# LABORATORIO 3

Semilleros
by DSRP

# Ejercicios

Para comenzar, ejecute la celda de código a continuación

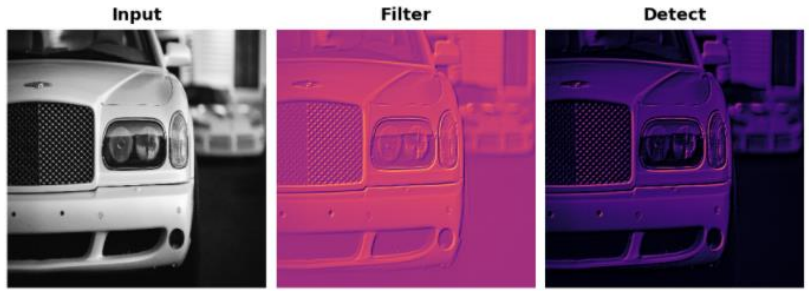
```
# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.computer_vision.ex3 import *

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib import gridspec
import learntools.computer_vision.visiontools as visiontools

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

Ejecute esta celda para volver a donde lo dejó en la lección anterior. Usaremos un kernel predefinido esta vez.

```
# Read image
image_path = '../input/computer-vision-resources/car_illus.jpg'
image = tf.io.read_file(image_path)
image = tf.io.decode_jpeg(image, channels=1)
image = tf.image.resize(image, size=[400, 400])

# Embossing kernel
kernel = tf.constant([
    [-2, -1, 0],
    [-1, 1, 1],
    [0, 1, 2],
])

# Reformat for batch compatibility.
image = tf.image.convert_image_dtype(image, dtype=tf.float32)
image = tf.expand_dims(image, axis=0)
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast(kernel, dtype=tf.float32)

image_filter = tf.nn.conv2d(
    input=image,
    filters=kernel,
    strides=1,
    padding='VALID',
)
```

→

```
image_detect = tf.nn.relu(image_filter)

# Show what we have so far
plt.figure(figsize=(12, 6))
plt.subplot(131)
plt.imshow(tf.squeeze(image), cmap='gray')
plt.axis('off')
plt.title('Input')
plt.subplot(132)
plt.imshow(tf.squeeze(image_filter))
plt.axis('off')
plt.title('Filter')
plt.subplot(133)
plt.imshow(tf.squeeze(image_detect))
plt.axis('off')
plt.title('Detect')
plt.show();
```




Semilleros by DSRP

# Ejercicios

***Paso 1: Apply Pooling to Condense***

Para el último paso de la secuencia, aplique maximum pooling utilizando una ventana de combinación de 2 × 2. Puede copiar este código para comenzar:

```
image_condense = tf.nn.pool(
    input=image_detect,
    window_shape=____,
    pooling_type=____,
    strides=(2, 2),
    padding='SAME',
)
```

```
# YOUR CODE HERE
image_condense = tf.nn.pool(
    input=image_detect,
    window_shape=(2, 2),
    pooling_type='MAX',
    strides=(2, 2),
    padding='SAME',
)
# Check your answer
q_1.check()
```

Correct

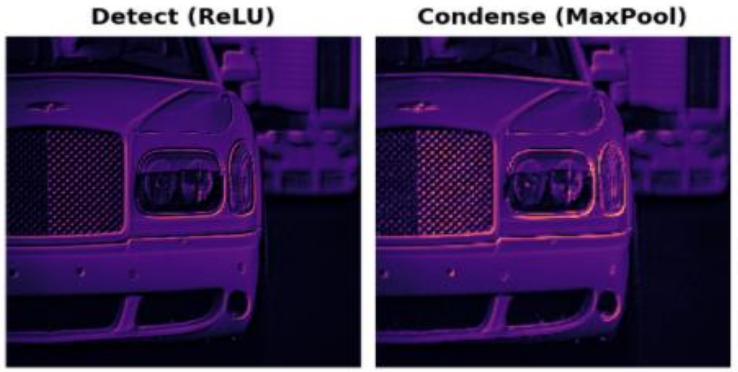
Semilleros by DSRP

# Ejercicios

***Paso 1: Apply Pooling to Condense***

Ejecute la siguiente celda para ver lo que hizo la agrupación máxima a la función.

