set que permite identificar hasta 1000 objetos.



Para hacer esto conviene eliminar la salida del Softmax y agregar las nuestras (en nuestro caso T,M,N). Algunas redes tienen parámetros que permiten que los pesos de las capas anteriores no sean entrenadas sino que mantengan los valores entrenados con ImageNet. Estos parámetros suelen ser denotados como trainableParameter=0 o freeze=1.



¿Qué sucede si se tiene una cantidad de datos bastante grande que puede entrenar una red de algunas capas? En este caso conviene deshacernos de una porción de la red y agregar nuestras propias capas ocultas y Softmax y entrenarlos con nuestros datos.



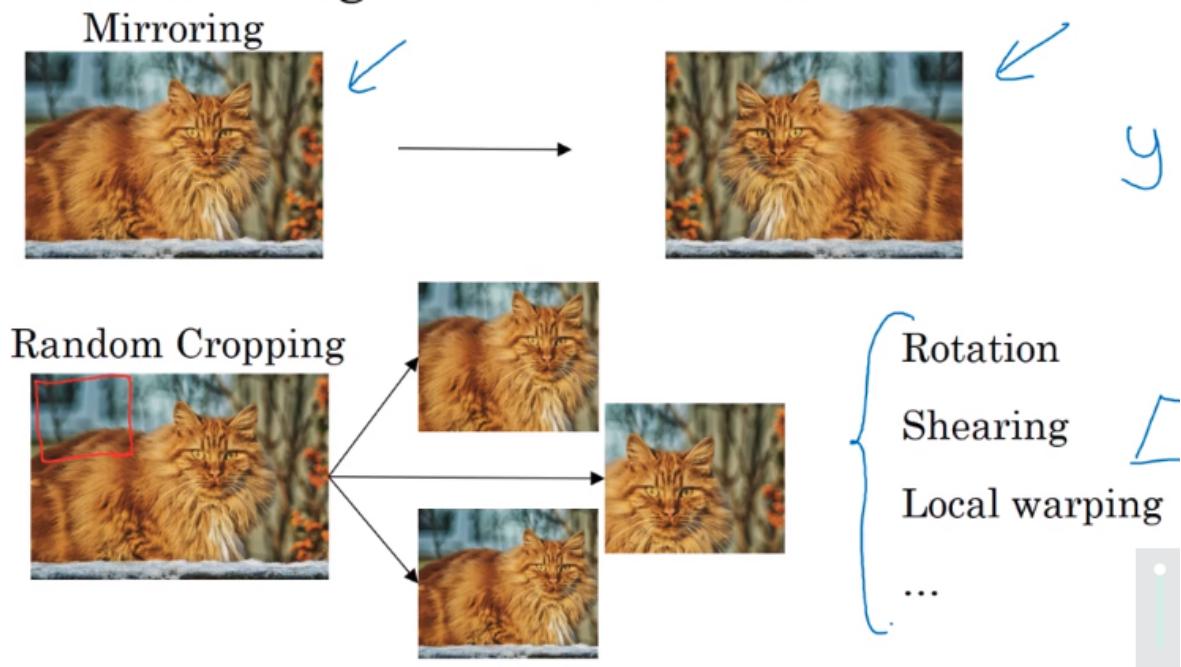
Por último, como caso límite, si se poseen suficientes datos, se puede entrenar la red completa pero, en lugar de inicializar con números aleatorios los parámetros, se realiza con

los parámetros descargados previamente y luego realizar gradiente descendiente para actualizar estos parámetros.

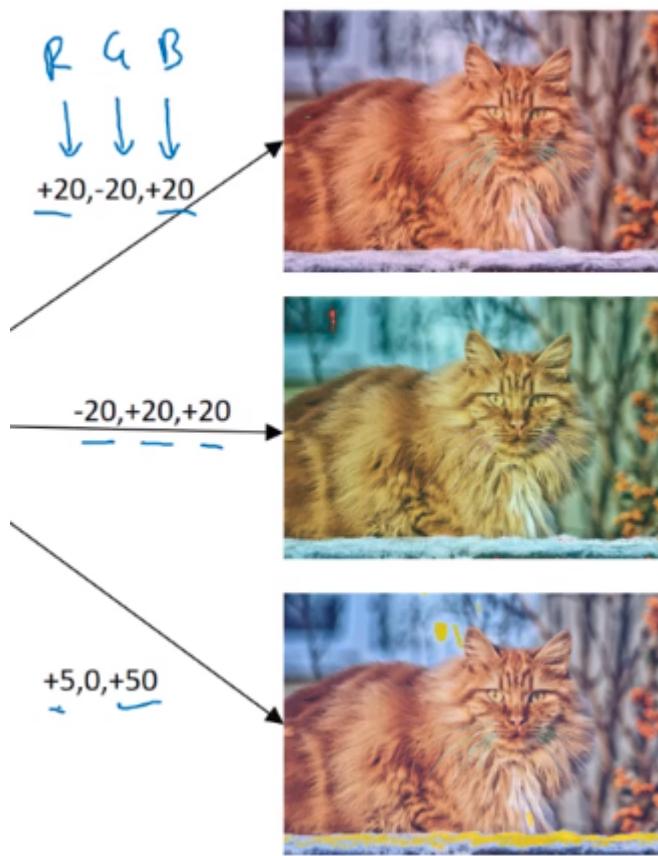
- Aumento de datos (data augmentation)

Espejado: será una buena técnica si preserva la identificación del objeto.

Cortado aleatorio, rotación, achique, etc.

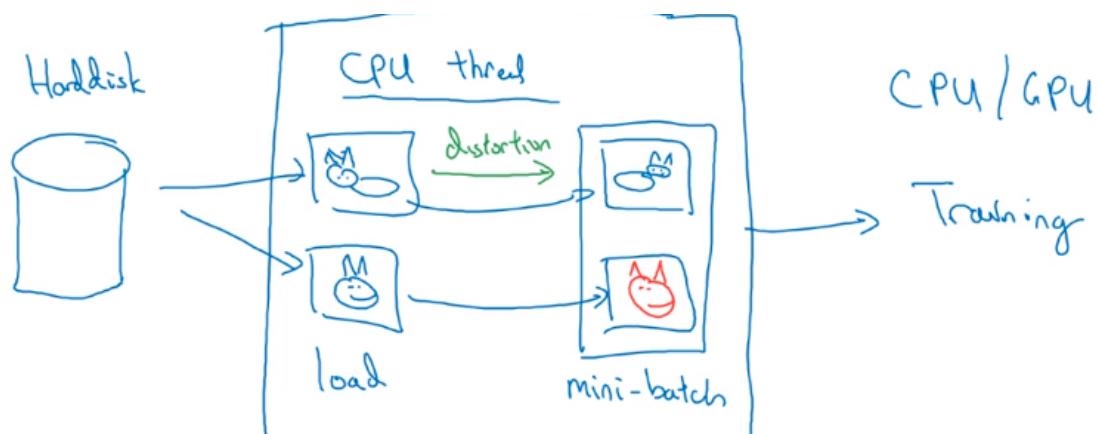


Cambiado de color: distorsionar los colores RGB para tener distintos colores.



Implementar distorsiones durante el entrenamiento

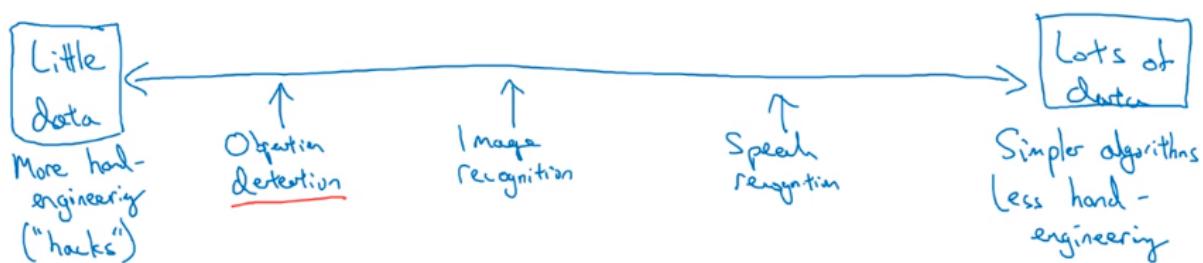
Mientras se entrena un conjunto de imágenes otro thread/hilo de la CPU/GPU puede estar generando nuevas imágenes mediante distorsión. El resultado de esto será un mini-batch del cual se realimentará la red para continuar entrenándola.



