



Semilleros
by DSRP



Ruta de Aprendizaje

COMPUTER VISION

- **6 lecciones** de Computer Vision

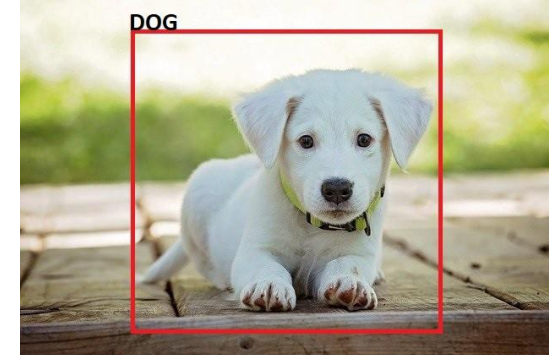
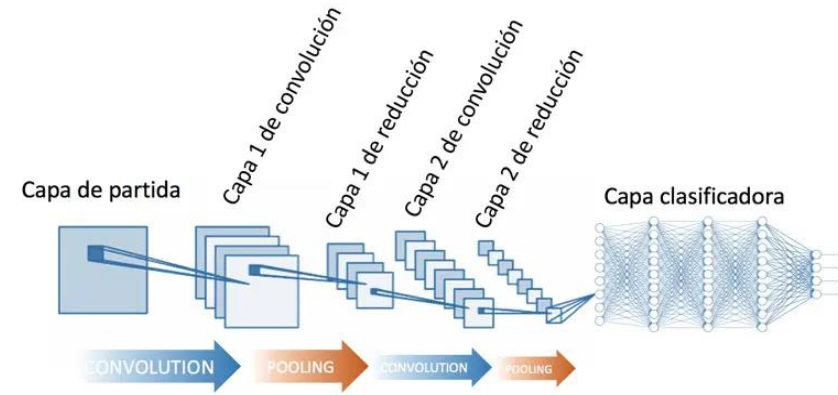


OCTOBER 2021

SUN	MON	TUE	WED	THU	FRI	SAT
					1	2
3 TERCER FEEDBACK	4	5	6	7	8	9 →
10 SEGUNDO AVANCE	11	12	13	14	15	16
17 CUARTO FEEDBACK	18	19	20	21	22	23
24 TERCER AVANCE	25	26	27	28	29	30
31 CIERRE CURSO 2						

Lesson 1

The Convolutional Classifier

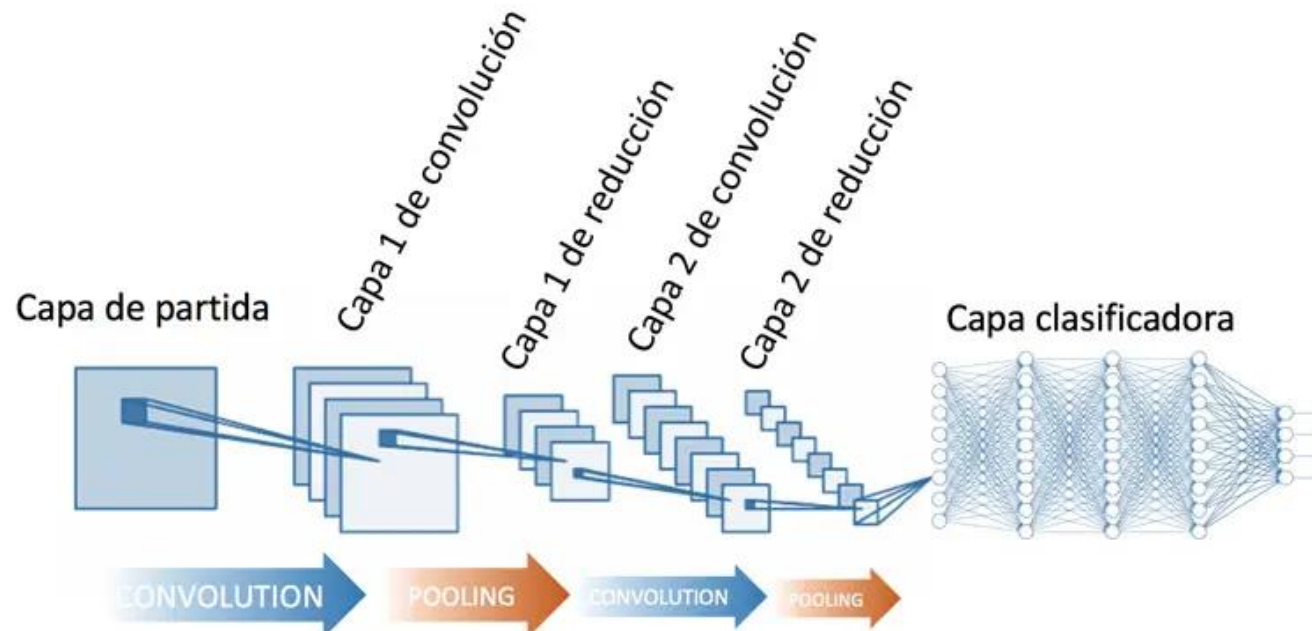


Lesson 0: Introduction

Las Redes Neuronales Convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones.