```
plt.figure(figsize=(8, 6))
plt.subplot(121)
plt.imshow(tf.squeeze(image_detect))
plt.axis('off')
plt.title("Detect (ReLU)")
plt.subplot(122)
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.title("Condense (MaxPool)")
plt.show();
```




Aprendimos cómo las capas MaxPool2D otorgan a una red convolucional la propiedad de **translation invariance** (invariancia de traducción) en distancias pequeñas. En este ejercicio, tendrá la oportunidad de observar esto en acción. Esta siguiente celda de código aplicará aleatoriamente un pequeño cambio a un círculo y luego condensará la imagen varias veces con la máxima agrupación. Ejecute la celda una vez y tome nota de la imagen que aparece al final.

```
REPEATS = 4
SIZE = [64, 64]

# Create a randomly shifted circle
image = visiontools.circle(SIZE, r_shrink=4, val=1)
image = tf.expand_dims(image, axis=-1)
image = visiontools.random_transform(image, jitter=3, fill_method='replicate')
image = tf.squeeze(image)

plt.figure(figsize=(16, 4))
plt.subplot(1, REPEATS+1, 1)
plt.imshow(image, vmin=0, vmax=1)
plt.title("Original\nShape: {}x{}".format(image.shape[0], image.shape[1]))
plt.axis('off')
```

```
# Now condense with maximum pooling several times
for i in range(REPEATS):
    ax = plt.subplot(1, REPEATS+1, i+2)
    image = tf.reshape(image, [1, *image.shape, 1])
    image = tf.nn.pool(image, window_shape=(2,2), strides=(2, 2), padding='SAME', pooling_type='MAX')
    image = tf.squeeze(image)
    plt.imshow(image, vmin=0, vmax=1)
    plt.title("MaxPool {}\nShape: {}x{}".format(i+1, image.shape[0], image.shape[1]))
    plt.axis('off')
```


Semilleros by DSRP

# Ejercicios

***Paso 2: Explore Invariance***

## Global Average Pooling

¿Qué está haciendo esta capa? Observe que ya no tenemos la capa *flatten* (Aplanar) que generalmente viene después de la base para transformar los datos de entidades 2D en datos 1D que necesita el clasificador. Ahora la capa *GlobalAvgPool2D* está cumpliendo esta función. Pero, en lugar de "desapilar" la característica (como Flatten), simplemente reemplaza todo el mapa de características con su valor promedio. Aunque es muy destructivo, a menudo funciona bastante bien y tiene la ventaja de reducir el número de parámetros del modelo.

Veamos lo que hace GlobalAvgPool2D en algunos mapas de características generados aleatoriamente. Esto nos ayudará a comprender cómo se puede "aplanar" la pila de mapas de características producidos por la base.

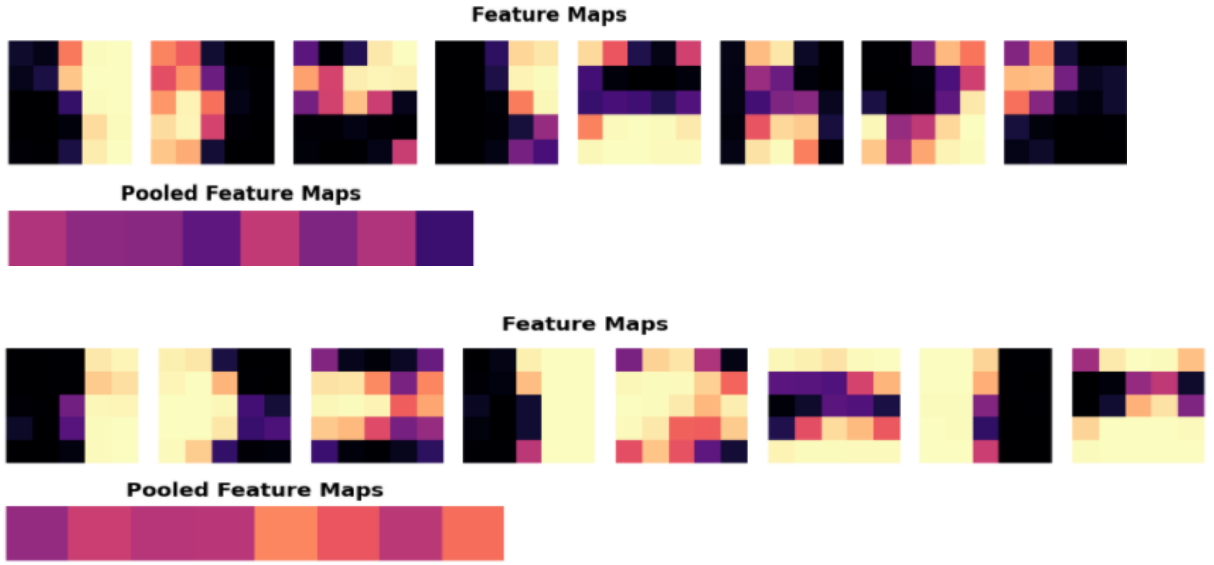
```
feature_maps = [visiontools.random_map([5, 5], scale=0.1, decay_power=4) for _ in range(8)]