Estado de la visión computacional

Datos vs ingeniería dura

Estados de los principales problemas en Deep Learning en función del número de datos disponibles. Detección de objetos posee muchos menos datos que reconocimiento de imágenes debido a que necesita el encuadre de los objetos.



Cuando un problema tiene muchos datos, se suele utilizar menos ingeniería dura, mientras que si se poseen menos datos se utiliza más (o hacks o trucos, Transfer Learning).

De esta manera se concluye que las dos fuentes de conocimiento de una red son: datos etiquetados o hacks que vienen dados por features modificados como los vistos anteriormente, modificaciones a la arquitectura de la red, etc.

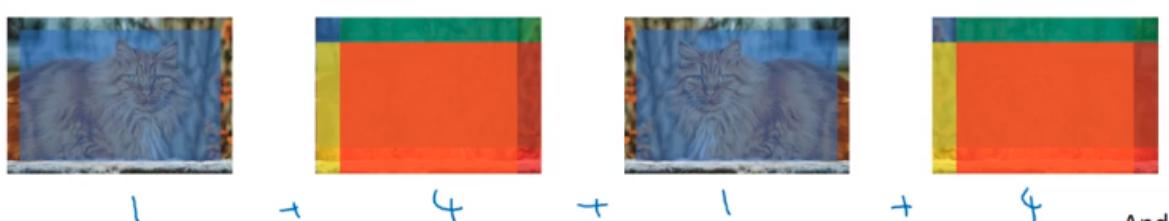
Two sources of knowledge

- • Labeled data (x, y)
- • Hand engineered features/network architecture/other components



Consejos para obtener buenos resultados en benchmarking/ganar competiciones

- Ensamblado: entrenar 3 a 15 redes neuronales separadamente y promediar su salida (no sirve promediar sus pesos). Esto entrega resultados 1% o 2% mejor. Como inconveniente el tiempo adicional que se tarda es proporcional al número de redes adicionales que se entrena.
- Multi cortado en tiempo de test (multicrop at test time): correr el clasificador en muchas versiones de imágenes de prueba y promediar los resultados.

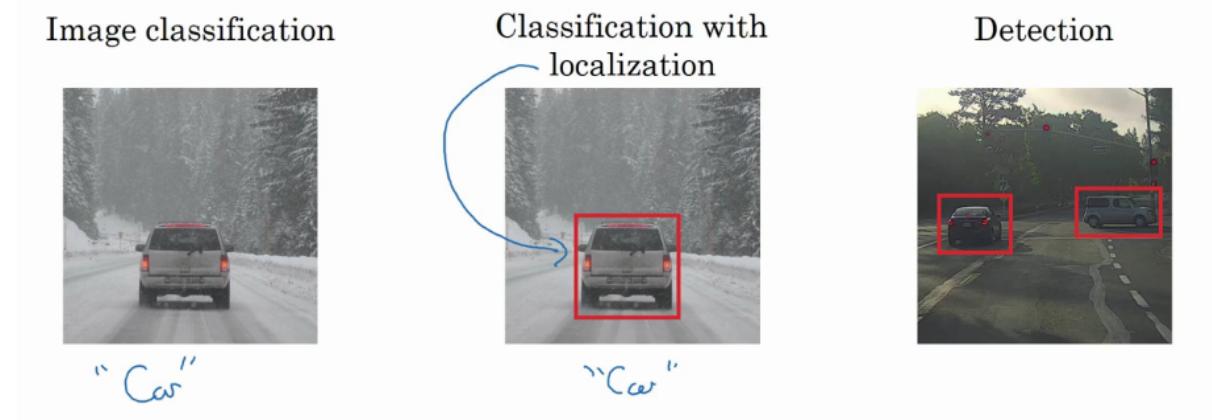


Week 3: Detección de objetos

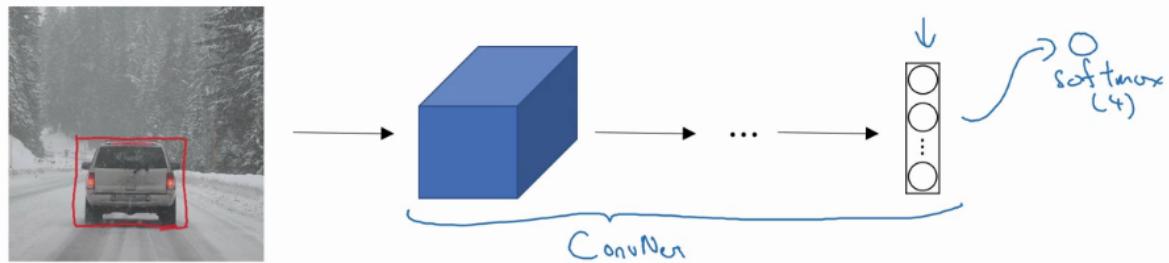
Antes de estudiar la detección de objetos vamos a estudiar la localización de objetos.

Localización de objetos

(izquierda) Como hemos visto en clasificación de imágenes, basta con decir si la imagen pertenece a un auto o no. (centro) Pero en clasificación con localización no solo hay que indicar si hay un auto, sino que hay que encuadrar el auto en cuestión. (derecha) múltiples autos son posibles y hay que identificarlos. No solo autos, sino también personas, bicicletas, etc.



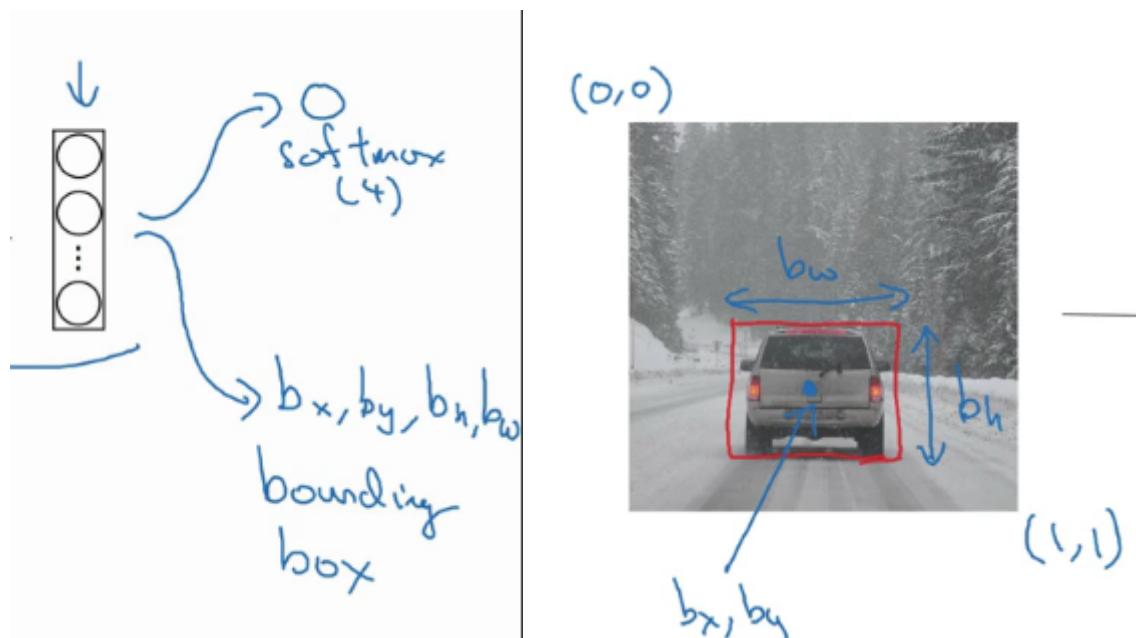
En el caso de que se quiera clasificar una foto, la red neuronal a implementar seria la siguiente:



- 1 - pedestrian ←
- 2 - car ←
- 3 - motorcycle ←
- 4 - background

Una red convolucional con un Softmax a la salida de 4 filas correspondiente a la detección de una persona, un auto, una moto o ninguna de las anteriores.

Para localizar un objeto necesitamos más salidas a la red neuronal. En este caso, 4 valores adicionales como pueden ser: b_x , b_y , b_h y b_w . Donde b_x y b_y corresponden al centro del rectángulo, b_h y b_w el alto y ancho de la caja respectivamente.



