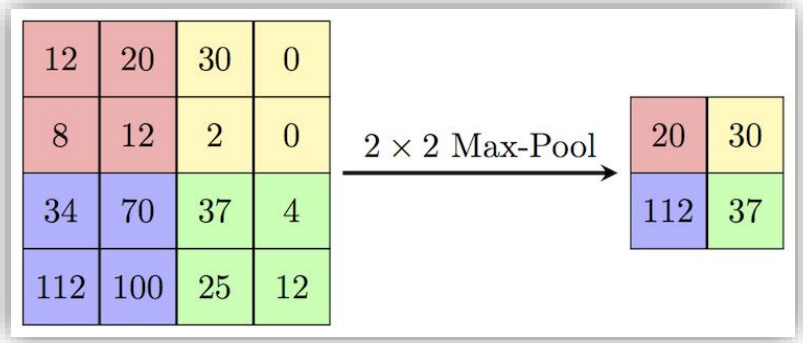
Semilleros
by DSRP

Maximum Pooling



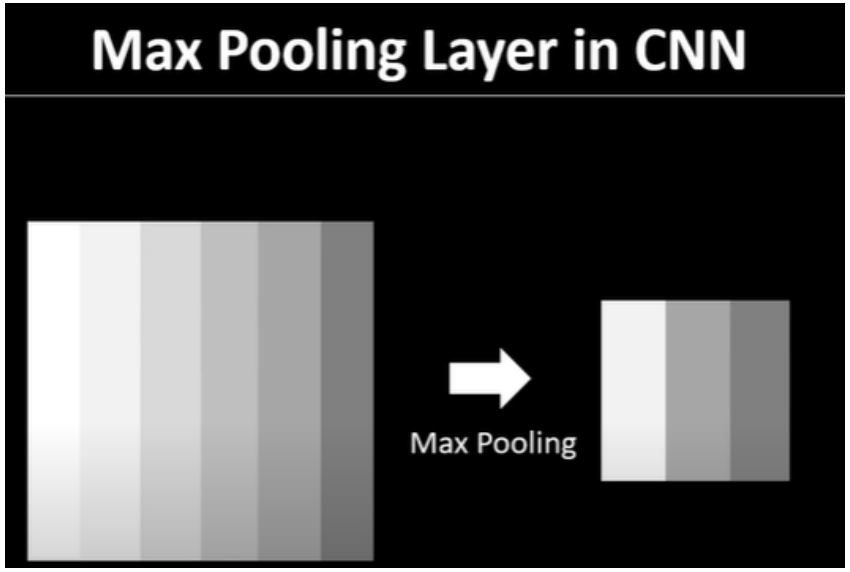
Semilleros by DSRP

Lesson: Review

El **pooling** es una operación que permite analizar el contenido de una imagen por regiones (o bloques) para extraer la información más representativa de las mismas.

- El **max pooling** permite reducir la cantidad de datos entre una capa y otra, facilitando así el procesamiento de las imágenes y el entrenamiento de la red, pero a la vez preservando la información más relevante.

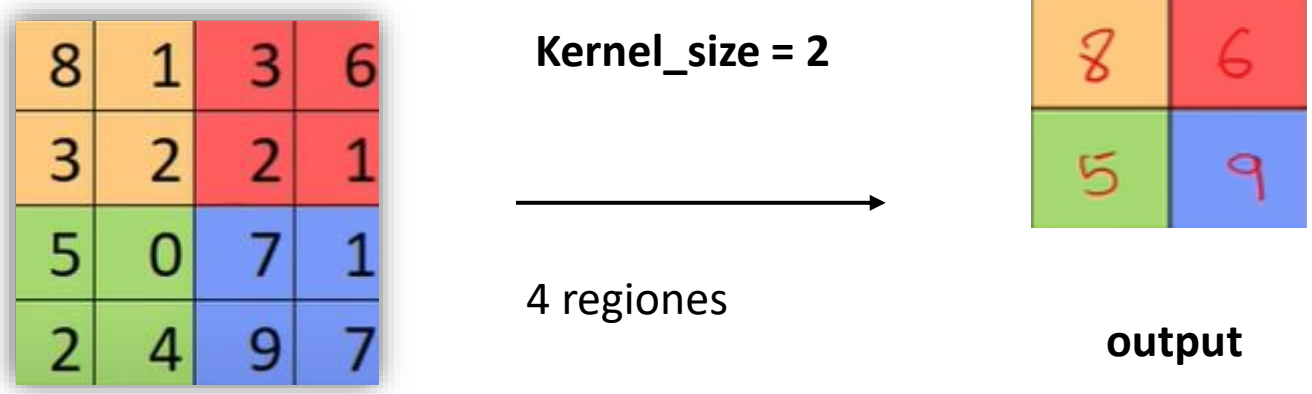
- En el **max pooling** la imagen es dividida en regiones del mismo tamaño, y para cada región se extrae simplemente el valor máximo que corresponderá a un pixel en la imagen resultante.