gs = gridspec.GridSpec(1, 8, wspace=0.01, hspace=0.01)
plt.figure(figsize=(18, 2))
for i, feature_map in enumerate(feature_maps):
    plt.subplot(gs[i])
    plt.imshow(feature_map, vmin=0, vmax=1)
    plt.axis('off')
plt.suptitle('Feature Maps', size=18, weight='bold', y=1.1)
plt.show()

# reformat for TensorFlow
feature_maps_tf = [tf.reshape(feature_map, [1, *feature_map.shape, 1])
                   for feature_map in feature_maps]

global_avg_pool = tf.keras.layers.GlobalAvgPool2D()
pooled_maps = [global_avg_pool(feature_map) for feature_map in feature_maps_tf]
img = np.array(pooled_maps)[:,:,0].T

plt.imshow(img, vmin=0, vmax=1)
plt.axis('off')
plt.title('Pooled Feature Maps')
plt.show();
```

Semilleros by DSRP

# Ejercicios

***Paso 2: Explore Invariance***

**Global Average Pooling**




Puede ejecutar el código anterior varias veces para visualizar el cambio.

Dado que cada uno de los mapas de características de 5 × 5 se redujo a un solo valor, la combinación global redujo la cantidad de parámetros necesarios para representar estas características en un factor de 25: ¡un ahorro sustancial!

Semilleros by DSRP

# Ejercicios

***Paso 2: Explore Invariance***

## Global Average Pooling

Ahora pasaremos a comprender las funciones agrupadas. Después de haber agrupado las características en un solo valor, ¿el encabezado todavía tiene suficiente información para determinar una clase? Esta parte del ejercicio investigará esa cuestión.

Pasemos algunas imágenes de nuestro conjunto de datos de automóviles o camiones a través de VGG16 y examinemos las características que resultan después de la agrupación. Primero ejecute esta celda para definir el modelo y cargar el conjunto de datos.

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing import image_dataset_from_directory

# Load VGG16
pretrained_base = tf.keras.models.load_model(
    '../input/cv-course-models/cv-course-models/vgg16-pretrained-base',
)

model = keras.Sequential([
    pretrained_base,
    # Attach a global average pooling layer after the base
    layers.GlobalAvgPool2D(),
])

# Load dataset
ds = image_dataset_from_directory(
    '../input/car-or-truck/train',
    labels='inferred',
    label_mode='binary',
    image_size=[128, 128],
    interpolation='nearest',
    batch_size=1,
    shuffle=True,
)

ds_iter = iter(ds)
```

Semilleros by DSRP

# Ejercicios

***Paso 2: Explore Invariance***

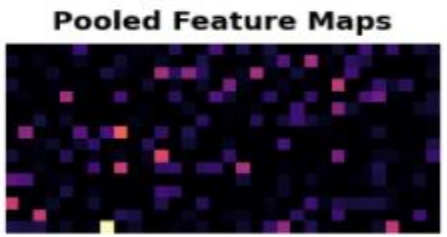
## Global Average Pooling

Observe cómo hemos adjuntado una capa *GlobalAvgPool2D* después de la base VGG16 pre-entrenada. Normalmente, VGG16 producirá 512 mapas de características para cada imagen. La capa GlobalAvgPool2D reduce cada uno de estos a un solo valor, un "píxel promedio", si lo desea.

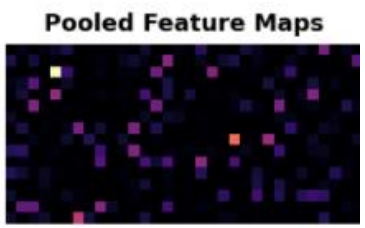
La siguiente celda ejecutará una imagen del conjunto de datos de automóvil o camión a través de VGG16 y le mostrará los 512 píxeles promedio creados por GlobalAvgPool2D.