¿Cómo construir la salida 'y' a partir de una imagen?

El primer elemento nos dice si se ha detectado un objeto. Caso negativo, el resto de la salida no nos interesa.

Need to output b_x, b_y, b_h, b_w , class label (1-4)

$x =$

$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$

is there any object?

$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \leftarrow \text{"don't care"}$

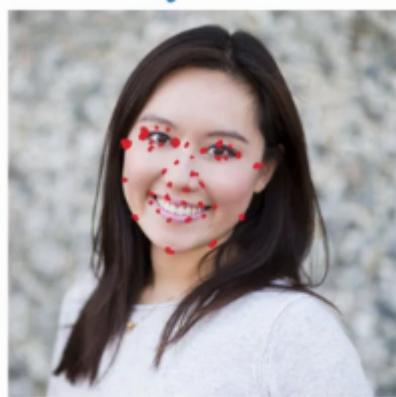
La función de coste podría definirse a partir de cuadrados mínimos como

$$\ell(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ + \dots + (\hat{y}_8 - y_8)^2 & \text{if } \underline{y_1 = 1} \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

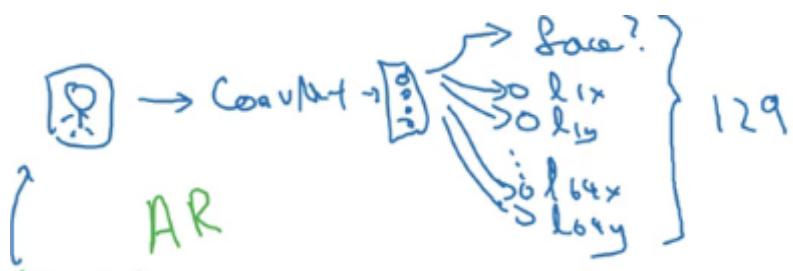
También se podría haber determinado **log-likelihood loss** para c_1, c_2, c_3 , regresión logística para p_c y SRE para b 's, aunque esta última puede funcionar bien en total como se dijo previamente.

Detección de puntos importantes (landmarks)

Supongamos que usamos una red neuronal para detectar los puntos de un ojo o cualquier otro punto arbitrario de una cara, por ejemplo.



$l_1x, l_1y,$
 $l_2x, l_2y,$
 $l_3x, l_3y,$
 l_4x, l_4y
⋮
 $l_{64}x, l_{64}y$



Esto suele ser utilizado en aplicaciones como la realidad aumentada para editar fotos en tiempo real, por ejemplo. A la salida uno obtiene por ejemplo una fila que dice si se detectó una cara y luego si limitamos el número de puntos (cada dos correspondiente a un landmark, x e y) a 64 entonces se tendrán 129 filas en total.

Detección de objetos

Algoritmo de detección de autos

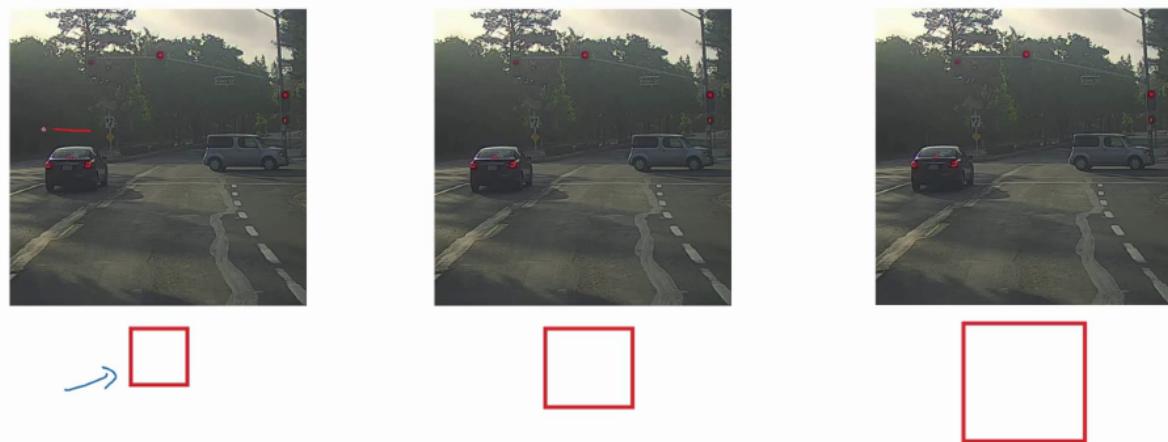
1. Creamos una base de datos con autos cortados de manera cercana y otros sin autos
2. Luego de crearla se puede entrenar una CNN de la siguiente manera



Algoritmo de las ventanas deslizantes



Ahora dada una imagen que puede contener uno o varios autos, se selecciona una ventana de un dado tamaño como se muestra en la figura y se empieza a mover a lo largo de la imagen. La ConvNet procesa cada uno de estos cuadrados que cubren el total de la imagen y detecta si hay un auto o no en base a lo entrenado anteriormente.

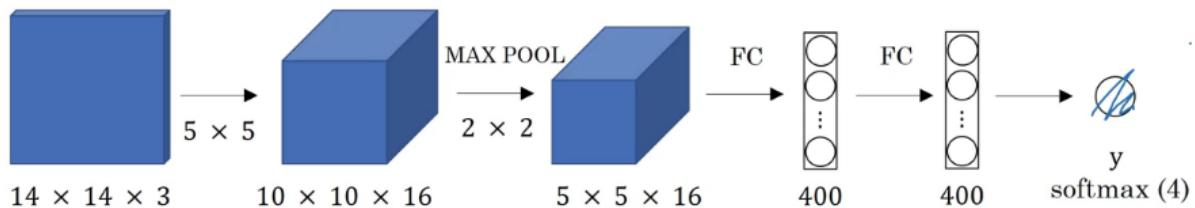


Luego esto se puede repetir para otros tamaños de rectángulos y distintos strides.

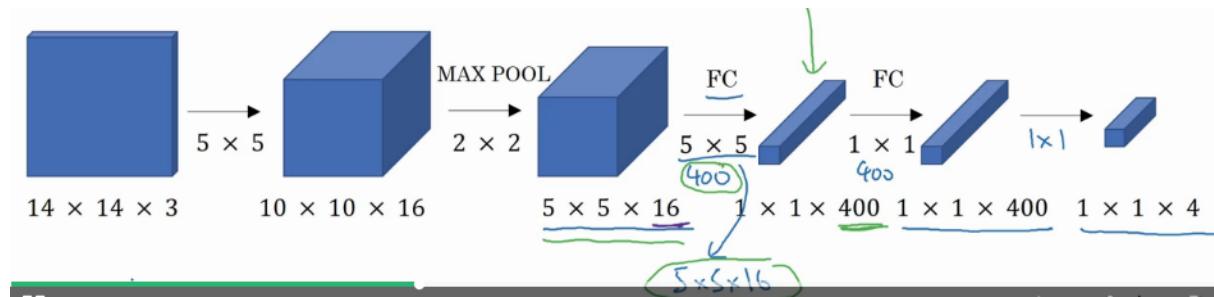
La desventaja de este método es el costo computacional que requiere evaluar, aun utilizando strides grandes. Se podría aumentar el stride para mejorar el costo computacional, pero disminuiría la precisión. Afortunadamente esto se puede implementar de una manera mucho más eficiente.

Implementación convolucional de las ventanas deslizantes

Antes de explicar estas ideas, es necesario introducir como pasar de una capa completamente conectada (FC) a una capa convolucional. Para ello primero dada la siguiente red



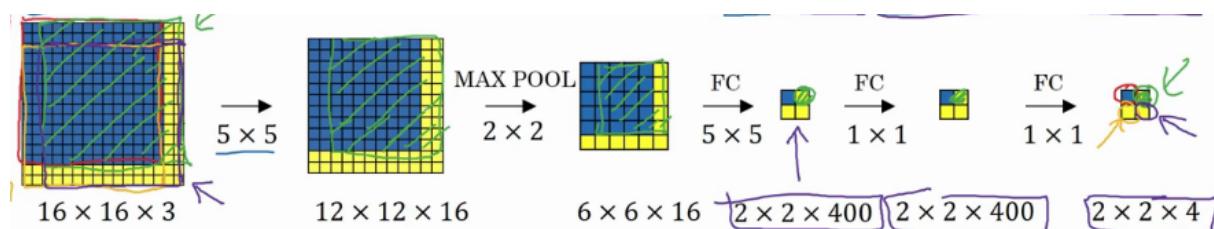
Se pueden reemplazar las capas completamente conectadas utilizando por ejemplo 400 filtros de 5×5 . En este caso la salida será de tamaño $1 \times 1 \times 400$. A esta última se le aplican 400 filtros de 1×1 como muestra la figura.



