UNIVERSIDAD
NACIONAL
DE COLOMBIA

Analítica Predictiva

Carlos A. Madrigal

Profesor Ocasional

Departamento de Ciencias de la Computación y de la Decisión

Maestría en ingeniería - ingeniería de sistemas

Maestría en ingeniería - analítica

Especialización en sistemas

Nota: Este material se ha adaptado con base a diferentes fuentes de información académica

Contenido

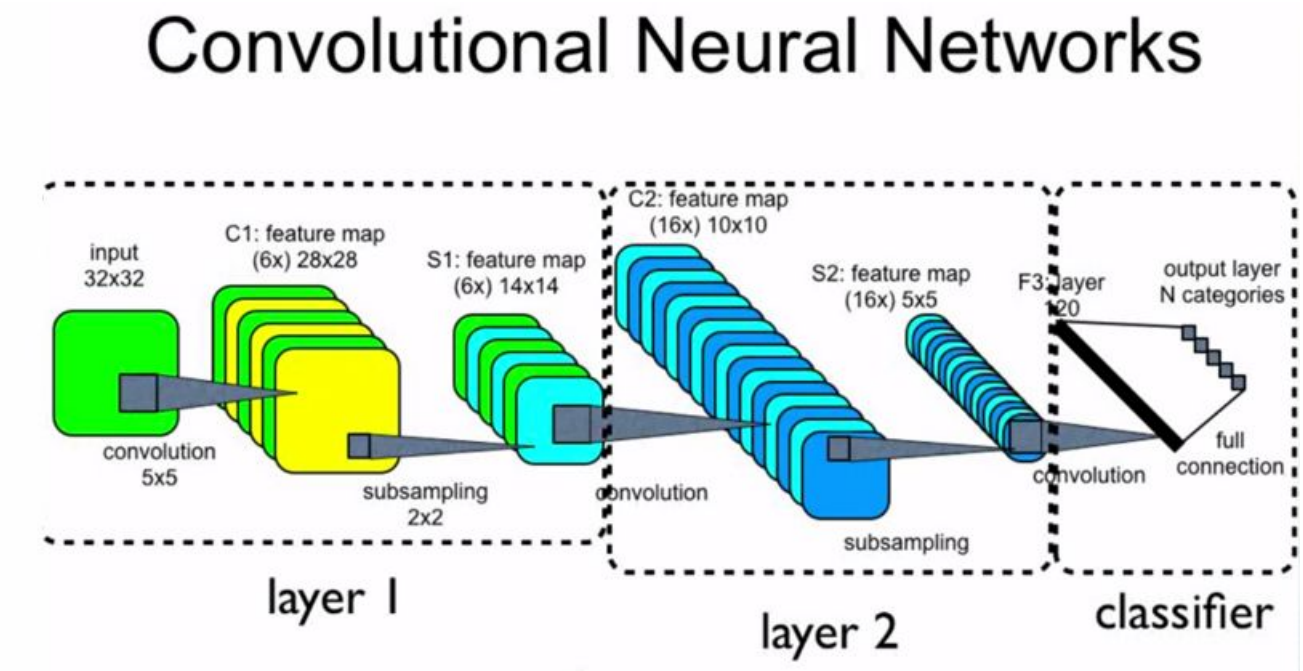
Redes Neuronales Convolucionales

- Detecciòn de bordes
- primera derivada de una imagen
- Convolución
- Descenso del gradiente
- Operadores detectores de borde
- Capa convolucional
- Capa pooling
- Arquitecturas Comunes

Redes Neuronales Convolucionales

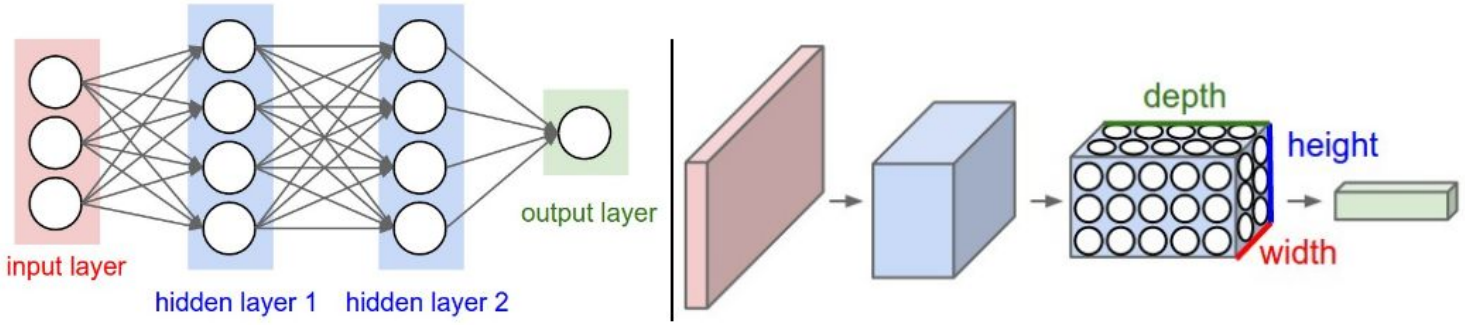
CNN Redes Neuronales Convolucionales

Son una variación de la Red Neuronal Perceptron Multicapa MLP, donde cada capa oculta está compuesta por capas convolucionales que luego se conectan a una o más capas *full connected.* Las CNN han probado ser muy efectivas en tareas de reconocimiento y clasificación de imágenes.



MLP vs CNN

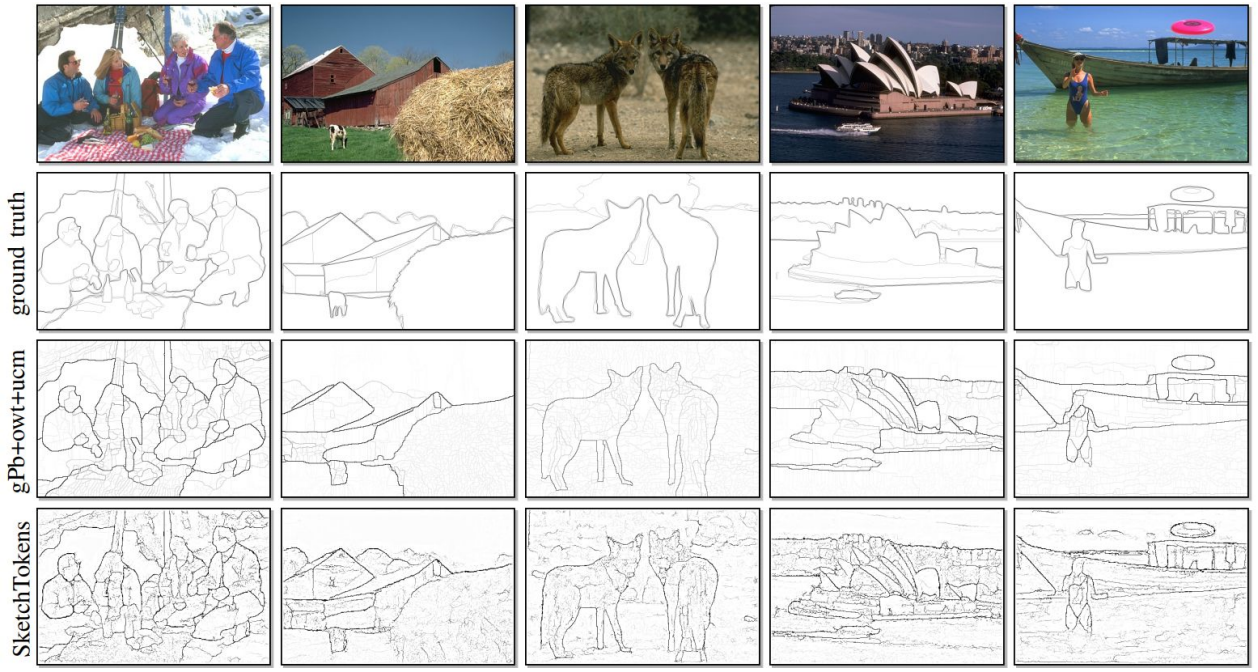
La red neuronal perceptrón multicapa recibe como entrada un vector de características y por su estructura *full connected* se le hace inmanejable computacionalmente recibir una imagen de mediana resolución como entrada a la red. Las redes convolucionales solucionan la limitación de las MLP y permiten que sus neuronas de entradas reciban una imagen completa.



