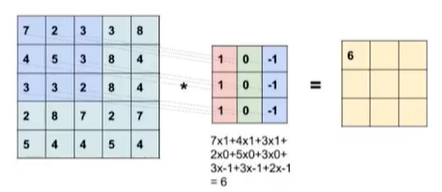
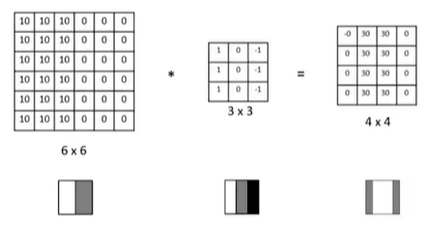
El kernel es una matriz que se desliza desde la esquina superior izquierda hasta la esquina inferior derecha, paso por paso, hasta terminar toda la imagen, haciendo una operación matemática llamada convolución.

CONVOLUCION



Detectar bordes verticales



import numpy as np

import matplotlib.pyplot as plt

import scipy.ndimage as nd

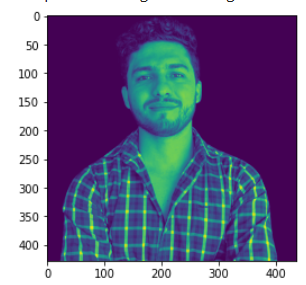
from skimage import  io, color

img = io.imread('yo.png')

img\_gray = color.rgb2gray(img)

img\_gray.shape

plt.imshow(img\_gray)



kernel = np.array([[-1,0,1],

[-1,0,1],

[-1,0,1]])

kernel = np.array([[ -1, -1, -1],

[ 0, 0, 0],

[ 1, 1, 1]])

kernel = np.array([[ 0, 0, 0],

[ 0, 1, 0],

[ 0, 0, 0]])

kernel = np.array([[ 0, 0, 0],

[ 0, 0, 0],

[ 0, 0, 0]])

APLICANDO

img\_new = nd.convolve(img\_gray, kernel)

fig, axes = plt.subplots(1,2, figsize = (15,10))

axes[0].imshow(img\_gray, cmap=plt.cm.gray)

axes[0].set\_title('Original')

axes[0].axis('off')

axes[1].imshow(img\_new, cmap=plt.cm.gray)

axes[1].set\_title('Convolution')

axes[1].axis('off')



Kernel bordes horizontales.

img\_new\_horizontal = nd.convolve(img\_gray,kernel.T)

def test\_kernels(img: np.ndarray, kernel: np.ndarray):

img\_new = nd.convolve(img\_gray, kernel)

fig, axes = plt.subplots(1,2, figsize=(15,10))

axes[0].imshow(img, cmap=plt.cm.gray)

axes[0].set\_title('Original')

axes[0].axis('off')

axes[1].imshow(img\_new, cmap=plt.cm.gray)

axes[1].set\_title('New')

axes[1].axis('off')

test\_kernels(img\_gray, kernel)

# test con matriz identidad

test\_kernels(img\_gray, [[1,0,0],[0,1,0],[0,0,1]])

Que ocurre con el primer pixel? solo se le toma una sola vez.

De ahi la necesidad de Padding y Strides.

**PADDING.**

Aumentamos la imagen con '0's alrededor. Asi (1,1) no pasara 1 vez.

**STRIDES**

Se mueve de 1 en 1, y si

Si kernes de 2x2 y la imagen de 6x6 y striades=1 sale imagen de 5x5

Si kernel de 2x2 y la imagen de 6x6 y strides=2 sale imagen 3x3

paddling=

same(tamano imagen = tamano salida, agrega '0'

valid, no aplica paddling, se reduce imagen

kernel\_size = tamano kernel

strides= desplaza vertical, desplaza vertical

tf.keras.layers.Conv2D(

filters,

kernel\_size,

strides=(1,1),

padding=['valid','same'])

paddling=

same(tamano imagen = tamano salida, agrega '0'

valid, no aplica paddling, se reduce imagen

kernel\_size = tamano kernel

strides= desplaza vertical, desplaza vertical

PROBLEMA

- MUCHOS FILTROS EN CADA CAPA, RED MAS COMPLEJA

- MUCHOS PARAMETROS, MAS COMPLEJA

- MAS PROCESAMIENTO

POOLING: otro kernel, matriz de 3x3

Despues de la convolucion

- Saca el maximo y el promedio, eliminando ruido

- Toma de la matriz 3x3 y toma el maximo y el promedio

- Max comun el es max pooling, rara vez el average pooling

Parametros = Filtrox x tamano kernel x profundidad de la capa anterior + Filtros (bias)1