Ejecute la celda varias veces y observe los píxeles producidos por los automóviles frente a los píxeles producidos por los camiones.

```
car = next(ds_iter)

car_tf = tf.image.resize(car[0], size=[128, 128])
car_features = model(car_tf)
car_features = tf.reshape(car_features, shape=(16, 32))
label = int(tf.squeeze(car[1]).numpy())

plt.figure(figsize=(8, 4))
plt.subplot(121)
plt.imshow(tf.squeeze(car[0]))
plt.axis('off')
plt.title(["Car", "Truck"][label])
plt.subplot(122)
plt.imshow(car_features)
plt.title('Pooled Feature Maps')
plt.axis('off')
plt.show();
```






Semilleros by DSRP

# Ejercicios

***Paso 3: Understand the pooled Features***

¿Que ves? ¿Son las funciones agrupadas para automóviles y camiones lo suficientemente diferentes como para diferenciarlos? ¿Cómo interpretaría estos valores agrupados? ¿Cómo podría ayudar esto a la clasificación?

Una vez que lo haya pensado, ejecute la siguiente celda para obtener una respuesta. (¡O vea una pista primero!)

```
# View the solution (Run this code cell to receive credit!)
q_3.check()
```

Correct:

The VGG16 base produces 512 feature maps. We can think of each feature map as representing some high-level visual feature in the original image -- maybe a wheel or window. Pooling a map gives us a single number, which we could think of as a *score* for that feature: large if the feature is present, small if it is absent. Cars tend to score high with one set of features, and Trucks score high with another. Now, instead of trying to map raw features to classes, the head only has to work with these scores that `GlobalAvgPool2D` produced, a much easier problem for it to solve.

La base VGG16 produce 512 mapas de características. Podemos pensar que cada mapa de características representa una característica visual de alto nivel en la imagen original, tal vez una rueda o una ventana. La agrupación de un mapa nos da un número único, que podríamos considerar como una puntuación para esa característica: grande si la característica está presente, pequeña si está ausente. Los automóviles tienden a obtener un puntaje alto con un conjunto de características y los camiones obtienen un puntaje alto con otro. Ahora, en lugar de intentar mapear características en bruto a clases, el jefe solo tiene que trabajar con estas puntuaciones que generó GlobalAvgPool2D, un problema mucho más fácil de resolver.

La agrupación de promedios globales se utiliza a menudo en convnets modernos. Una gran ventaja es que reduce en gran medida la cantidad de parámetros en un modelo, al mismo tiempo que le indica si alguna característica estaba presente en una imagen o no, que para la clasificación suele ser lo único que importa. Si está creando un clasificador convolucional, ¡vale la pena probarlo!

Semilleros by DSRP

# LABORATORIO 4

Semilleros
by DSRP

# Ejercicios

En estos ejercicios, explorará las operaciones que utilizan un par de arquitecturas convnet populares para la extracción de características, aprenderá cómo las convnets pueden capturar características visuales a gran escala a través de capas de apilamiento y, finalmente, verá cómo se puede utilizar la convolución en datos unidimensionales. en este caso, una serie de tiempo.

Ejecute la celda a continuación para configurar todo.

```
# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.computer_vision.ex4 import *

import tensorflow as tf
import matplotlib.pyplot as plt
import learntools.computer_vision.visiontools as visiontools


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

Semilleros by DSRP

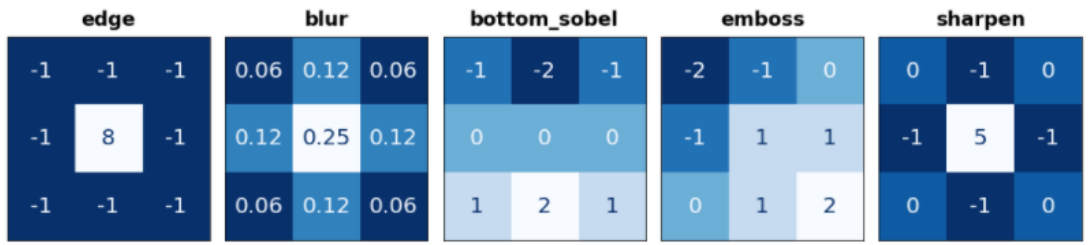
# Ejercicios

**(Optional) Experimenting with Feature extraction**

Este ejercicio está destinado a brindarle la oportunidad de explorar los cálculos de la ventana deslizante y cómo sus parámetros afectan la extracción de características. No hay respuestas correctas o incorrectas, ¡es solo una oportunidad para experimentar!

Le proporcionamos algunas imágenes y núcleos que puede utilizar. Ejecute esta celda para verlos.