http://cs231n.github.io/convolutional-networks/#overview

Detección de Bordes

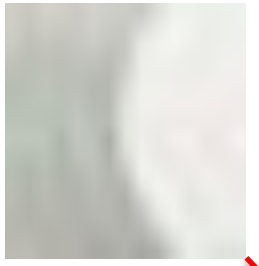
Los bordes en una imagen se pueden definir como los cambios más o menos bruscos de intensidades de gris entre dos regiones, y entre más bruscos sean estos cambios, mayor será el borde. Los bordes son de gran importancia, porque nos dan información sobre el tamaño, la forma y la textura del objeto.



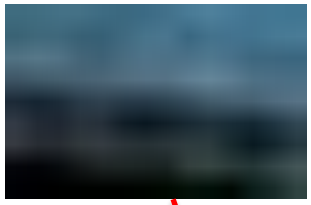
Dollár, P., & Zitnick, C. L. (2015). Fast edge detection using structured forests. *IEEE transactions on pattern analysis and machine intelligence*, *37*(8), 1558-1570.

Origen de los Bordes

Discontinuidad en el color



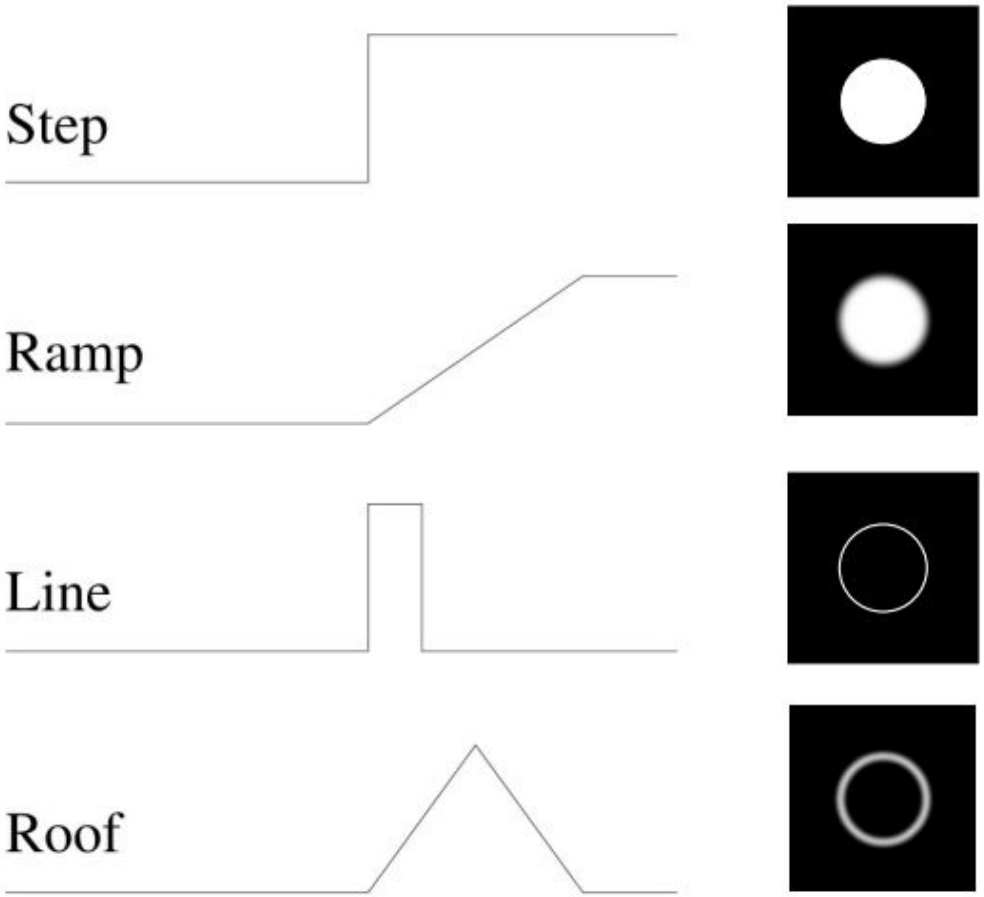
Discontinuidad en la iluminación



Discontinuidad en la profundidad

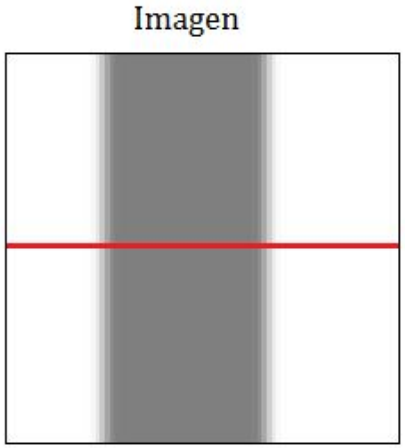
Discontinuidad en la normal de la superficie

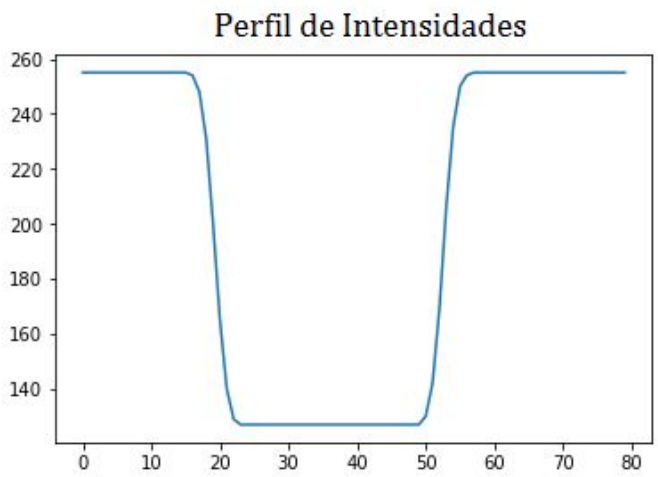
Como es un Borde?

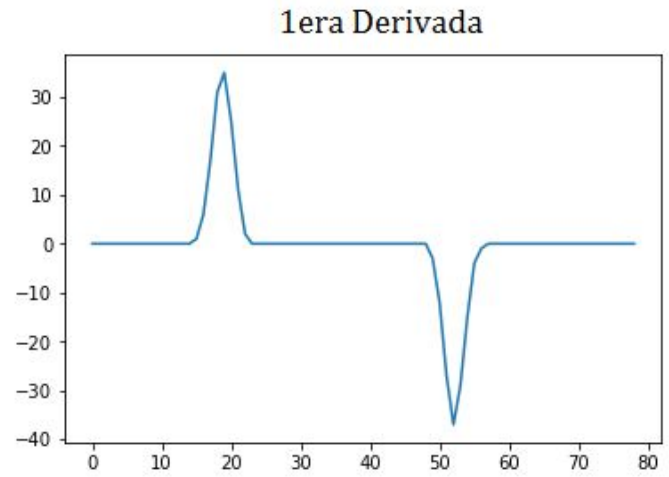


Operador Primera Derivada

Los filtros utilizados para la detección de bordes, son filtros diferenciales que se basan en la derivación. En la fig1. se muestra una imagen con bordes , en la fig2. el perfil de intensidades sobre una fila de la imagen y en la fig.3 la primera derivada sobre una fila de la imagen. La primera derivada es cero en las regiones uniformes en intensidad y tiene un valor constante durante los cambios de intensidad o bordes.