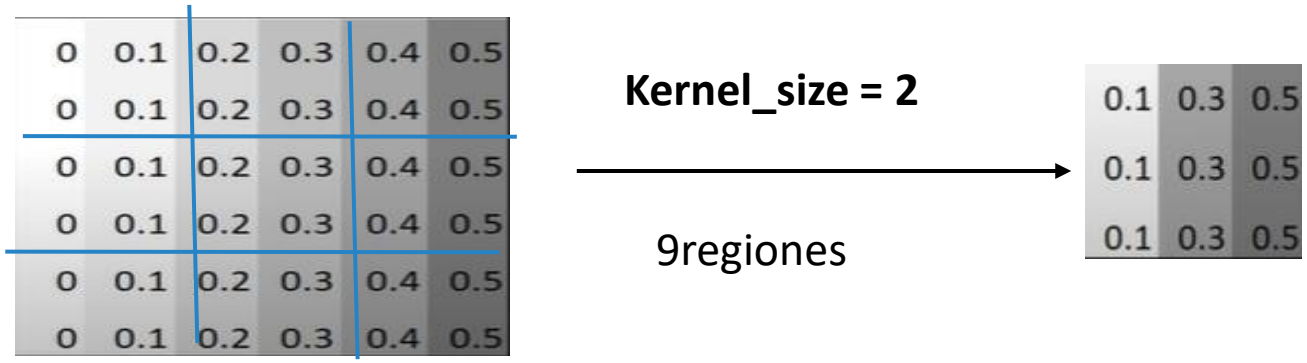


Semilleros by DSRP

Lesson 0: Introduction

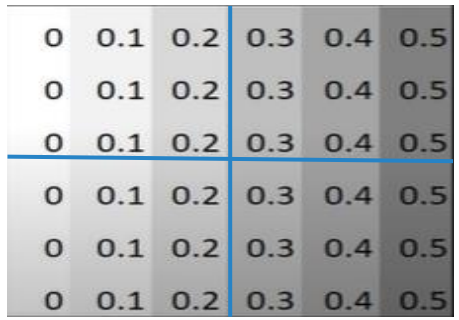
En el **max pooling** la imagen es dividida en regiones del mismo tamaño, y para cada región se extrae simplemente el valor máximo que corresponderá a un pixel en la imagen resultante.





https://www.codificandobits.com/blog/padding-strides-maxpooling-stacking-redes-convolucionales/

Kernel_size = 3



input

4 regiones

0.2	**0.5**
0.2	0.5

output

Lesson 3.1: Condense with Maximum Pooling

Agregar paso de condensación al modelo que teníamos antes, nos dará esto:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Conv2D(filters=64, kernel_size=3), # activation is None
    layers.MaxPool2D(pool_size=2),
    # More layers follow
])
```

Semilleros by DSRP

Example – Apply Maximum Pooling

Agreguemos el paso "**condense**" a la extracción de características que hicimos en el ejemplo de la Lección 2.

Usaremos otra de las funciones en ***tf.nn*** para aplicar el paso de agrupación, **tf.nn.pool**. Esta es una función de Python que hace lo mismo que la capa MaxPool2D que usa cuando construye un modelo, pero, al ser una función simple, es más fácil de usar directamente.