¿Cómo implementamos el algoritmo de ventanas deslizantes utilizando CNN? Esto está basado en el paper siguiente:

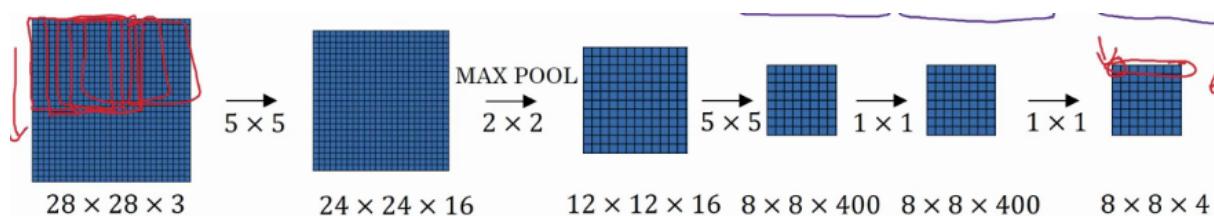
- [Sermanet, Pierre, et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks." arXiv preprint arXiv:1312.6229 \(2013\).](#)

Si, por ejemplo, se tiene una imagen de $16 \times 16 \times 3$ con un stride de 2. El algoritmo se debería correr 4 veces para realizar la detección de objetos. Esto provoca que haya mucho cómputo repetido. Veamos que si, en su lugar, se realiza el cómputo de la siguiente red

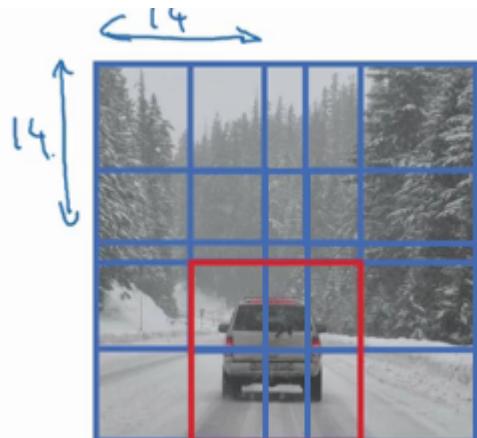


La salida será de, en lugar de $1 \times 1 \times 4$, de $2 \times 2 \times 4$, donde el primer subset de $1 \times 1 \times 4$ (arriba a la izquierda) corresponde a la primera ventana de $14 \times 14 \times 3$, el segundo subset (arriba a la derecha) corresponde a la segunda ventana de $14 \times 14 \times 3$ y así sucesivamente.

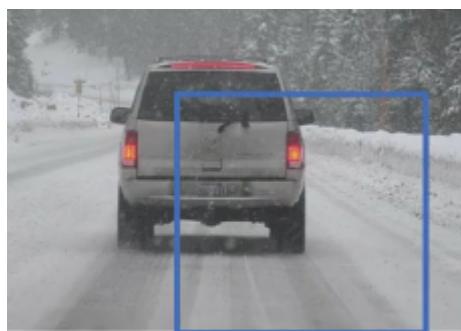
Un ejemplo más grande sería el siguiente, donde la salida se corresponde con un stride de 2.



Aplicando esto al problema del principio, la imagen se toma como entrada a la red y aplicando convolución la red detectará que en el recuadro rojo hay un auto efectivamente.



Este algoritmo todavía presenta un problema que está relacionado con la salida de los cuadrados (ver figura inferior). Veamos a continuación como solucionar este problema.



Predicciones de los cuadrados (bounding box)

Algoritmo YOLO (You Only Look Once)

Basado en el siguiente paper:

- [Redmon, Joseph, et al. "You only look once: Unified, real-time object detection."](#)
[Proceedings of the IEEE conference on computer vision and pattern recognition.](#)
[2016.](#)



Consideremos primero una imagen a analizar de 100×100 . La misma se dividirá en una grilla que en este caso será de 3×3 pero que usualmente será más fina, digamos, de 19×19 .

Cada una de estas grillas se procesará por una NN. La salida de las mismas vendrá dada por los parámetros que vimos a principio de la semana, estos son: p_c , si hubo detección, la posición del centro de la grilla, b_x, b_y , el ancho y alto del rectángulo, b_w, b_h y un conjunto de números que indica el tipo de detección que hubo c_1, c_2 y c_3 .

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

La salida de la primera grilla donde no hay objeto se obtendrá un 0 y un Don't care (nada) para todo lo demás. El algoritmo si detecta un objeto en la grilla, detecta el punto medio correspondiente al objeto que detectó.

La salida total (target output) de la red será de tamaño $3 \times 3 \times 8$ o, lo que es lo mismo, $\# \text{degrillas} \times 8$, donde 8 es el número de parámetros que indican la detección del objeto y el tamaño de la grilla. Para la salida de la grilla del medio a la izquierda se obtendrá una de los vectores de la figura de la izquierda.

El problema de tener varios objetos en una grilla se discutirá más adelante.

Nota: El algoritmo no se corre un #de grillas veces sino que al ser convolucional se hace una vez, donde la salida viene dada por este número y cada uno de ellos equivale a una posición en particular de la grilla. Este algoritmo es lo suficientemente rápido como para funcionar en detección y clasificación de objetos en tiempo real.

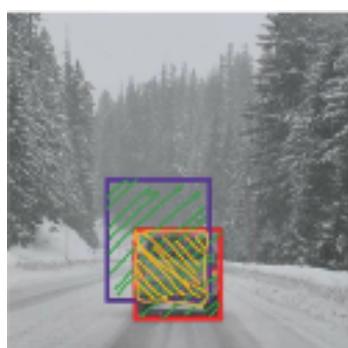
¿Cómo se codean los parámetros b_x, b_y, b_w y b_h ?



Podemos notar que una grilla en particular posee extremos entre $(0,0)$ y $(1,1)$. La posición de la mitad del objeto se calcula y deben ser números entre 0 y 1, Por otra parte, el ancho y el alto del objeto no necesariamente tienen que ser menores a uno pueden ser mayores si el objeto es más ancho o alto que la grilla en la que se encuentra.

Ahora vemos algunas características que hacen funcionar el algoritmo un poco mejor.

Intersección sobre unión (IoU)



Medida de solapamiento entre dos bounding boxes: Dado el rectángulo obtenido como salida al evaluar la localización de un objeto (azul) y el rectángulo correspondiente como verdadero (rojo), la intersección sobre unión viene dado como el cociente entre el tamaño del rectángulo de la intersección (amarillo) dividido el tamaño del rectángulo de la unión (rojo). En el caso de obtener un IoU mayor o igual a 0.5 (a veces se usa 0.6 también) la predicción de evaluar la localización del objeto se tomará como correcta. Mientras que si es menor a 0.5 será incorrecto.

¿Cómo evitar que un objeto se detecte una vez en lugar de detectar varias veces un solo objeto?

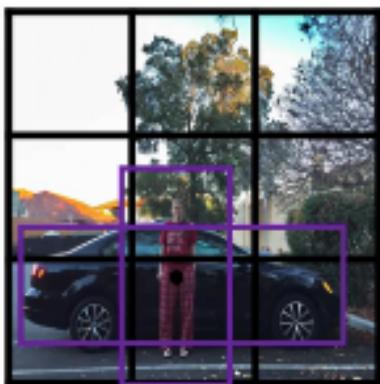


Podríamos usar Non-max suppression.

Un ejemplo donde el algoritmo detecta varios objetos duplicados donde debería haber solo uno, se muestra en la figura de la derecha, cada rectángulo con una probabilidad. Non-max suppression lo que hace es quedarse con el rectángulo de máxima probabilidad y descartar los que tengan mucho solapamiento entre si.