GRADIENTE

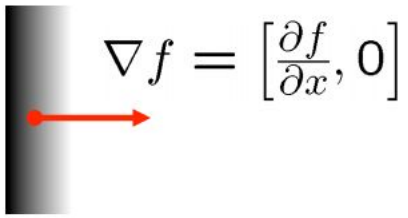
Para una funcion bidimensional f(x,y), la derivada es un vector con magnitud **G** y angulo Φ(x,y), que apunta en la dirección de la máxima variación de f(x,y) , a este vector se le denomina gradiente.

$$\boldsymbol{G}[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix}$$

$$|\boldsymbol{G}| = \sqrt[2]{{G_x}^2 + {G_y}^2}$$

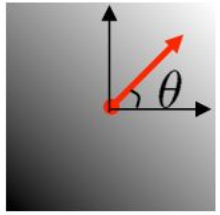
$$|\boldsymbol{G}| = |G_x| + |G_y|$$

$$\Phi(x,y) = \tan^{-1}\frac{G_y}{G_x}$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Gradiente

$$\frac{df}{dx} = \frac{f(x+dx) - f(x)}{dx} \approx \frac{f(x+dx) - f(x-dx)}{2dx}$$

Derivada en 1D

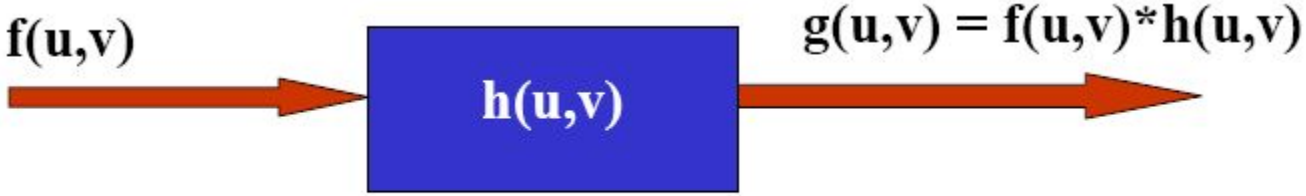
$$\frac{d^2f}{d^2x} \approx \frac{f(x+dx) - 2f(x) + f(x-dx)}{dx^2}$$

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+dx,y) - f(x,y)}{dx} \approx \frac{f(x+dx,dy) - f(x-dx,dy)}{2dx}$$

Derivada en 2D

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+dy) - f(x,y)}{dy} \approx \frac{f(x,y+dy) - f(x,y-dy)}{2dy}$$

CONVOLUCIÓN



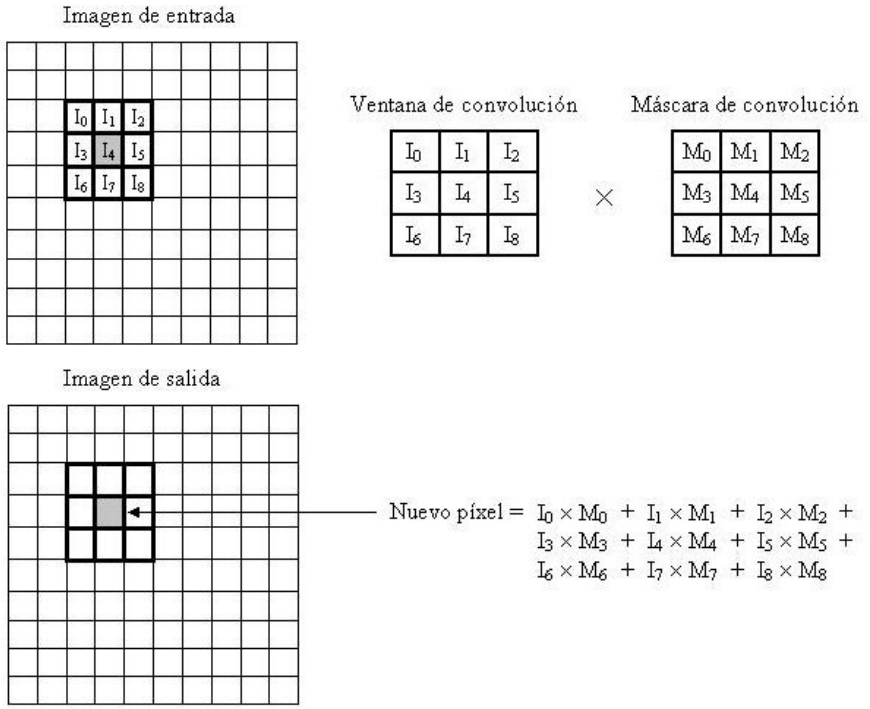
$$g(u,v) = h(u,v) * f(u,v) = \sum\sum f(i,j)h(u-i,v-j)$$

La convolución es una suma ponderada de píxeles en el vecindario del píxel a tratar. Los pesos son determinados por una pequeña matriz llamada máscara de convolución, que determina los coeficientes a aplicar sobre los puntos de vecindad.

f(u,v): Es la Imagen de entrada
h(u,v): Es la función respuesta al impulso del filtro a aplicar (o *máscara de convolución*).
g(u,v): Es la Imagen de salida.

CONVOLUCIÓN

La máscara o núcleo de convolución se centra sobre una ventana de igual tamaño y el resultado se le asigna al pixel central. Dependiendo del núcleo de convolución que apliquemos se obtendrán diferentes efectos sobre la imagen.