```
import tensorflow as tf

image_condense = tf.nn.pool(
    input=image_detect, # image in the Detect step above
    window_shape=(2, 2),
    pooling_type='MAX',
    # we'll see what these do in the next lesson!
    strides=(2, 2),
    padding='SAME',
)

plt.figure(figsize=(6, 6))
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.show();
```

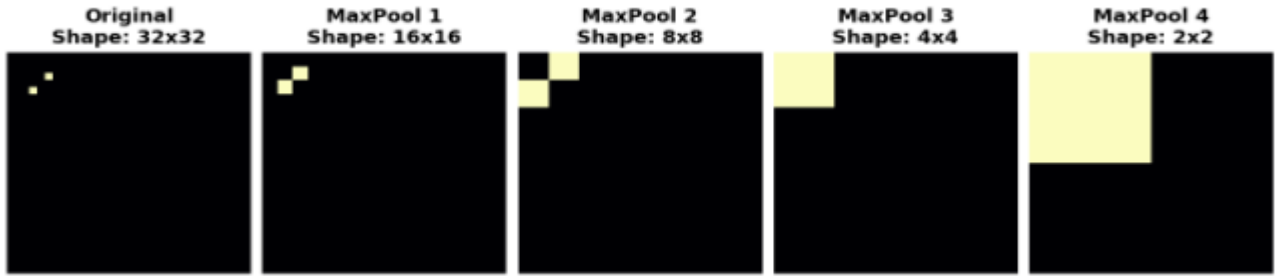
puede ver cómo el **pooling step** (paso de agrupación) pudo intensificar la función al condensar la imagen alrededor de los píxeles más activos.

Example – Translation Invanriance

Llamamos a los píxeles cero "sin importancia". ¿Significa esto que no tienen ninguna información?

De hecho, los píxeles cero llevan información posicional. El espacio en blanco aún coloca la característica dentro de la imagen. Cuando MaxPool2D elimina algunos de estos píxeles, elimina parte de la información de posición en el mapa de características. Esto le da a un convnet una propiedad llamada **translation invariane** (invariancia de traducción). Esto significa que un convnet con una agrupación máxima tenderá a no distinguir las características por su ubicación en la imagen. ("Translation" es la palabra matemática para cambiar la posición de algo sin rotarlo o cambiar su forma o tamaño).

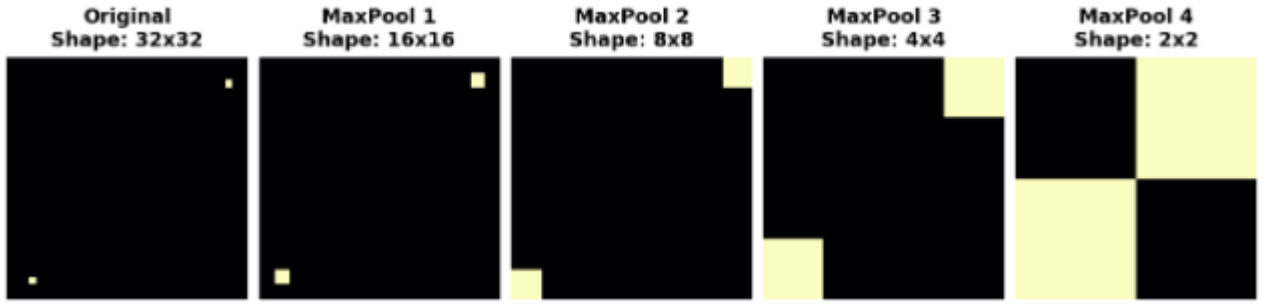
Observe lo que sucede cuando aplicamos repetidamente la maximum pooling al siguiente feature map .



Example – Translation Invanriance

Los dos puntos de la imagen original se volvieron indistinguibles después de repetidos agrupamientos. En otras palabras, la agrupación destruyó parte de su información posicional. Dado que la red ya no puede distinguir entre ellos en los mapas de características, tampoco puede distinguirlos en la imagen original: se ha vuelto invariable a esa diferencia de posición.