```
from learntools.computer_vision.visiontools import edge, blur, bottom_sobel, emboss, sharpen, circle

image_dir = '../input/computer-vision-resources/'
circle_64 = tf.expand_dims(circle([64, 64], val=1.0, r_shrink=4), axis=-1)
kaggle_k = visiontools.read_image(image_dir + str('k.jpg'), channels=1)
car = visiontools.read_image(image_dir + str('car_illus.jpg'), channels=1)
car = tf.image.resize(car, size=[200, 200])
images = [(circle_64, "circle_64"), (kaggle_k, "kaggle_k"), (car, "car")]

plt.figure(figsize=(14, 4))
for i, (img, title) in enumerate(images):
    plt.subplot(1, len(images), i+1)
    plt.imshow(tf.squeeze(img))
    plt.axis('off')
    plt.title(title)
plt.show();

kernels = [(edge, "edge"), (blur, "blur"), (bottom_sobel, "bottom_sobel"),
           (emboss, "emboss"), (sharpen, "sharpen")]
plt.figure(figsize=(14, 4))
for i, (krn, title) in enumerate(kernels):
    plt.subplot(1, len(kernels), i+1)
    visiontools.show_kernel(krn, digits=2, text_size=20)
    plt.title(title)
plt.show()
```

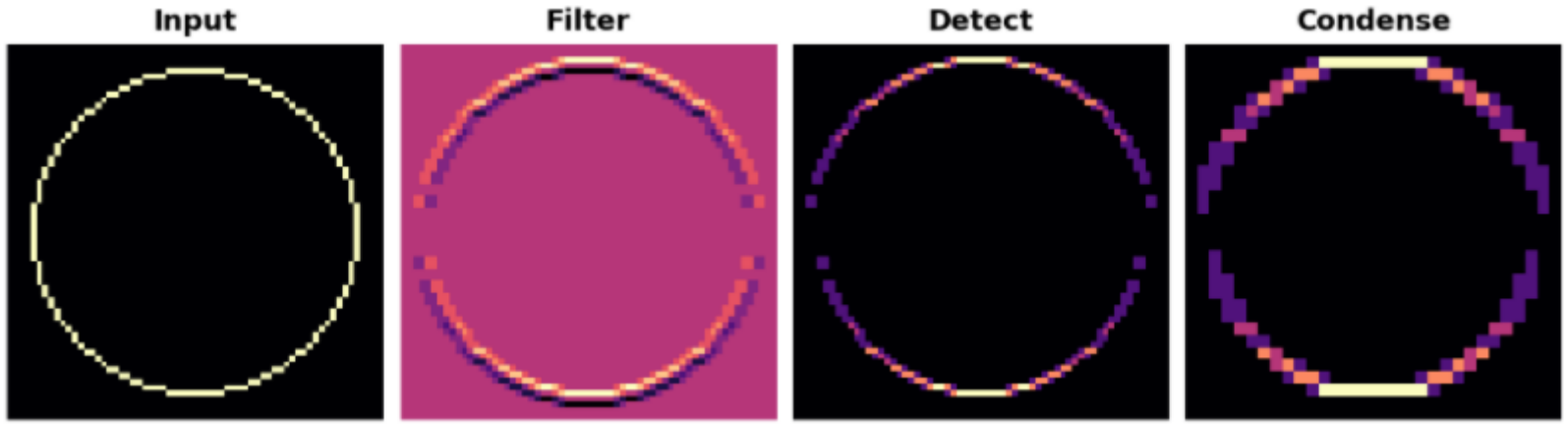

Semilleros by DSRP

# Ejercicios

**(Optional) Experimenting with Feature extraction**

Para elegir uno con el que experimentar, simplemente ingrese el nombre en el lugar apropiado a continuación. Luego, configure los parámetros para el cálculo de la ventana. ¡Pruebe algunas combinaciones diferentes y vea lo que hacen!