GRADIENTE

Convolución

$$\frac{\partial f(x,y)}{\partial x} \approx f(i+1,j) - f(i,j)$$

-1	1

$$\frac{\partial f(x,y)}{\partial y} \approx f(i,j+1) - f(i,j)$$

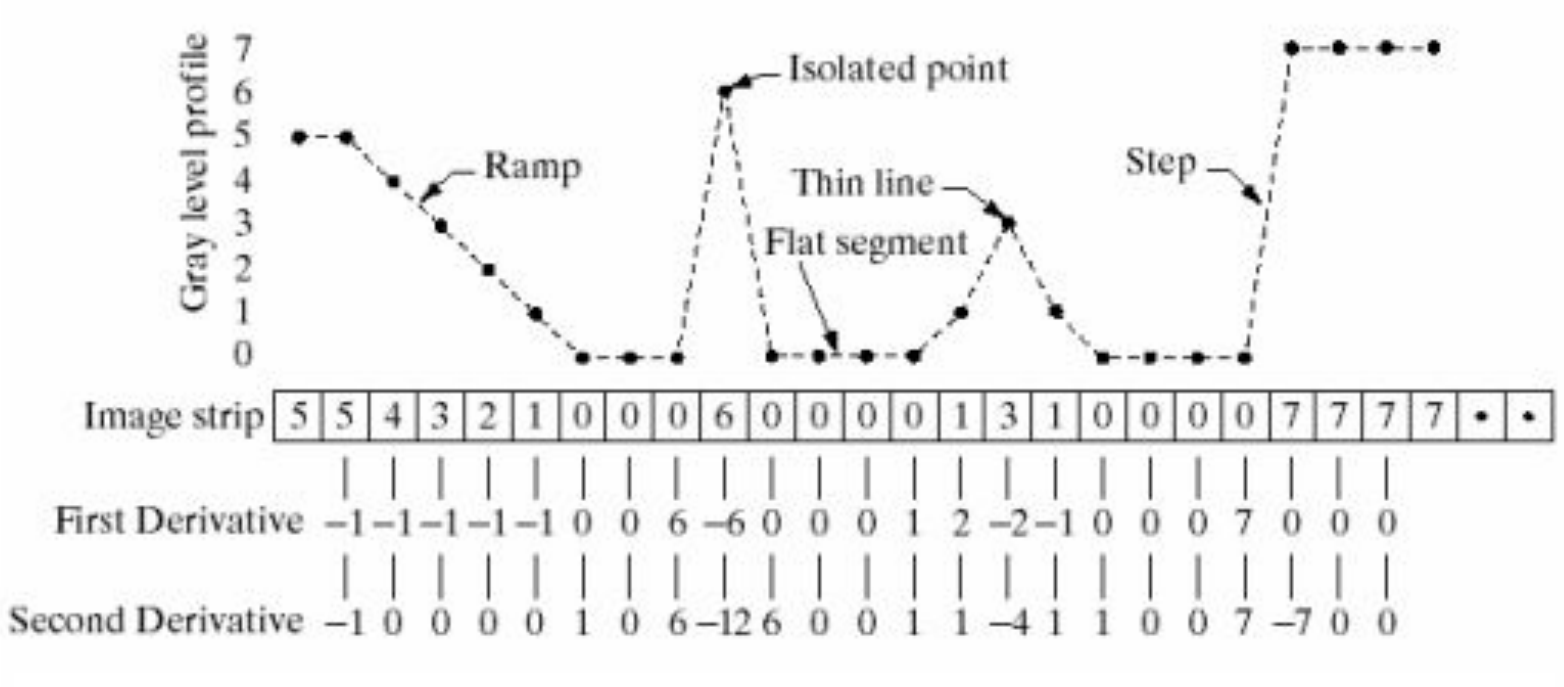
-1
1

$$\frac{\partial^2 f(x,y)}{\partial^2 x} \approx f(i+1,j) - 2f(i,j) + f(i-1,j)$$

$$\frac{\partial^2 f(x,y)}{\partial^2 y} \approx f(i,j+1) - 2f(i,j) + f(i,j-1)$$

1
-2
1

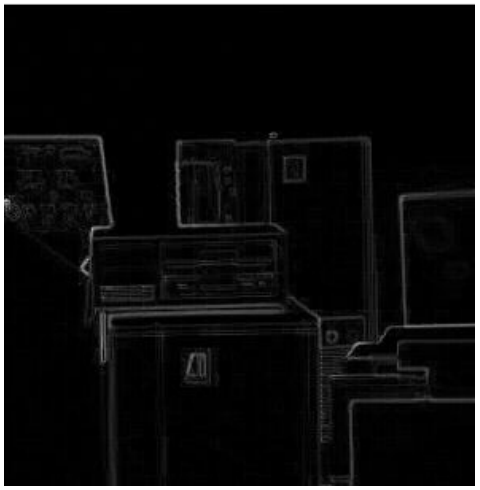
Operador Primera Derivada

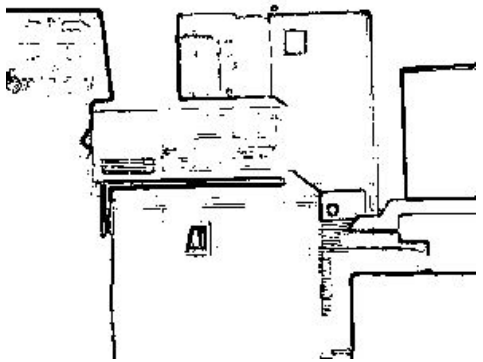


GRADIENTE

Como se visualiza en la ecuación de la magnitud del gradiente, es muy común que se aproxime al valor absoluto. Cuando se va a decidir si un punto es de borde o no, se le aplica una operación umbral, con umbral T a la imagen de la siguiente forma.

$$g(x,y) = \begin{cases} 1, & G[f(x,y)] > T \\ 0, & G[f(x,y)] \leq T \end{cases}$$





Operador Roberts

$$Mascara\mathbf{Roberts}\_Fila = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$Mascara\mathbf{Roberts}\_Columna = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

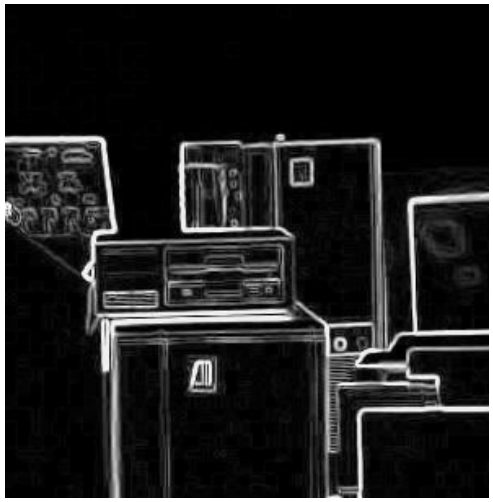
Este operador es buen detector de bordes, sin embargo es muy sensible al ruido por lo que sus prestaciones se ven altamente disminuidas.

Operador Prewitt

$$Mascara\mathbf{Prewitt}\_Fila = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$Mascara\mathbf{Prewitt}\_Columna = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Este operador es muy buen detector de bordes horizontales y verticales, además de presentar buena inmunidad al ruido.



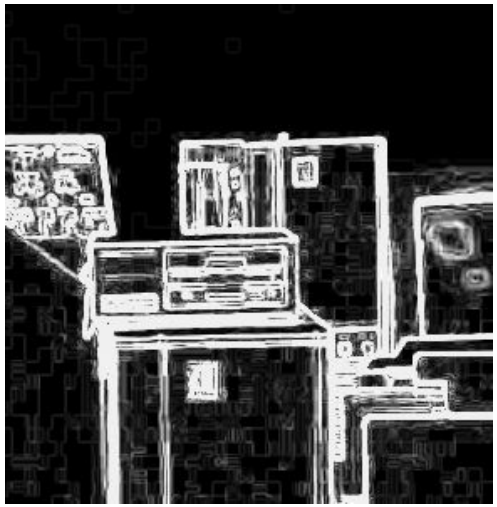
Operador Kirsch

$$Mascara\boldsymbol{Kirsch}\_0° = \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix}$$

$$Mascara\boldsymbol{Kirsch}\_45° = \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}$$

$$Mascara\boldsymbol{Kirsch}\_90° = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

Este operador también es llamado filtro brújula, porque a partir una mascara simple se obtienen 8 mascaras rotándola en las direcciones cardinales. (Norte, Nordeste, Oeste, Noroeste, Sur, Sureste, Este, Sudeste).