De hecho, la agrupación solo crea invariancia de traducción en una red en distancias pequeñas, como ocurre con los dos puntos de la imagen. Las características que comienzan muy separadas seguirán siendo distintas después de la agrupación; sólo se perdió parte de la información posicional, pero no toda.

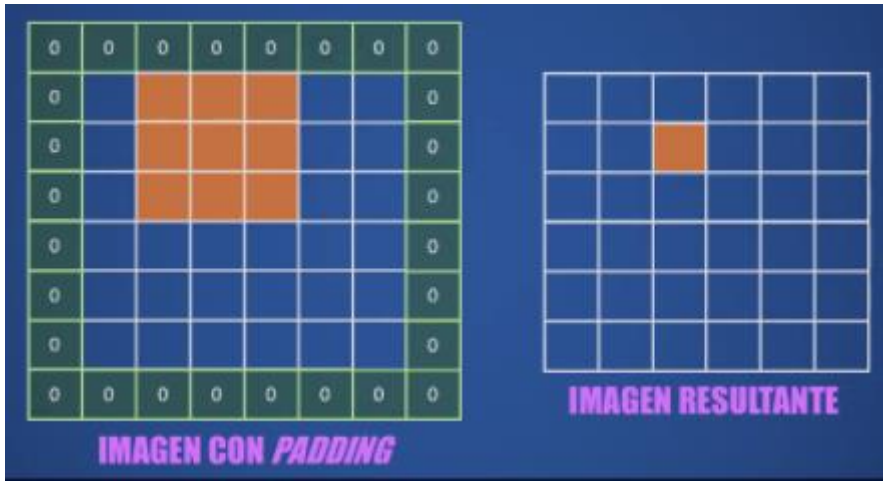


Esta invariancia a pequeñas diferencias en las posiciones de las características es una propiedad agradable para un clasificador de imágenes. Solo debido a las diferencias en la perspectiva o el encuadre, el mismo tipo de característica podría colocarse en varias partes de la imagen original, pero aún nos gustaría que el clasificador reconociera que son iguales. Debido a que esta invariancia está integrada en la red, podemos usar muchos menos datos para el entrenamiento: ya no tenemos que enseñarle a ignorar esa diferencia. Esto le da a las redes convolucionales una gran ventaja de eficiencia sobre una red con solo capas densas.

LABORATORIO 3

Semilleros
by DSRP

The Sliding Window



Semilleros by DSRP

Lesson 0: Sliding window

Sliding window: En el contexto de computer visión (y como su nombre indica), una ventana deslizante es una región rectangular de ancho y alto fijos que se "desliza" a través de una imagen, como en la siguiente figura: Figura 1: Ejemplo de deslizamiento de una ventana enfoque, donde deslizamos una ventana de izquierda a derecha y de arriba a abajo.

Estas técnicas, aunque sencillas, juegan un papel absolutamente crítico en la detección de objetos y la clasificación de imágenes.

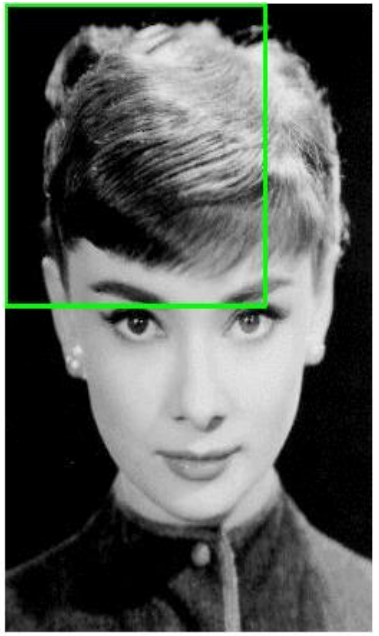
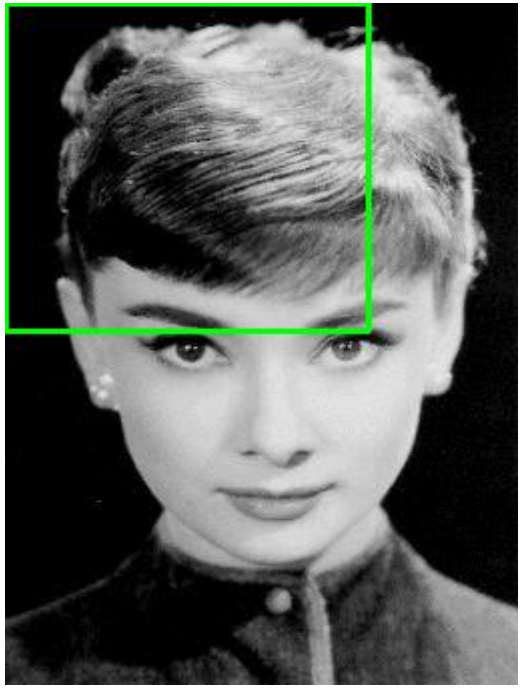


Figure 1: Example of the sliding a window approach, where we slide a window from left-to-right and top-to-bottom.



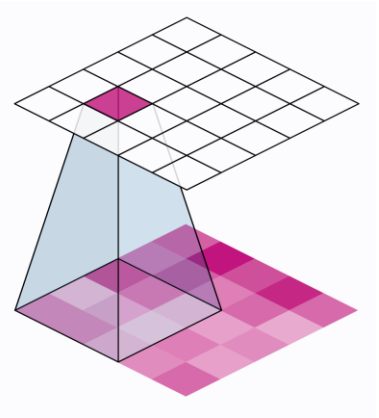