```
# YOUR CODE HERE: choose an image
image = circle_64

# YOUR CODE HERE: choose a kernel
kernel = bottom_sobel

visiontools.show_extraction(
    image, kernel,

    # YOUR CODE HERE: set parameters
    conv_stride=1,
    conv_padding='valid',
    pool_size=2,
    pool_stride=2,
    pool_padding='same',

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



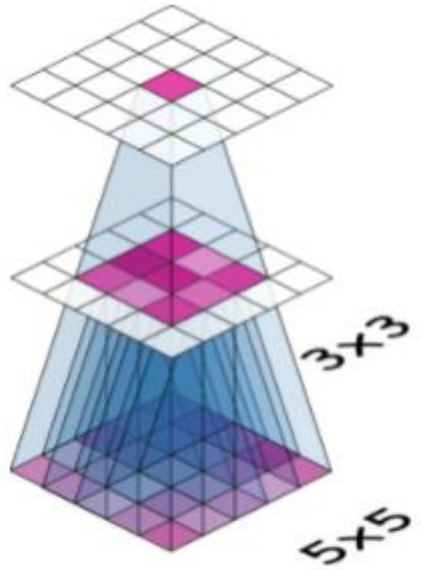

Semilleros by DSRP

# Ejercicios

**The receptive field**

Rastree todas las conexiones de alguna neurona y, finalmente, llegará a la imagen de entrada. Todos los píxeles de entrada a los que está conectada una neurona son el *receptive field* (campo receptivo) de esa neurona. El campo receptivo solo le dice de qué partes de la imagen de entrada recibe información una neurona.

Como hemos visto, si su primera capa es una convolución con kernels de 3 × 3, entonces cada neurona en esa capa recibe la entrada de un parche de píxeles de 3 × 3 (excepto tal vez en el borde).¿Qué sucede si agrega otra capa convolucional con granos de 3 × 3? Considere la siguiente ilustración:




Ahora rastrea las conexiones de la neurona en la parte superior y puedes ver que está conectada a un parche de píxeles de 5 × 5 en la entrada (la capa inferior): cada neurona en el parche de 3 × 3 en la capa intermedia está conectada a un Parche de entrada de 3 × 3, pero se superponen en un parche de 5 × 5. Entonces esa neurona en la parte superior tiene un campo receptivo de 5 ×5

Semilleros by DSRP

# Ejercicios

***Paso 1: Growing the Receptive Field***

Ahora, si agrega una tercera capa convolucional con un (3, 3) kernel, ¿qué campo receptivo tendrían sus neuronas?

```
# View the solution (Run this code cell to receive credit!)
q_1.check()
```

Correct:

The third layer would have a $7 \times 7$ receptive field.

Entonces, ¿por qué apilar capas como esta? Tres (3, 3) núcleos tienen 27 parámetros, mientras que un núcleo (7, 7) tiene 49, aunque ambos crean el mismo campo receptivo. Este truco de capas de apilamiento es una de las formas en que los convnets pueden crear grandes campos receptivos sin aumentar demasiado el número de parámetros. ¡Verá cómo hacerlo usted mismo en la próxima lección!

Semilleros by DSRP

# Ejercicios

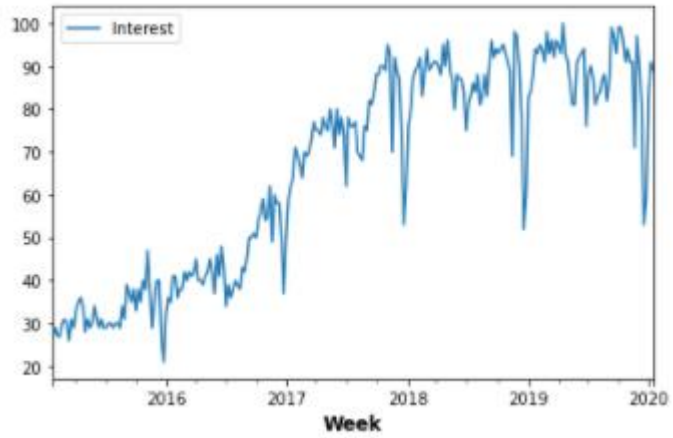
**(Optional) One-Dimensional Convolution**

Las redes convolucionales resultan ser útiles no solo en imágenes (bidimensionales), sino también en cosas como series de tiempo (unidimensionales) y video (tridimensionales).