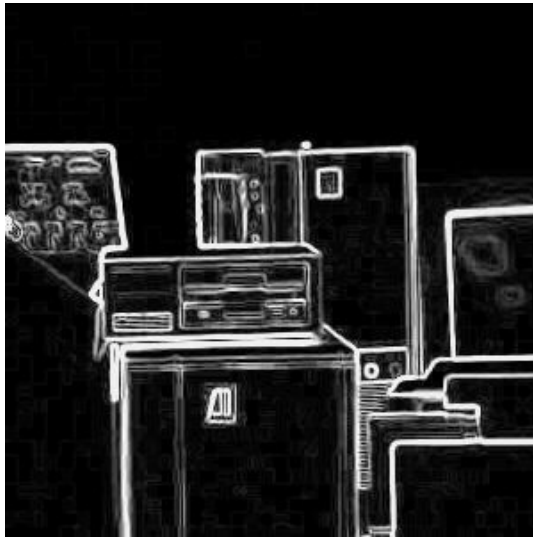


Operador Sobel

$$Mascara\boldsymbol{Sobel}\_Fila = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

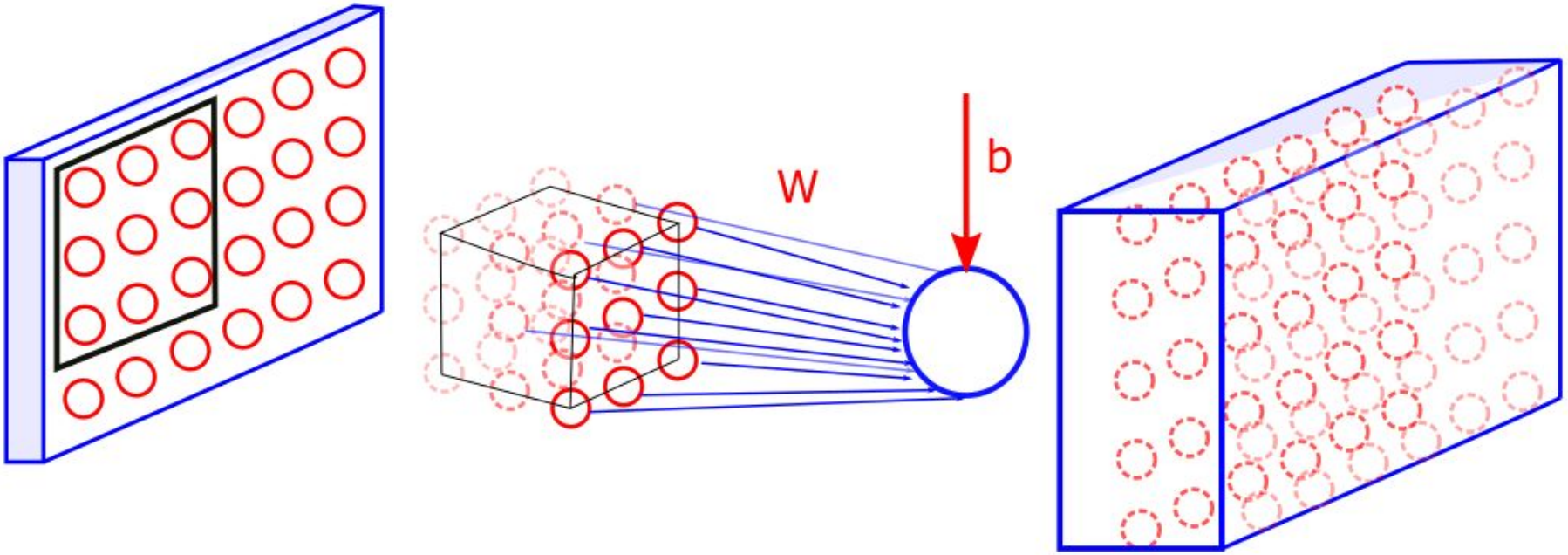
$$Mascara\boldsymbol{Sobel}\_Columna = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Este operador es un poco más sensible a los bordes diagonales que el operador Prewitt.



Capa Convolucional

La capa convolucional es el corazón de las CNN y permite convertir un volumen de entrada a un volumen de salida a partir de la aplicación de filtros convolucionales, donde cada valor de la matriz respuesta al impulso del filtro es un peso w de la neurona de conexión.



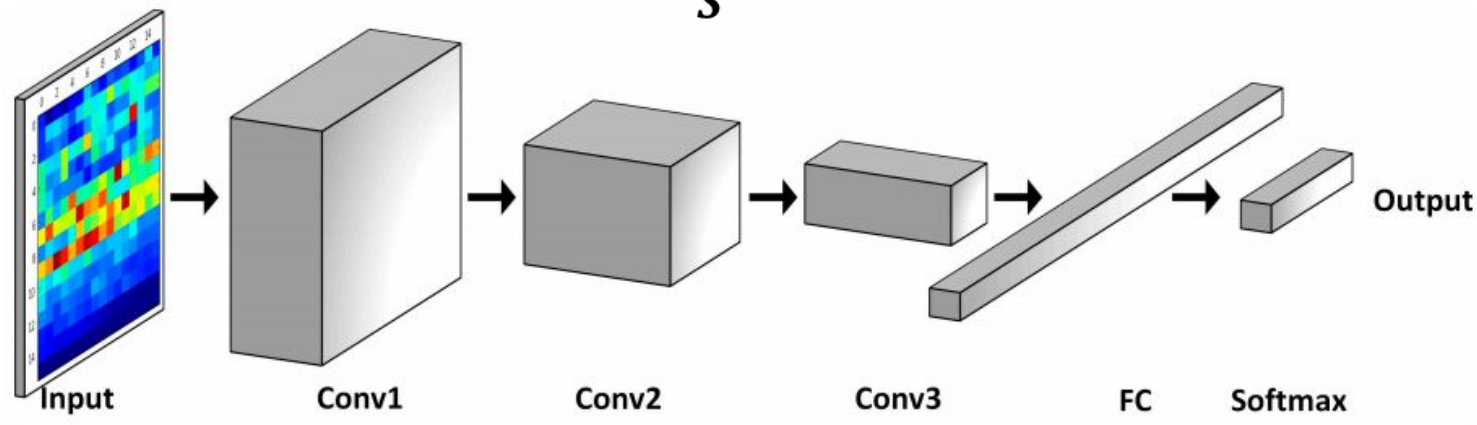
Capa Convolucional

Stride: Es el número de saltos en pixeles que da el filtro para aplicarse a la imagen completa.

Padding: Es el número de columnas y filas de borde que se le colocan al volumen de entrada con el fin de que la resolución espacial del volumen de salida sea igual al de entrada.