https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

Semilleros by DSRP

Lesson: Introduction

En las dos lecciones anteriores, aprendimos sobre las tres operaciones que llevan a cabo la extracción de características de una imagen:

- filtro con una capa de **convolución**
- detectar con activación **ReLU**
- condensar con una capa de **maximum pooling**



Las operaciones de convolución y pooling comparten una característica común: ambas se realizan sobre una sliding window (ventana deslizante). Con convolución, esta "ventana" viene dada por las dimensiones del kernel, el parámetro kernel_size. Con la agrupación, es la ventana de agrupación, dada por pool_size.

Hay dos parámetros adicionales que afectan tanto a las capas de convolución como a las de agrupación: estos son los *strides* de la ventana y si se debe usar padding (relleno) en los bordes de la imagen.

El parámetro strides dice qué tan lejos debe moverse la ventana en cada paso, y el parámetro padding describe cómo manejamos los píxeles en los bordes de la entrada.

Con estos dos parámetros, definir las dos capas se convierte en:

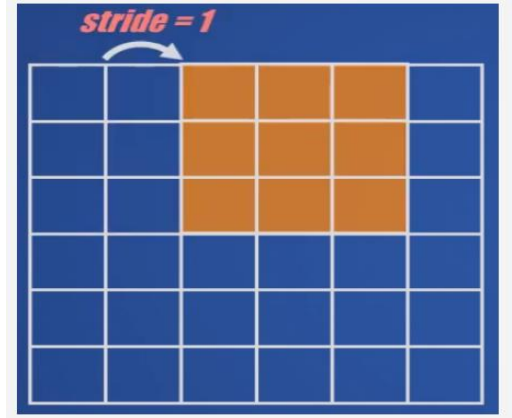
```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Conv2D(filters=64,
                  kernel_size=3,
                  strides=1,
                  padding='same',
                  activation='relu'),
    layers.MaxPool2D(pool_size=2,
                     strides=1,
                     padding='same')
    # More layers follow
])
```

Semilleros by DSRP

Lesson 4.1: Stride

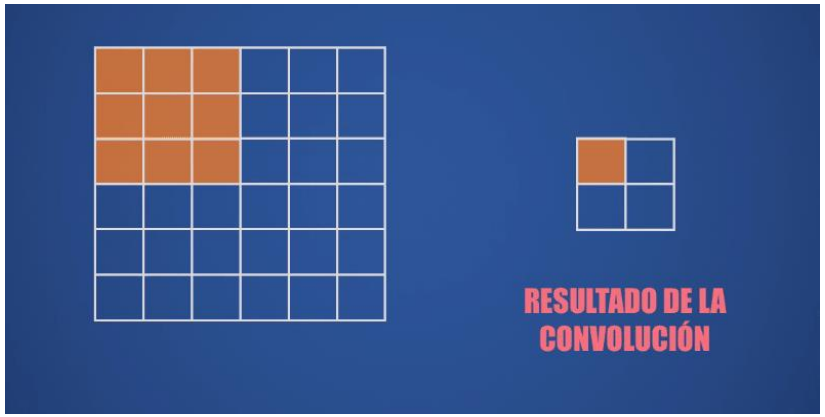
Stride:

En la convolución original, el kernel se desplaza un pixel hacia la derecha, o hacia abajo, durante cada iteración. Cada uno de estos desplazamientos se conoce como *stride* (o salto).



En la convolución original el 'stride' es igual a 1