Implementación:

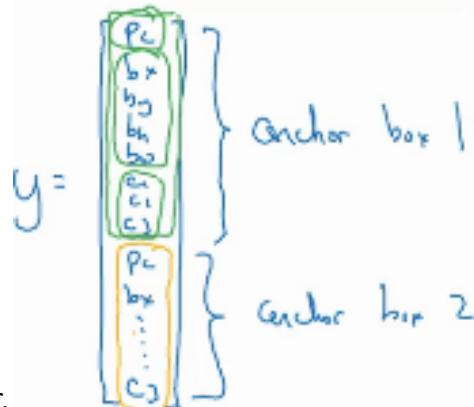
1. Descartar todos los rectángulos con probabilidad menor a 0.6.
2. Mientras hayan rectángulos restantes
 - a. Elegir un rectángulo con el p_c más grande. Predecir su salida.
 - b. Descartar cualquier rectángulo restante con IoU mayor o igual a 0.5 con respecto a la salida del paso anterior.



Anchor boxes

Volvamos al problema donde hay varios objetos por cuadrado de grilla, ¿Cómo hacemos para detectar dos o más objetos en un mismo cuadro? Veamos un ejemplo primero. En la

figura de la derecha se puede notar que los puntos medios de los objetos están



prácticamente en el mismo lugar.

Para ello la salida se concatena para poder realizar la detección.

Antes: cada objeto de entrenamiento era asignado a cada celda de la grilla que contiene el punto medio del objeto.

Ahora con dos anchor boxes: cada objeto en la imagen de entrenamiento es asignado a la celda de la grilla que contiene el punto medio del objeto y anchor box para la celda de la

grilla con IoU más grande.

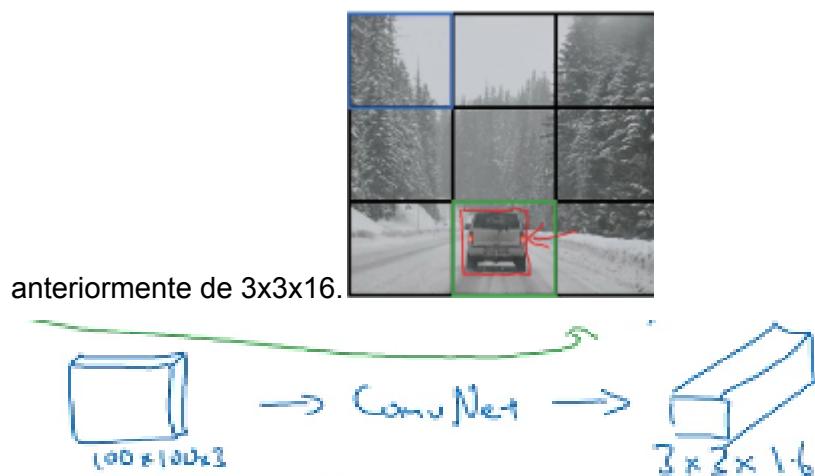


Todavía siguen habiendo varias cuestiones inconclusas. Pero antes previa detección se espera que se detecte objetos con una forma determinada denotada en la imagen como anchor box 1 y 2. Si hay dos anchor boxes y tres objetos, el algoritmo no funcionará correctamente. Si hay dos objetos con el mismo tipo de anchor box tampoco funcionará correctamente. Cuando se usa un grillado chico el uso de anchor boxes no es tan importante dado que es poco probable que haya detección de varios objetos en una región tan compacta de la figura. Por otra parte, cabe agregar que el tipo anchor box 1 se suele utilizar para personas (altas y angostas) mientras que el tipo 2 suele ser utilizado para automóviles (anchos y bajos). En la práctica se suelen elegir entre 5 y 10 tipos de anchor boxes.

Juntando todo lo visto en el algoritmo YOLO

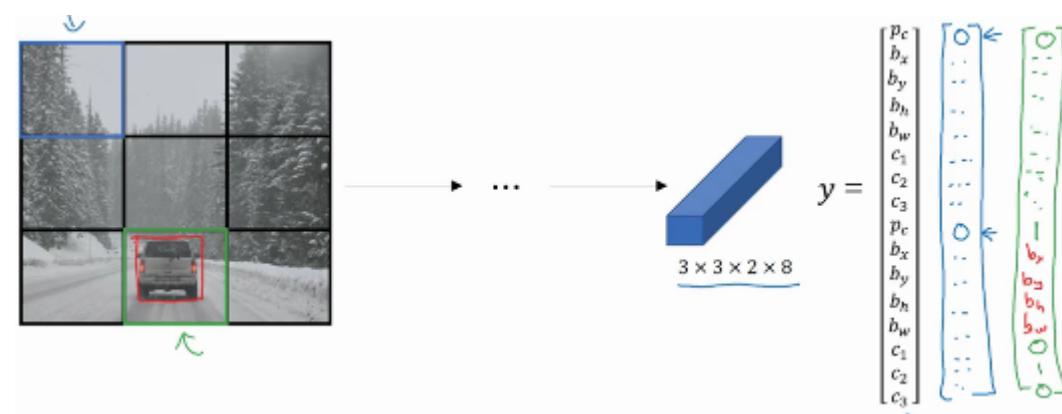
Entrenamiento

Dada la siguiente figura y una grilla de 3x3 donde se quieren detectar 1. personas, 2. autos o 3. motocicletas, la salida poseerá un tamaño dado por $3 \times 3 \times 2 \times 8$, donde 2 = #anchor boxes (esto suele ser a elección. Generalmente se eligen 5) y 8 = $5 + \# \text{clases} = \# \text{parametros de objeto localizado}$. Usualmente el tamaño de la salida suele ser de $19 \times 19 \times 40$ debido al aumento del grillado y el número de anchors a 5. La salida del elemento 1x1 al no haber detección vendrá dada por un 0 en el elemento p_c y un don't care en los restantes, mientras que en la celda inferior central hay una detección de un 2. automóvil por lo cual elemento 3x2 de tamaño 16 tiene sus primeros 8 elementos como un 0 en p_c y don't cares en el resto, los siguientes 8 elementos vienen dados por un 1 en p_c dada la detección positiva, los siguientes cuatro números determinan el tamaño y posición del rectángulo localizador y los últimos 3 la clase de objeto detectado, en este caso, 0 1 0. Por último, la entrada viene a ser una imagen de $100 \times 100 \times 3$ en este caso que pasa por una ConvNet y otorga la salida dicha



Predicciones

Nuevas imágenes serán pasadas como entradas a la red neuronal y si todo funciona bien predecirá lo que es esperado a la salida de la misma.



Salida del algoritmo non-max suppressed

- Para cada celda de la grilla, se obtienen dos cajas de predicciones.
- Deshacerse de las predicciones con probabilidad baja.
- Para cada clase (persona, auto, moto) usar non-max suppression para general la predicción final.

Región de propuestas (R-CNN, Region proposal)

Basado en el paper:

- [Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.](#)



Básicamente como la figura muestra hay figuras en las que grandes regiones no importantes pueden ser estudiadas perdiendo gran tiempo de cómputo. En lugar de correr sliding windows en todas las ventanas se hace en una cierta cantidad. Esto se detecta corriendo un algoritmo llamado algoritmo de segmentación



Como muestra la figura, el algoritmo va detectando figuras de interés y la red se corre únicamente en estos objetos que sean de interés, por ejemplo, del orden de 2000.

R-CNN: Propone regiones. Clasifica regiones propuestas una a la vez. Salida del rectángulo localizador y etiqueta.

Algoritmos más rápidos

Fast R-CNN: Usa la implementación convolucional de ventanas deslizantes para clasificar todas las regiones propuestas. Las regiones para proponer regiones siguen siendo lo más lento. Paper:

- [Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.](#)

Faster R-CNN: Usa redes convolucionales para proponer regiones. Paper:

- [*Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.*](#)

Week 4: Aplicaciones especiales: Reconocimiento de caras & Neural Style transfer

Reconocimiento facial

¿Qué es reconocimiento facial?

En las oficinas de Baidu, China se utiliza reconocimiento facial para dejar entrar a una persona a cierto lugar, pero a su vez usa la técnica de detección de “liveness” para evitar que alguien entre con una cara de la persona impresa en una foto, por ejemplo. Esto es logrado simplemente con aprendizaje supervisado, pero en el curso se prestará más atención al reconocimiento facial.