Hemos visto cómo las redes convolucionales pueden aprender a extraer características de imágenes (bidimensionales). Resulta que los convnets también pueden aprender a extraer características de cosas como series de tiempo (unidimensionales) y videos (tridimensionales).En este ejercicio (opcional), veremos cómo se ve la convolución en una serie de tiempo.

La serie de tiempo que usaremos es de Google Trends. Mide la popularidad del término de búsqueda "aprendizaje automático" durante semanas desde el 25 de enero de 2015 hasta el 15 de enero de 2020.

```
import pandas as pd

# Load the time series as a Pandas dataframe
machinelearning = pd.read_csv(
    '../input/computer-vision-resources/machinelearning.csv',
    parse_dates=['Week'],
    index_col='Week',
)

machinelearning.plot();
```




Semilleros by DSRP

# Ejercicios

**(Optional) One-Dimensional Convolution**

¿Qué pasa con los granos? Las imágenes son bidimensionales, por lo que nuestros núcleos eran matrices 2D. Una serie de tiempo es unidimensional, entonces, ¿cuál debería ser el kernel? ¡Una matriz 1D! A continuación, se muestran algunos núcleos que a veces se utilizan en datos de series de tiempo:

```
detrend = tf.constant([-1, 1], dtype=tf.float32)

average = tf.constant([0.2, 0.2, 0.2, 0.2, 0.2], dtype=tf.float32)

spencer = tf.constant([-3, -6, -5, 3, 21, 46, 67, 74, 67, 46, 32, 3, -5, -6, -3], dtype=tf.float32) / 320
```

La convolución en una secuencia funciona como la convolución en una imagen. La diferencia es solo que una ventana deslizante en una secuencia solo tiene una dirección para viajar, de izquierda a derecha, en lugar de las dos direcciones en una imagen. Y al igual que antes, las características seleccionadas dependen del patrón de números en el kernel.

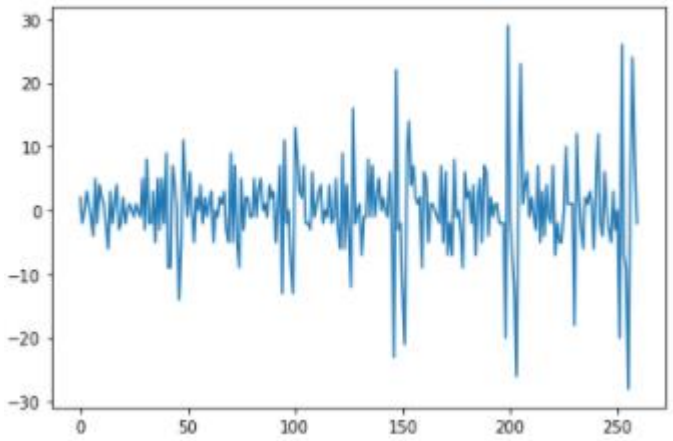
Semilleros by DSRP

# Ejercicios

**(Optional) One-Dimensional Convolution**

¿Puedes adivinar qué tipo de características extraen estos núcleos?

```
# UNCOMMENT ONE
kernel = detrend
# kernel = average
# kernel = spencer

# Reformat for TensorFlow
ts_data = machinelearning.to_numpy()
ts_data = tf.expand_dims(ts_data, axis=0)
ts_data = tf.cast(ts_data, dtype=tf.float32)
kern = tf.reshape(kernel, shape=(*kernel.shape, 1, 1))

ts_filter = tf.nn.conv1d(
    input=ts_data,
    filters=kern,
    stride=1,
    padding='VALID',
)

# Format as Pandas Series
machinelearning_filtered = pd.Series(tf.squeeze(ts_filter).numpy())

machinelearning_filtered.plot();
```




De hecho, el kernel de *detrend* filtra los cambios en la serie, mientras que el *average* y el *spencer* son "suavizadores" que filtran los componentes de baja frecuencia de la serie.Si estaba interesado en predecir la popularidad futura de los términos de búsqueda, podría entrenar a un convnet en series de tiempo como esta. Intentaría saber qué características de esas series son más informativas para la predicción.

Aunque los convnets no suelen ser la mejor opción por sí mismos para este tipo de problemas, a menudo se incorporan a otros modelos por sus capacidades de extracción de características.

Semilleros by DSRP