Sin embargo, también es posible realizar la convolución con *strides* mayores, lo que permite obtener imágenes resultantes **de menor tamaño** en comparación con las obtenidas en la convolución original:



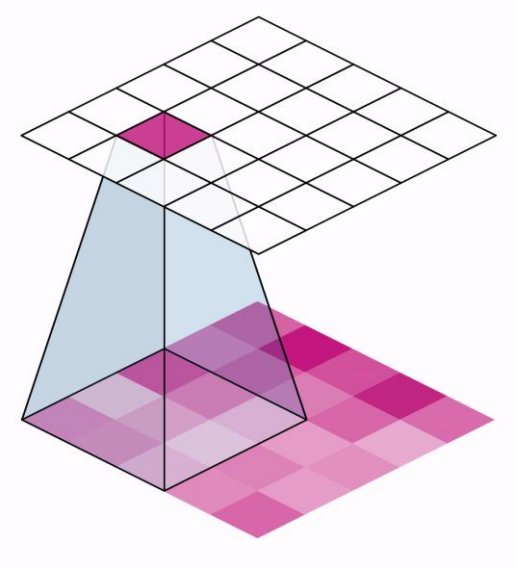
Al usar un 'stride' mayor que 1, el tamaño de la imagen resultante se reduce, pasando por ejemplo de 4x4 con un stride de 1 a 2x2 con un 'stride' de 2

En las redes convolucionales esto es útil, pues reduce la cantidad de datos a procesar entre una y otra capa de la red.

https://www.codificandobits.com/blog/padding-strides-maxpooling-stacking-redes-convolucionales/

Lesson 4.1: Stride

Necesitamos especificar **stride** en ambas dimensiones de la imagen: una para moverse de izquierda a derecha y otra para moverse de arriba a abajo. Esta animación muestra strides = (2, 2), un movimiento de 2 píxeles en cada paso.



¿Qué efecto tiene stride? Siempre que el paso en cualquier dirección sea mayor que 1, la ventana deslizante saltará algunos de los píxeles de la entrada en cada paso.

Debido a que queremos utilizar características de alta calidad para la clasificación, las capas convolucionales suelen tener pasos = (1, 1). Aumentar el paso significa que perdemos información potencialmente valiosa en nuestro resumen. Sin embargo, las capas de **maximum pooling** casi siempre tendrán valores de **strides** superiores a 1, como (2, 2) o (3, 3), pero no más grandes que la ventana en sí.

Finalmente, tenga en cuenta que cuando el valor de los strides es el mismo número en ambas direcciones, solo necesita establecer ese número; por ejemplo, en lugar de strides = (2, 2), puede usar **strides = 2** para la configuración de parámetros.

Lesson 4.2: Padding

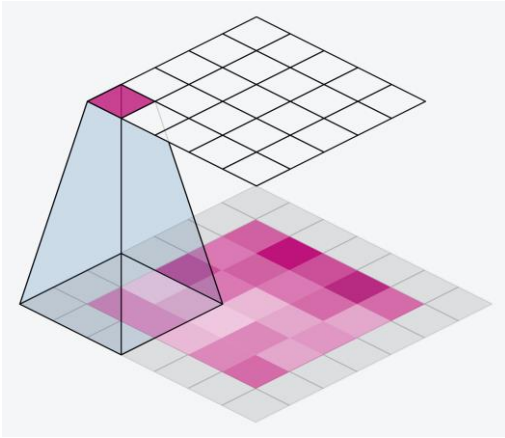