Verificación facial vs Reconocimiento facial

La verificación es un proceso más simple (1-1). Se necesita una imagen de entrada y un nombre o ID con el objetivo de determinar si la imagen es la de la persona en cuestión. Por otra parte, en reconocimiento se posee una base de datos de K personas (1-K), se requiere una imagen como entrada y como salida se obtiene la ID en el caso de que la imagen coincida con alguna de las K personas (o no reconocida en el caso de no estar en la base de datos). Cabe notar que, si se posee un algoritmo de verificación facial de 99% de accuracy, el reconocimiento para dadas 100 personas, será mucho menor, por lo cual un accuracy del orden de 99.9% es requerida.

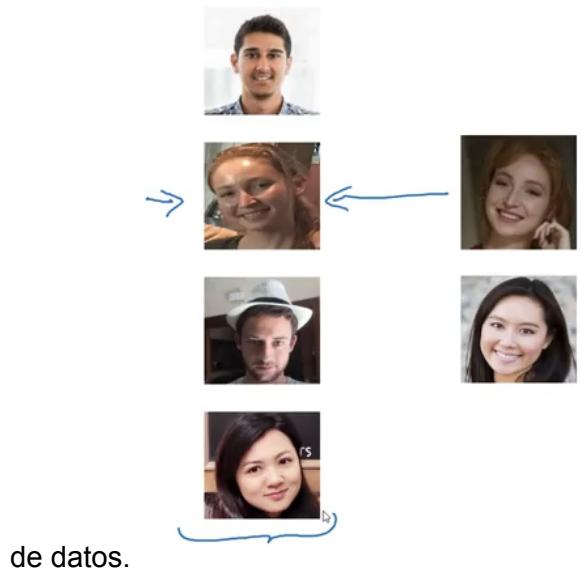
Face verification: "is this the claimed person?". For example, at some airports, you can pass through customs by letting a system scan your passport and then verifying that you (the person carrying the passport) are the correct person. A mobile phone that unlocks using your face is also using face verification. This is a 1:1 matching problem.

Face Recognition - "who is this person?". For example, the video lecture showed a face recognition video (<https://www.youtube.com/watch?v=wr4rx0Spihs>) of Baidu employees entering the office without needing to otherwise identify themselves. This is a 1:K matching problem.

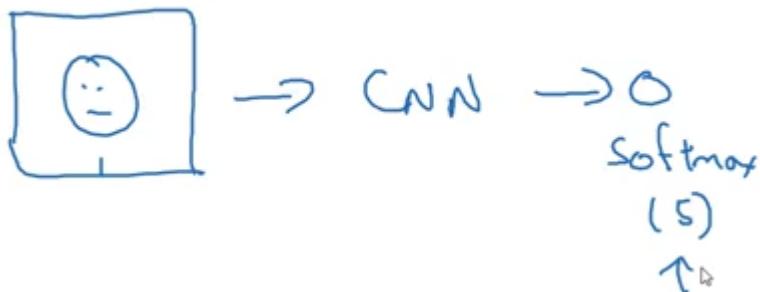
One Shot Learning

Uno de los desafíos más importantes para el reconocimiento facial es el hecho de poder reconocer una persona con una única imagen de ejemplo.

Por ejemplo, dadas las caras de 4 empleados (primera columna) uno quiere detectar si una persona ahora (segunda columna) se corresponde con alguno de los empleados de la base



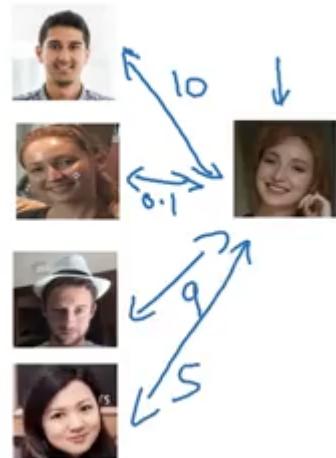
de datos.



Una manera de tratar este problema sería agarrar las 4 fotos y entrenar una CNN con una salida Softmax (5) cada una correspondiente a una persona y una para ninguna. Pero dada la baja cantidad de empleados esto no funcionará correctamente. Por otra parte si un nuevo empleado es tomado, la salida deberá ser Softmax(6) lo cual provocaría que se tenga que entrenar la red nuevamente, siendo un proceso sumamente ineficiente.

En su lugar, se aprenderá una función de “similitud”. Esta mide la diferencia entre las imágenes y se compara si esta es menor a una threshold. En caso afirmativo se considera que las imágenes pertenecen a la misma persona, mientras que si la diferencia es mayor al

threshold se consideran a las personas distintas. Como se muestra en la figura de la izquierda, se espera que, si las personas son las mismas, la diferencia entre ellas sea menor al threshold y que para las restantes personas sea mayor. Por otra parte, si una nueva persona es incorporada a la empresa, no habrá que entrenar una red como sucede



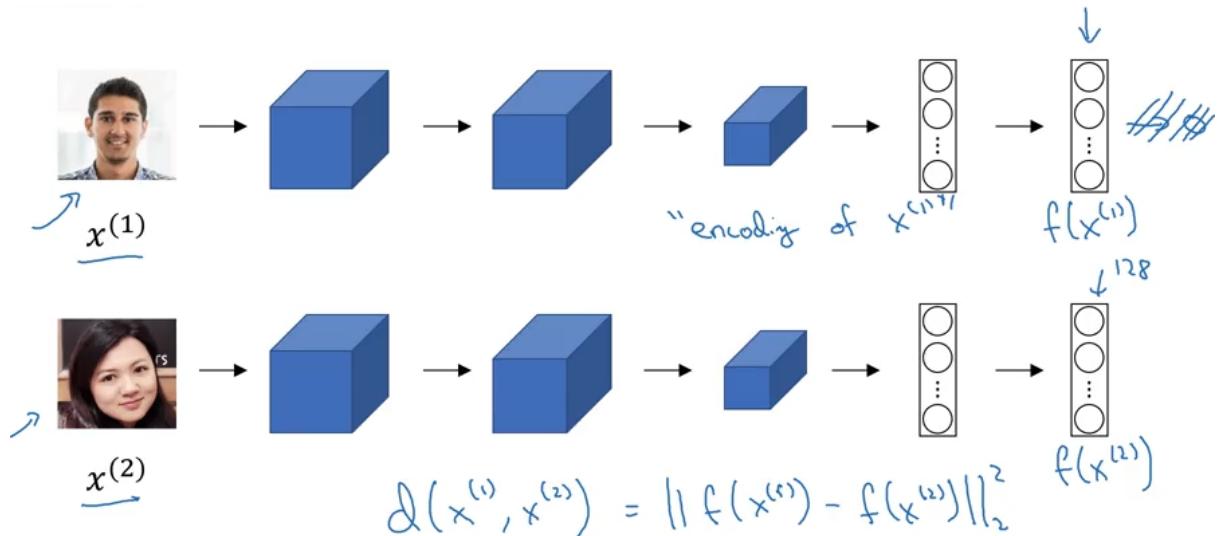
con la CNN.

Pero, ¿Cómo se entrena una red neuronal para aprender esta función?

Siamese Network

Esta red fue introducida en el siguiente trabajo:

- [Taigman, Yaniv, et al. "Deepface: Closing the gap to human-level performance in face verification." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.](#)



Básicamente, dada dos imágenes que se quieren comparar, estas se insertan en una red como la de la figura compuesta por ejemplo de filtros y dos capas completamente conectadas (FC) (sin Softmax). Esto hecho por cada imagen se llama Encoding de $x(i)$ donde la salida de la red es $f(x(i))$. La manera de entrenar esta red es aprender los parámetros de manera tal que la diferencia entre estos Encodings sea pequeña dadas imágenes de la misma persona.

Learn parameters so that:

If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large.

Pero, ¿Cómo se define una función de costo que entrene los parámetros de la red?

Triplet Loss

Este trabajo fue desarrollado principalmente del paper siguiente:

- [Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.](#)

Dadas dos personas iguales yo quiero que los Encodings sean pequeños, mientras que son si no son la misma, que sea grande. En la terminología Anchor (A) es la imagen con la que

comparar unas imágenes Positiva (P) o negativa (N), estas son la misma persona u otra, respectivamente.