Depth: Profundidad de un volumen de entrada o salida

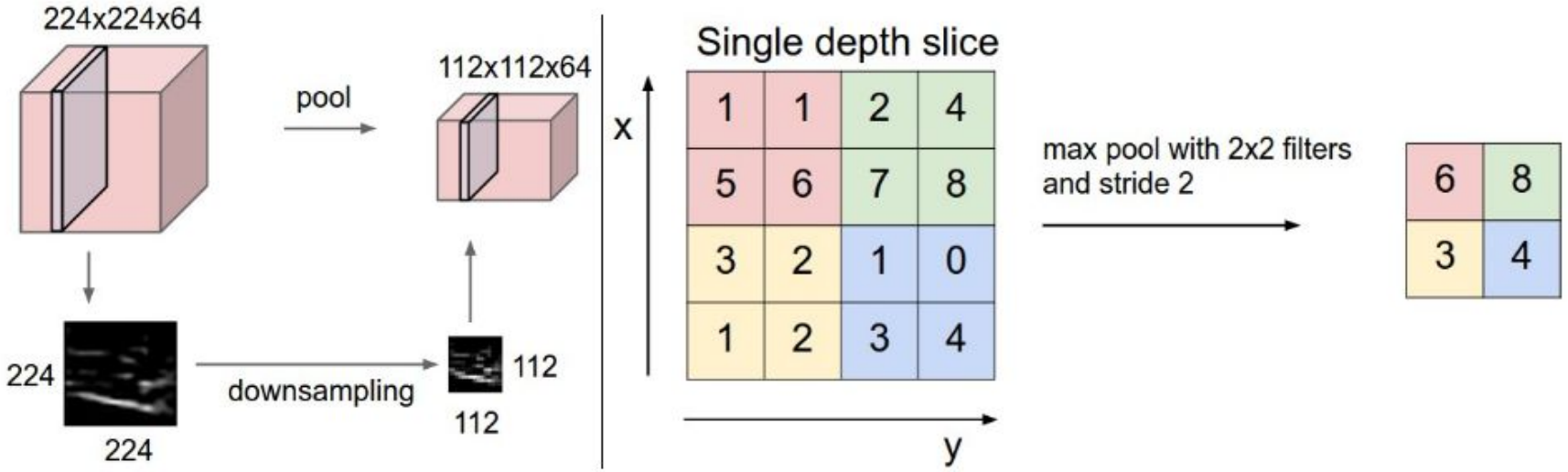
$$\frac{W - F + 2P}{S} + 1$$



http://www.mdpi.com/1424-8220/17/8/1729

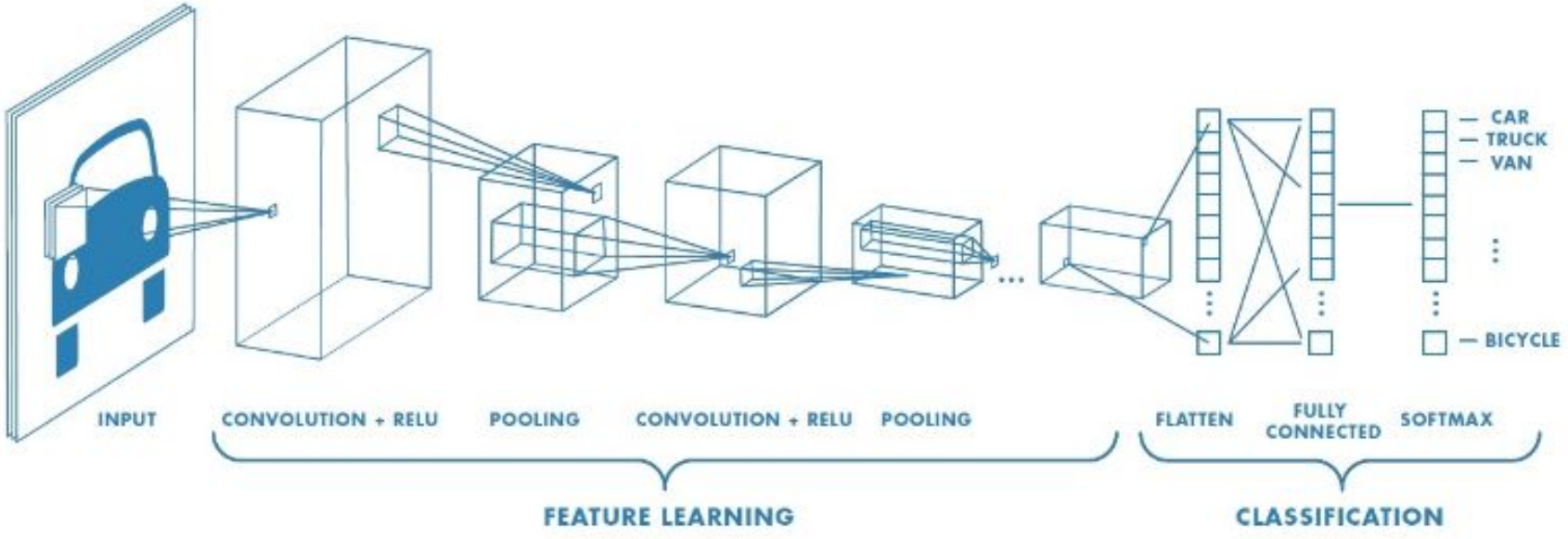
Capa Pooling

La capa pooling permite hacer una reducción espacial del volumen de salida de la capa convolucional, generando una representación de menor tamaño cada que se avanza en capas de mayor profundidad.



http://cs231n.github.io/convolutional-networks/#overview

ARQUITECTURAS



https://de.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

ARQUITECTURAS

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

```
INPUT -> FC
```

Clasificador Lineal

```
INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC
```

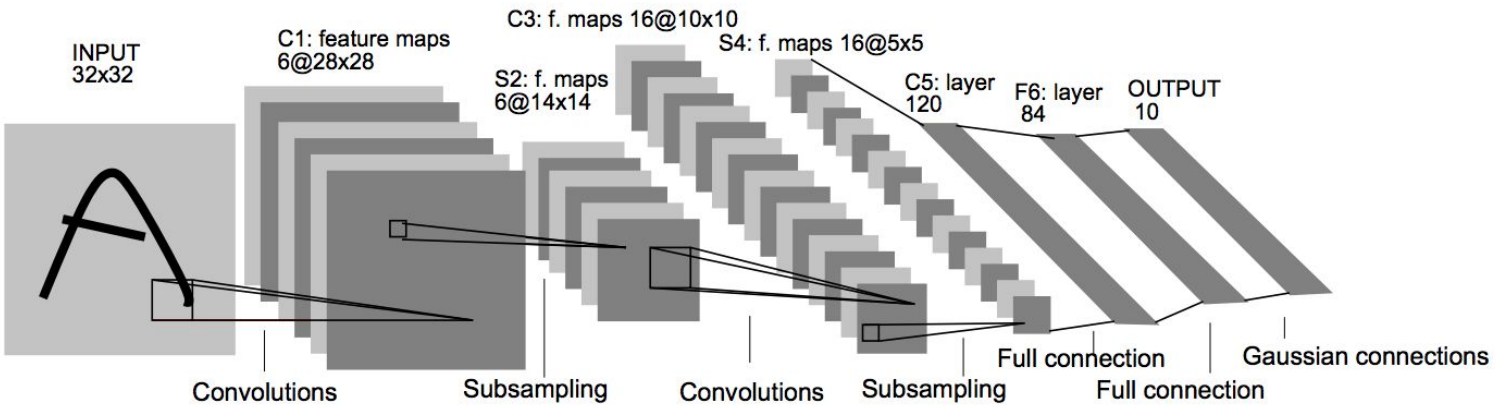
2 capas convolucionales

```
INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC
```