Al realizar el cálculo de la ventana deslizante, existe la duda de qué hacer en los límites de la entrada. Permanecer completamente dentro de la imagen de entrada significa que la ventana nunca se asentará directamente sobre estos píxeles de límite como lo hace con cualquier otro píxel de la entrada. Dado que no estamos tratando todos los píxeles exactamente de la misma manera, ¿podría haber algún problema?

Lo que hace la convolución con estos valores límite está determinado por su parámetro de relleno. En TensorFlow, tiene dos opciones: **padding = 'same' o padding = 'valid**'. Hay compensaciones con cada uno.

- Cuando establecemos padding = 'valid', la ventana de convolución permanecerá completamente dentro de la entrada. El inconveniente es que la salida se encoge (pierde píxeles) y se encoge más para núcleos más grandes. Esto limitará la cantidad de capas que puede contener la red, especialmente cuando las entradas son pequeñas.

- La alternativa es usar padding = 'same'. El truco aquí es rellenar la entrada con ceros alrededor de sus bordes, usando solo suficientes ceros para hacer que el tamaño de la salida sea el mismo que el tamaño de la entrada. Sin embargo, esto puede tener el efecto de diluir la influencia de los píxeles en los bordes.

La siguiente animación muestra una ventana deslizante con “same" relleno.



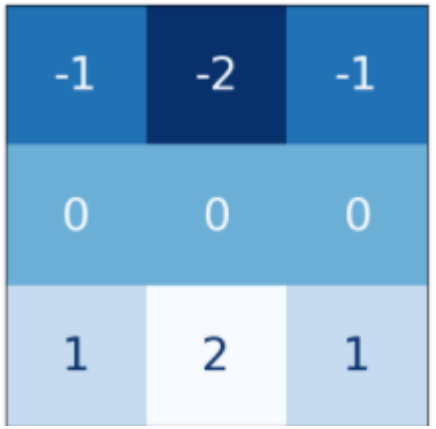
https://www.codificandobits.com/blog/padding-strides-maxpooling-stacking-redes-convolucionales/

Example – Exploring Sliding windows

Para comprender mejor el efecto de los parámetros de **sliding windows** (ventana deslizante), puede ser útil observar una extracción de características en una imagen de baja resolución para que podamos ver los píxeles individuales. Echemos un vistazo a un círculo simple.

Esta próxima celda creará una imagen y un núcleo para nosotros.