Anchor
A

Positive
P

Anchor
A

Negative
N

Básicamente, queremos que la diferencia entre las imágenes anchor y positive sea menor que la anchor y negative, pero para evitar que la red entrene soluciones triviales (iguales a cero) se agrega un hiper parámetro llamado margen (α).

$$\text{Want: } \underbrace{\frac{\|f(A) - f(P)\|^2}{d(A, P)}}_{\alpha} \leq \underbrace{\frac{\|f(A) - f(N)\|^2}{d(A, N)}}_{\alpha}$$

$$\frac{\|f(A) - f(P)\|^2}{\alpha} - \frac{\|f(A) - f(N)\|^2}{\alpha} + 2 \leq 0 \quad \text{if } f(\text{img}) = \vec{0}$$

Por ejemplo, en el caso de que α fuera cero, si la diferencia entre Anchor y positive fuera 0.5, bastaría con que la diferencia entre anchor y negative sea 0.51 por ejemplo, pero si elegimos $\alpha = 0.2$ haría que la diferencia $d(A, N)$ sea del orden de 0.7.

Definamos la función de pérdida como:

$$L(A, P, N) = \max \left(\underbrace{\frac{\|f(A) - f(P)\|^2}{\alpha} - \frac{\|f(A) - f(N)\|^2}{\alpha} + 2}_{\alpha}, 0 \right)$$

Para minimizar esto, la red quiere que el primer término sea lo más negativo posible. La función de coste total sobre un training set será la suma de esta función de coste sobre cada imagen.

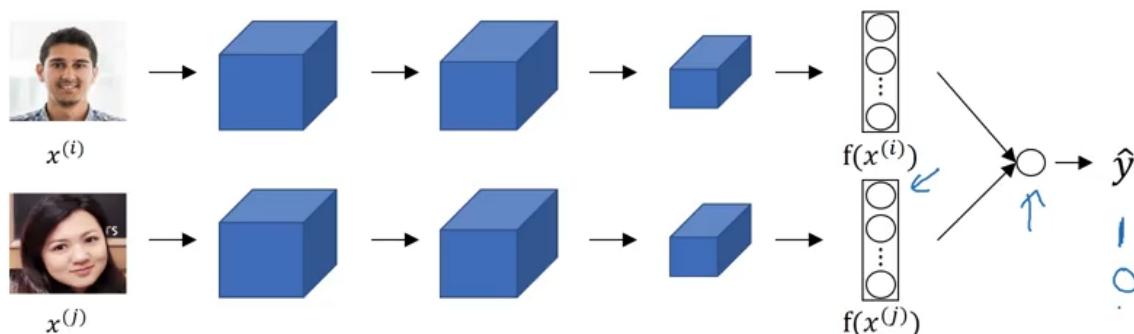
$$J = \sum_{i=1}^m L(A^{(i)}, P^{(i)}, N^{(i)})$$

Training set: 10k pictures of 1k persons

Para hacer esto necesitamos si o si pares de imágenes de la misma persona.

Un inconveniente con esto es que si se eligen A, P y N aleatoriamente entonces el constrain será fácilmente satisfecho, ya que $d(A.N)$ será muy grande. Es por ello que conviene utilizar triplets de tal manera que la red sea “difícil” de entrenar.

Verificación facial y clasificación binaria



El reconocimiento facial puede ser pensado como un problema de clasificación binaria mediante la figura superior.

¿Qué es lo que hace la unidad de regresión logística final? La salida y_{hat} será evaluar una función sigmoide digamos a la diferencia entre vectores de la siguiente manera

$$\hat{y} = \sigma \left(\sum_{k=1}^{128} w_i \underbrace{|f(x^{(i)})_k - f(x^{(j)})_k|}_{+ b} \right)$$

Esta unidad puede tener parámetros w y b adicionales los cuales deben ser entrenados para predecir si o no las imágenes son de la misma persona. Otra manera de computar esto sería mediante la diferencia chi cuadrada entre $f(x(i))$ y $f(x(j))$ como

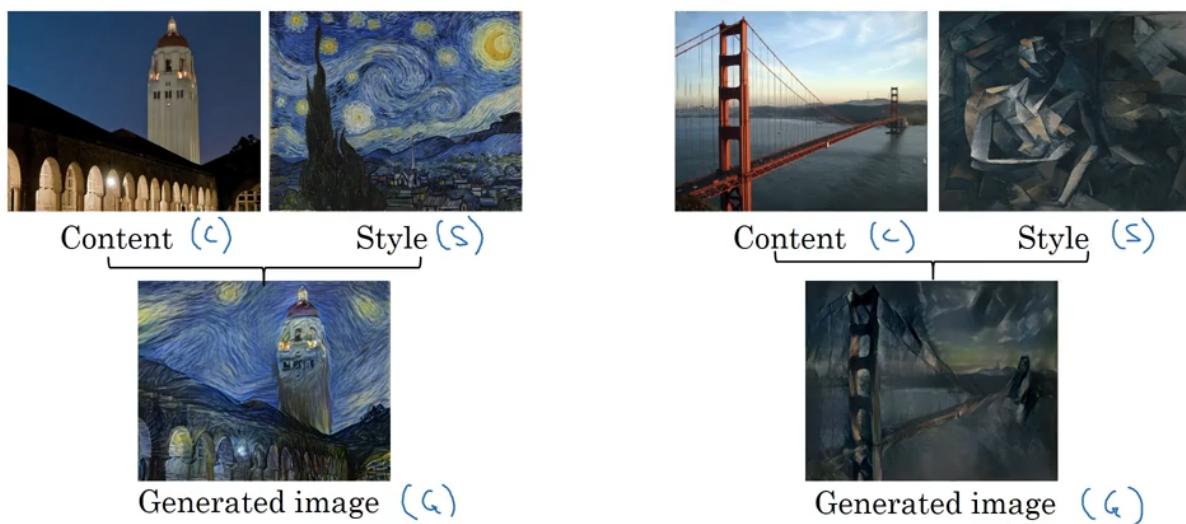
$$\frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$$

La entrada es un par de imágenes. La salida es 1 o 0 dependiendo si las imágenes son similares o no. Tanto la red siamesa superior como inferior poseen parámetros los cuales hay que entrenar.

Una manera de agilizar el entrenamiento consiste en si, un empleado ingresa en lugar de recomputar toda la red nuevamente, lo que se puede hacer es pre-computarla así, si alguien nuevo entra, usa esos Encodings para realizar la predicción y_{hat} .

Neural Style Transfer

Aplicación de las CNN más interesantes, genera obras de arte. Dada una imagen a la que llamamos contenido (c) se le va a aplicar un estilo de imagen (s), la salida de este proceso se llamará (g).



En estos ejemplos, ¿qué es lo que las capas profundas de la red convolucional aprende?

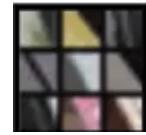
Esta sección está basada en la investigación realizada por el paper:

- [Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham. 2014.](#)

Tomemos por ejemplo una red tipo AlexNet. Agarremos una unidad oculta de la primera capa y miremos la imagen que maximiza esa activación de la unidad.

Podemos ver que buscará líneas inclinadas hacia la izquierda. Si agarramos otras buscará líneas inclinadas hacia la derecha, otras líneas horizontales

anaranjadas. Es decir que en la primera capa se distinguen fragmentos simples como bordes.



En las capas más profundas de la red neuronal veremos estructuras más profundas. En resumen, parece que la capa 1 detecta bordes. La capa 2, texturas verticales más complejas, figuras redondas en la parte superior de la imagen, líneas delgadas, etc. En la capa 3, tenemos personas, figuras redondas, texturas, etc. En la capa 4, se detectan patas de animales, agua, perros pero muy similares entre si, etc. Por último, la capa 5 parece detectar cosas más sofisticadas, perros más diferentes entre si, flores o texto.



Layer 1



Layer 2



Layer 3



Layer 4



Layer 5

Función de coste

La idea principal de Neural Style Transfer es obra del siguiente paper:

- [Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style." arXiv preprint arXiv:1508.06576 \(2015\).](#)



Definamos una función de coste que mide cuán bien es una imagen generada. Esta posee dos partes: una que mide qué tan bien la imagen generada representa el contenido de C y qué tan bien la imagen generada representa el contenido de S, cada una con su respectivo peso asociado.

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

En orden para generar la nueva imagen lo que se hace es lo siguiente

1. Iniciar G aleatoriamente, por ejemplo, G: 100x100x3.
2. Usar gradiente descendente para minimizar J(G).

Es por ello que a medida que aumentan las iteraciones, la imagen generada pasa de ser una imagen completamente ruidosa a algo que adquiere el estilo de S, pero manteniendo el contenido de C.