Capas más profundas para extraer características más complejas

Arquitecturas Comunes

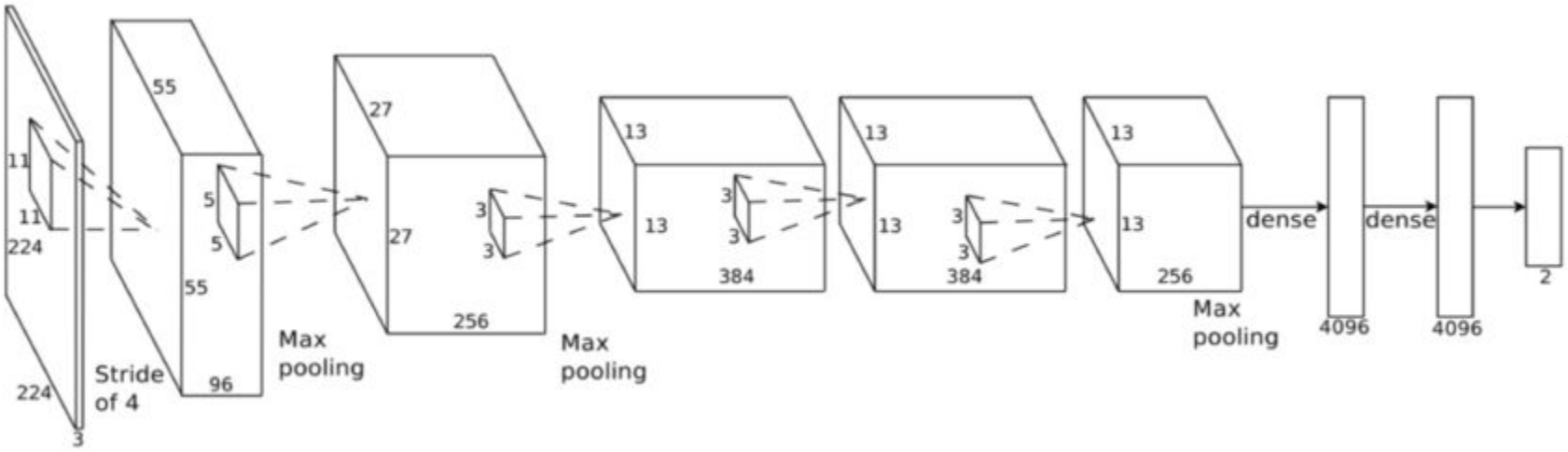
LeNet-5 - 1998



60.000 parámetros

Arquitecturas Comunes

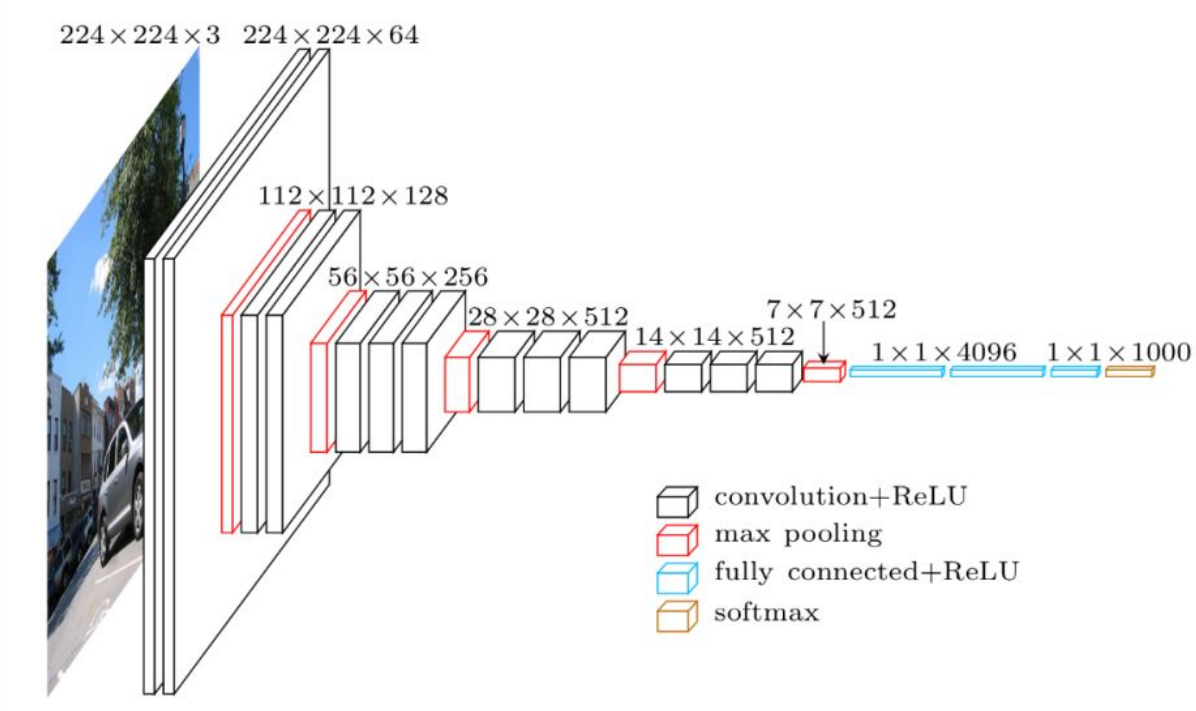
AlexNet - 2012



60.000.000 parámetros

Arquitecturas Comunes

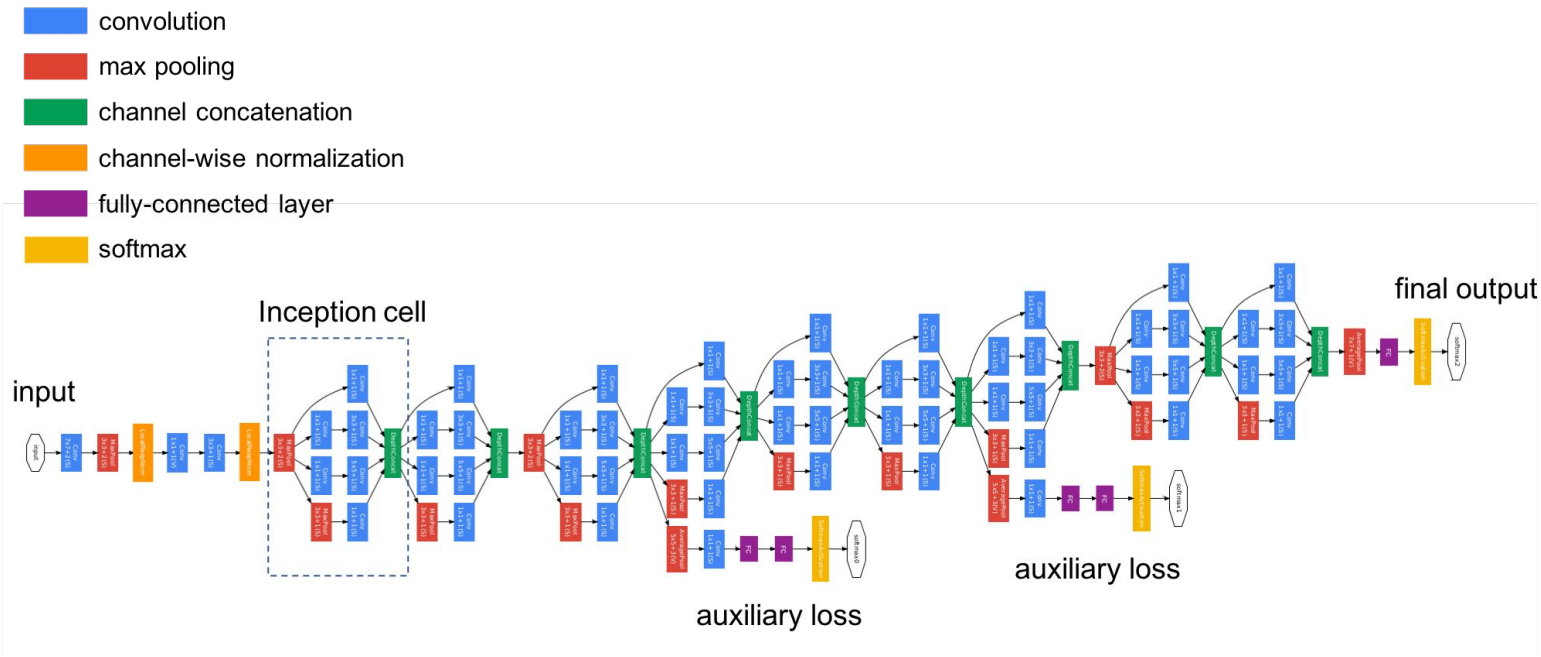
VGG-16 - 2014



138.000.000 parámetros

Arquitecturas Comunes

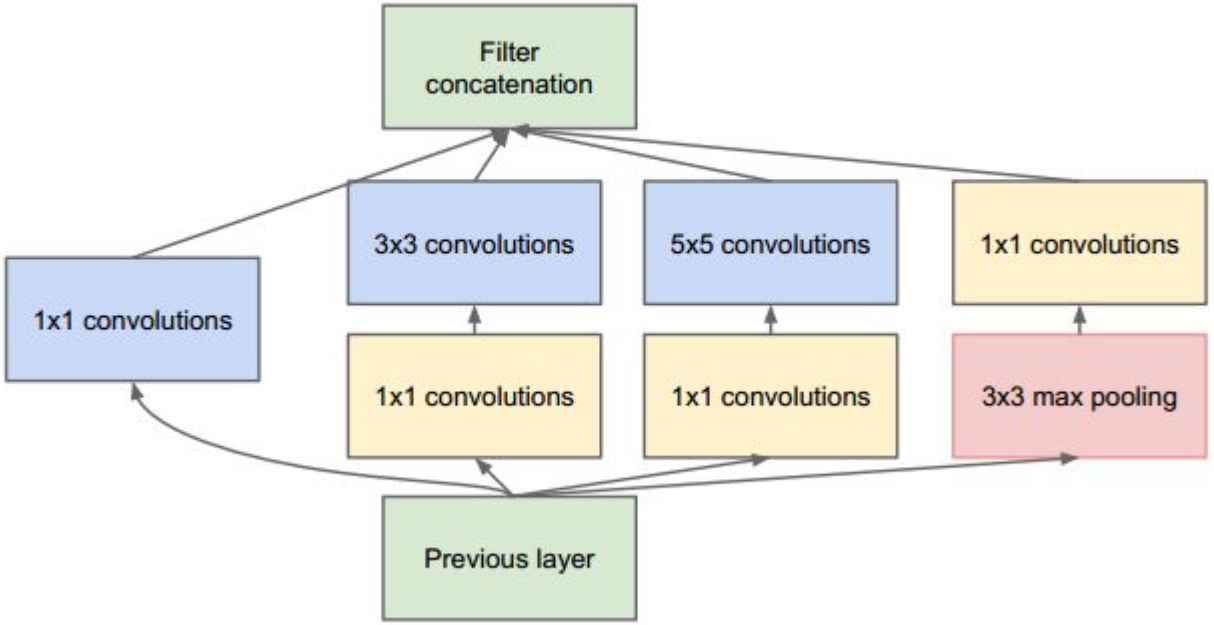
Inception (GoogLeNet) - 2014



5.000.000 parámetros

Arquitecturas Comunes

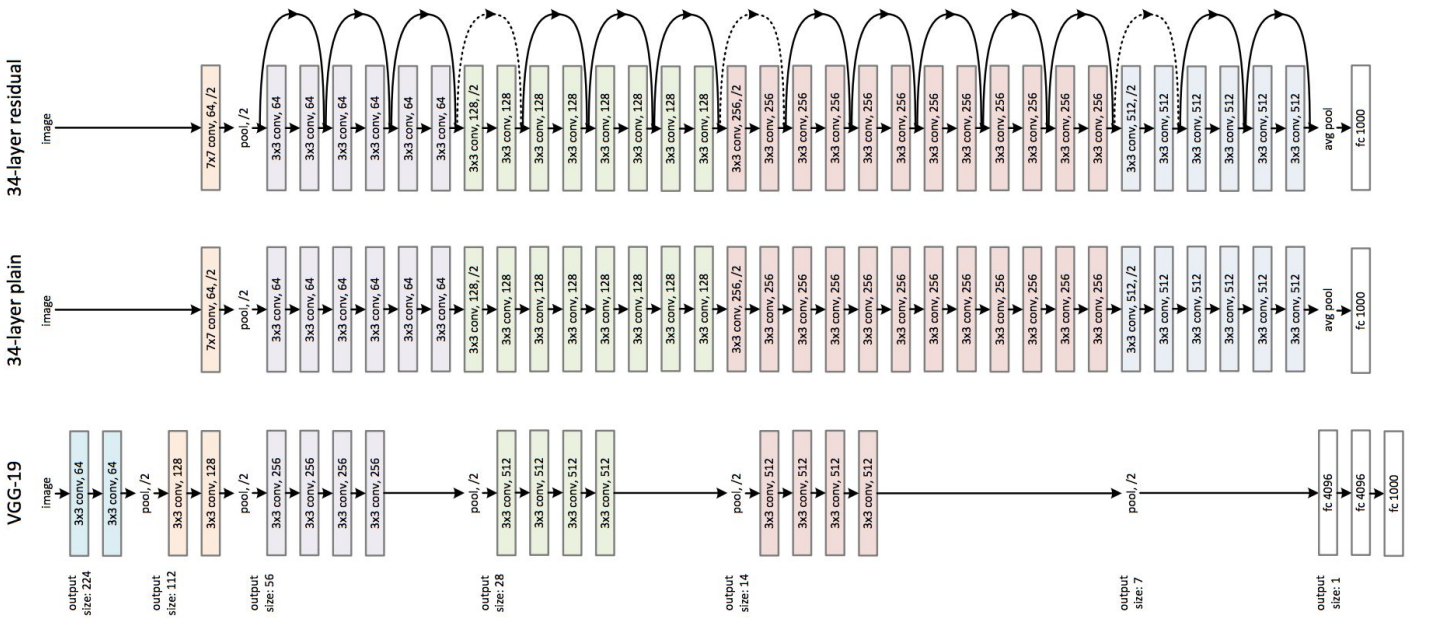
Inception (GoogLeNet) - 2014



5.000.000 parámetros

Arquitecturas Comunes

Resnet



Funciones en Tensorflow

Convolución:

```
tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=True, data_format='NHWC',
dilations=[1, 1, 1, 1], name=None)
```

input: Tensor 4-D de entrada del tipo: half, bfloat16, float32, float64.
filter: Tensor 4-D de tamaño [filter_height, filter_width, in_channels, out_channels]
strides: Tensor 1-D de longitud 4. Se convierte en el paso para deslizar la ventana para cada dimensión de entrada.

padding: String con valores: "SAME“ aplica el zero-padding necesario para que la resolución espacial del volumen de salida sea igual al de entrada, "VALID“ no se aplica zero-padding.
use_cudnn_on_gpu: Bandera opcional, por defecto es True
data_format: String opcional que selecciona el formato de los datos: "NHWC", "NCHW"
Dilatation_rate Factor de dilatación para cada dimensión de entrada.
name: Nombre de la operación. Es opcional

Relu:

```
tf.nn.relu(features,
name=None)
```

input: Tensor de tipo float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64, qint8.