```
import tensorflow as tf
import matplotlib.pyplot as plt

plt.rc('figure', autolayout=True)
plt.rc('axes', labelweight='bold', labelsize='large',
       titleweight='bold', titlesize=18, titlepad=10)
plt.rc('image', cmap='magma')

image = circle([64, 64], val=1.0, r_shrink=3)
image = tf.reshape(image, [*image.shape, 1])
# Bottom sobel
kernel = tf.constant(
    [[-1, -2, -1],
     [0, 0, 0],
     [1, 2, 1]],
)

show_kernel(kernel)
```

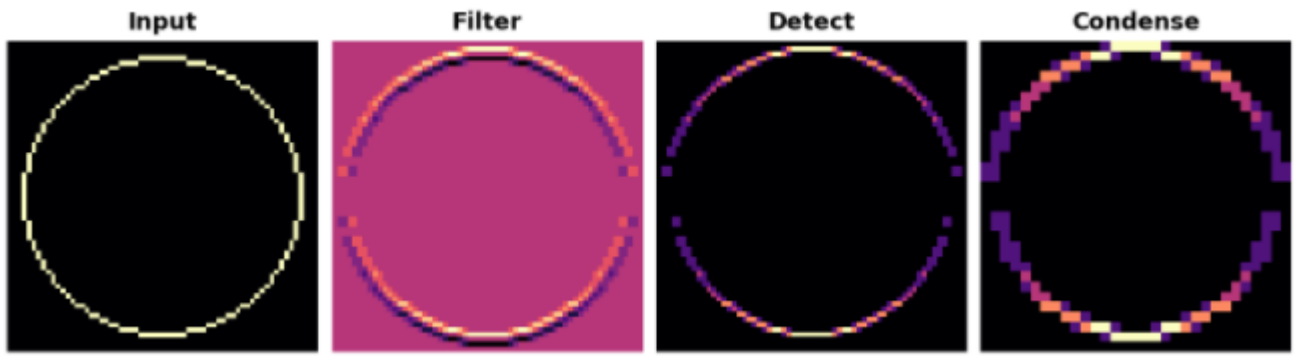


Semilleros by DSRP

Example – Exploring Sliding windows

La arquitectura de VGG es bastante simple. Utiliza *convolution* con *strides* de 1 y *maximum pooling* con ventanas de 2 × 2 y *strides* de 2. Hemos incluido una función en el script de la utilidad *visiontools* que nos mostrará todos los pasos.

```
show_extraction(
    image, kernel,

    # Window parameters
    conv_stride=1,
    pool_size=2,
    pool_stride=2,

    subplot_shape=(1, 4),
    figsize=(14, 6),
)
```

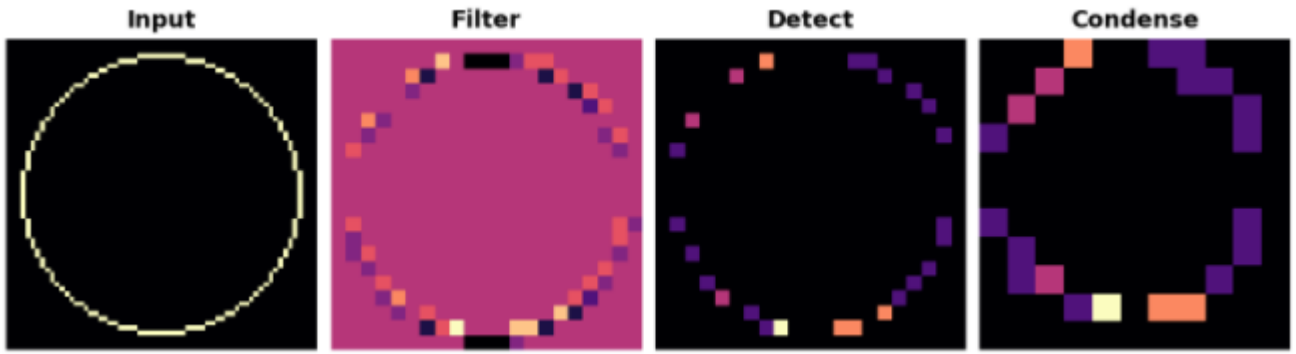


Example – Exploring Sliding windows

¡Y eso funciona bastante bien! El *kernel* fue diseñado para detectar líneas horizontales, y podemos ver que en el mapa de características resultante, las partes más horizontales de la entrada terminan con la mayor activación.

¿Qué pasaría si cambiamos los pasos de la convolución a 3?

```
show_extraction(
    image, kernel,

    # Window parameters
    conv_stride=3,
    pool_size=2,
    pool_stride=2,

    subplot_shape=(1, 4),
    figsize=(14, 6),
)
```



Esto parece reducir la calidad de la característica extraída. Nuestro círculo de entrada es bastante "finamente detallado", con solo 1 píxel de ancho. Una convolución con pasos de 3 es demasiado burda para producir un buen mapa de características a partir de ella.

A veces, un modelo utilizará una convolución con un paso más grande en su capa inicial. Por lo general, esto también se combinará con un kernel más grande. El modelo ResNet50, por ejemplo, utiliza núcleos de 7 × 7 con pasos de 2 en su primera capa. Esto parece acelerar la producción de características a gran escala sin sacrificar demasiada información de la entrada.

LABORATORIO 4

Semilleros
by DSRP