Función de coste de contenido

Digamos que usamos una capa oculta I para computar el costo. Si es la primera capa por ejemplo la imagen generada aleatoriamente deberá ser similar a C para no haber tanta diferencia. En la práctica, se utiliza un valor intermedio entre las primeras y las últimas.

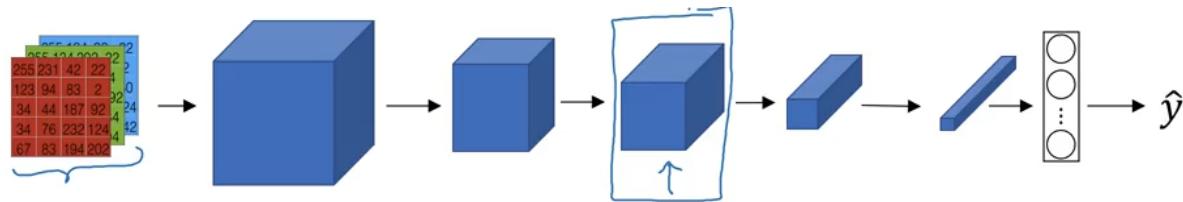
Usa una red entrenada previamente por ejemplo como ConvNet. ¿Cuán similares son C y G en contenido? Sean $a^{[l](C)}$ and $a^{[l](G)}$ las activaciones de la capa l en las imágenes C y G respectivamente. Si ambos numeros son similares, las imágenes poseen contenidos similares. Entonces se define la función de coste como

$$J_{\text{content}}(C, G) = \frac{1}{2} \|a^{[l](C)} - a^{[l](G)}\|^2$$

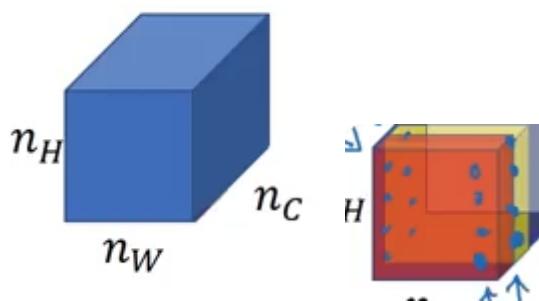
Función de coste de estilo

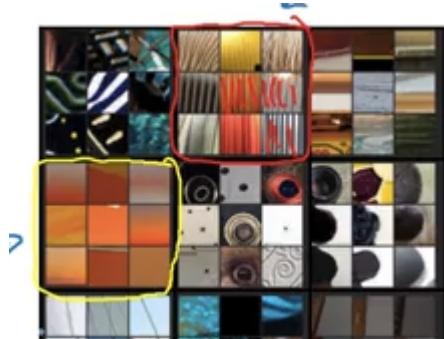
¿Qué significa el estilo de una imagen?

Dada la siguiente red, considerando la capa l sobre la cual se mide el estilo. *Definamos estilo como la correlación entre activaciones a lo largo de los canales.*



Dada la capa en cuestión, veamos cuan correlacionadas están las activaciones a lo largo de los diferentes canales (N_C). Veamos por ejemplo los primeros dos canales, para un h y w fijos tenemos definidos un par de puntos correspondientes uno a cada canal. Esto nos define para todo H y W conjuntos de puntos del cual tenemos que medir la correlación.





Viéndolo en base a un ejemplo anterior, recordemos que para la capa 2 teníamos la figura de la izquierda. Si un canal (rojo) se corresponde con las figuras de las líneas verticales y el otro (amarillo) se corresponde con la de las figuras horizontales, entonces una manera de medir correlación seria por ejemplo, si siempre que se obtienen líneas verticales rojas, en la otra capa hay figuras horizontales naranjas.

Dada una imagen, calculemos una matriz de estilo (Gram matrix en álgebra lineal). Sea

$a_{i,j,k}^{[l]}$ la activación en el punto (i,j,k) donde $i=1, \dots, n_h$, $j=1, \dots, n_w$ y $k=1, \dots, n_c$. Por otra parte, la matriz de estilos que define la correlación entre dos capas de activación k y k' posee la simbolización definida a la izquierda. Un elemento de esta matriz entonces se

$$G_{kk'}^{[l]} \quad |k=1, \dots, n_c$$

calcula de la siguiente manera

$$\underline{G_{kk'}^{[l]}} = \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

Cabe aclarar que esto no es una correlación sino una cross-covarianza sin normalizar. Esto se realiza tanto para la imagen S como con G . Para mejorar la notación mejor escribimos

$$\begin{aligned} \underline{G_{kk'}^{[l](S)}} &= \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} a_{ijk}^{[l](S)} a_{ijk'}^{[l](S)} \\ \underline{G_{kk'}^{[l](G)}} &= \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} a_{ijk}^{[l](G)} a_{ijk'}^{[l](G)} \end{aligned}$$

De esta manera la función de coste de estilo viene definida según

$$J_{style}^{[l]}(S, G) = \| G^{[l](s)} - G^{[l](G)} \|_F^2$$

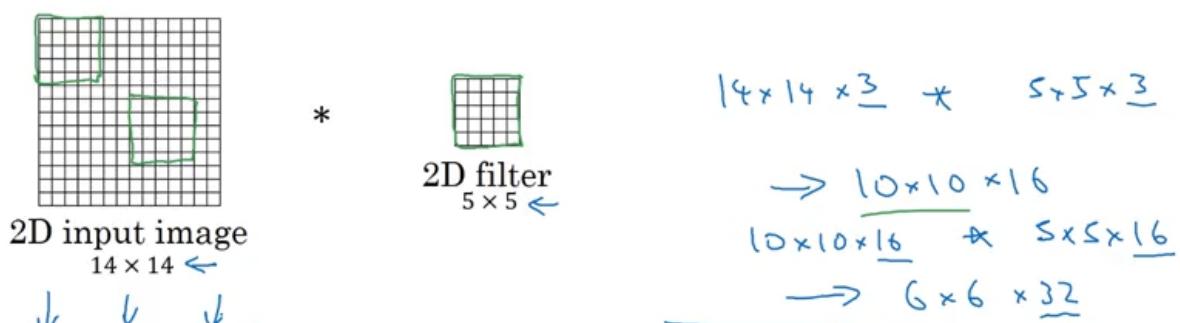
$$J_{style}^{[l]}(S, G) = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} (G_{kk'}^{[l](s)} - G_{kk'}^{[l](G)})$$

Se obtienen mejores resultados calculando la función de coste para distintas capas, que tienen en cuenta tanto high-level features como low-level.

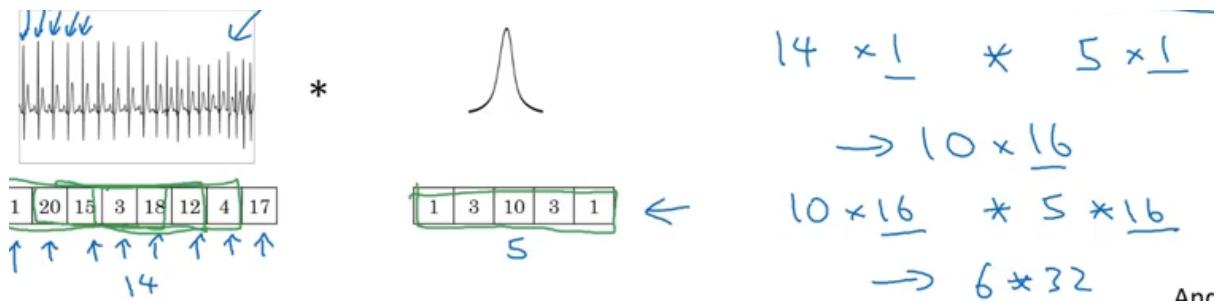
$$J_{style}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S, G)$$

Generalización a 1D y 3D

Hasta ahora hemos usado todo esto teniendo en cuenta imágenes en 2D, pero podemos ver cómo aplicarlo a otros tipos de datos. Recordemos que la convolución en 2D venía dada de la siguiente manera



En 1D se puede aplicar algo similar como sigue



Un ejemplo de dato de este estilo podría ser un electrocardiograma. Para 3D podemos pensar un CT scan, que toma distintos slices del cuerpo. Una convolución a un volumen de 3 dimensiones se puede calcular como sigue