Para calcular las dimensiones de la red luego de aplciar la convolucion se usa

Nout = ( Nin + 2p -k) /s +1

Nin: No. input features

Nout: no output features

k: kernel size

p: padding size

s: stride size

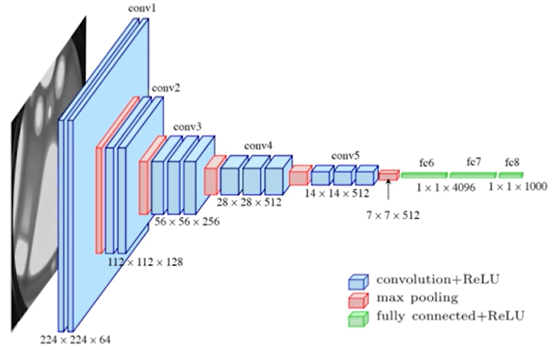
Una imagen 28x28, 1 filtro de 3x3, padding='same' stride=1 y 16 canales de salida

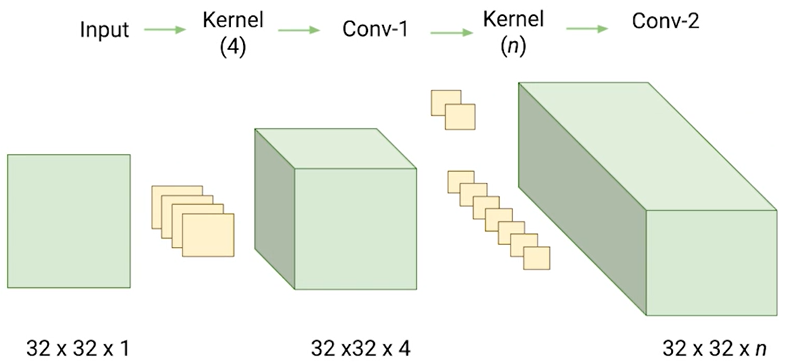
n eje x = (28 + 2(10 -3)/1 +1 = 28

n eje y = (28 + 2(10 -3)/1 +1 = 28

Las dimensiones de salida 28 x 28 x 16

<https://alexlenail.me/NN-SVG/index.html>





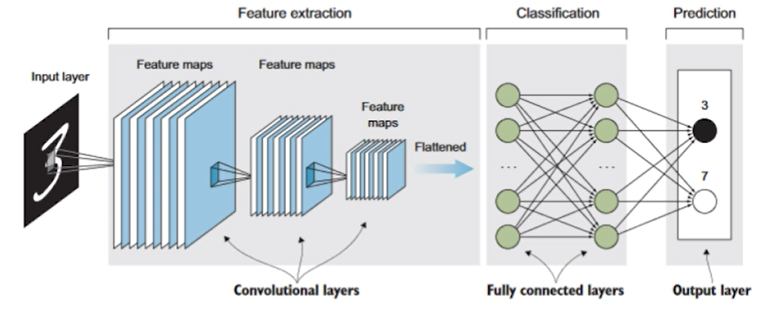
ARQUITECTURA, CAPAS

- CONVOLUCION, AUMENTA

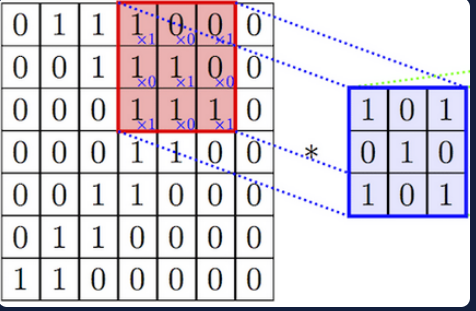
- MAX POOLING REDUCE

- FLATTEN PASA TENSOR A VECTOR

- APLICAMOS REDES DENSE PARA CLASIFICACION



Un kernel o filtro de este estilo: [[-1,0,1], [-1,0,1], [-1,0,1]] Sería capaz de identificar bordes de tipo: VERTICAL



Resultado de la convolución? 4

¿En una imagen de entrada con dimension de

8x8, un filtro de

2x2 y stride de 2 cual seria la salida?

(teniendo en cuenta que no hay padding)

4x4