name: Nombre de la operación. Es opcional

Funciones en Tensorflow

Max Pool:

```
tf.nn.max_pool(value, ksize, strides, padding,
data_format='NHWC', name=None)
```

value: Tensor 4-D de formato “data_format”.
ksize: Una lista o tupla de 4 enteros que representa el tamaño de la ventana para cada dimensión del tensor de entrada
strides: Tensor 1-D de longitud 4. Se convierte en el paso para deslizar la ventana para cada dimensión de entrada.
padding: String con valores: "SAME“ aplica el zero-padding necesario para que la resolución espacial del volumen de salida sea igual al de entrada, "VALID“ no se aplica zero-padding.
data_format: String opcional que selecciona el formato de los datos: "NHWC", "NCHW"
name: Nombre de la operación. Es opcional

Flatten:

```
tf.contrib.layers.flatten(inputs,
outputs_collections=None, scope=None )
```

input: Tensor de tamaño [batch_size, …]
scope: Nombre del scope. Es opcional

Retorna: Tensor flatten de la forma [batch_size, …]

Funciones en Tensorflow

Fully Connected:

```
tf.contrib.layers.fully_connected(inputs, num_outputs, activation_fn=tf.nn.relu,
normalizer_fn=None, normalizer_params=None,
weights_initializer=initializers.xavier_initializer(), weights_regularizer=None,
biases_initializer=tf.zeros_initializer(),
biases_regularizer=None, reuse=None, variables_collections=None, outputs_collections=None,
trainable=True, scope=None
)
```

inputs: Tensor de tamaño [batch_size, depth].
Num_ouputs: Número de unidades de la capa de salida
Activation_fn: función de activación, por defecto es RELU. Si el valor es None, la salida es el nivel de activación lineal