Redes Neuronales Convolucionales (CNN)

CNN aprenden a reconocer una diversidad de objetos dentro de imágenes, pero para ello necesitan “entrenarse” de previo con una cantidad importante de “muestras”, de ésta forma las neuronas de la red van a poder captar las características únicas de cada objeto y a su vez, poder generalizarlo.

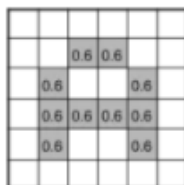


Lesson 0: Introduction

Todo comienza con un Pixel



Una imagen...



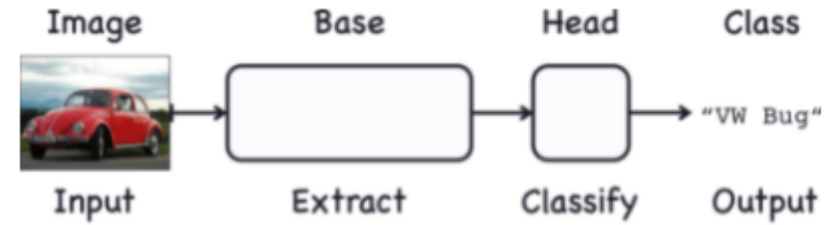
...es una matriz de pixeles.
El valor de los pixeles va de 0 a 255 pero se normaliza para la red neuronal de 0 a 1

Pixeles y neuronas

Para comenzar, la red toma como entrada los pixeles de una imagen. Si tenemos una imagen con apenas 28x28 pixeles de alto y ancho (equivale a 784 neuronas, escala de grises). Si tuviéramos una imagen a color, necesitaríamos 3 canales RGB (red, green, blue) y entonces usaríamos $28 \times 28 \times 3 = 2352$ neuronas.

Lesson 1.1: The Convolutional Classifier

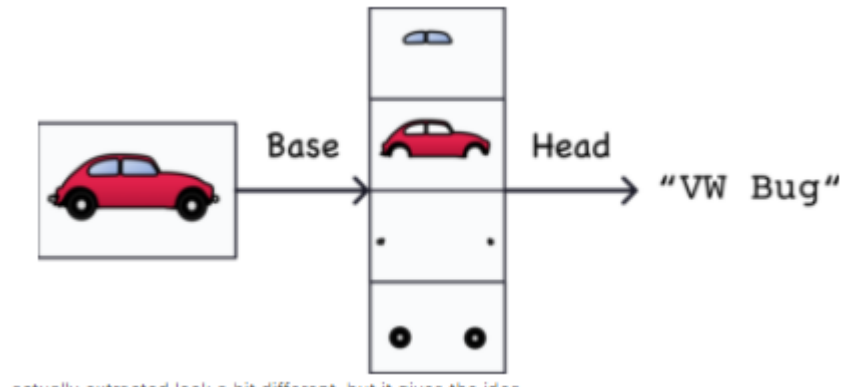
Una Red Neuronal Convolutiva para la clasificación de imágenes consta de 2 partes: Una **Convolutional base** y una **dense head**.



La **base** se utiliza para extraer las características de una imagen.

Head se usa para determinar la clase de imagen. Está formado principalmente por capas densas, pero puede incluir otras capas como *dropout*.

Todo el proceso es algo como esto:



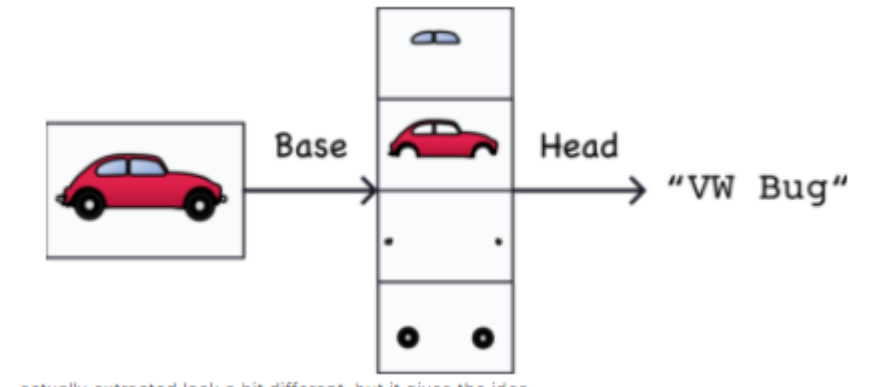
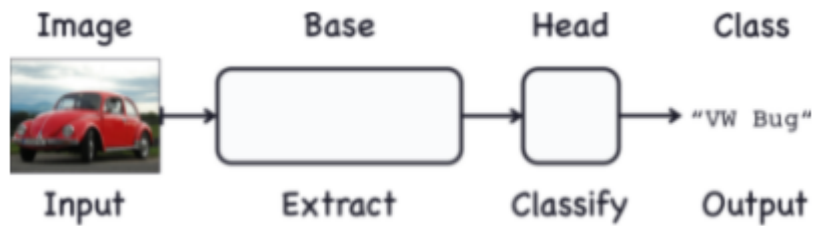
Note: ¿Qué entendemos por **visual feature**? Una característica podría ser una línea, un color, una textura, una forma, un patrón, etc.

Lesson 1.2: Training the Classifier

El objetivo de la red durante el entrenamiento es aprender dos cosas:

1. qué características extraer de una imagen (**base**)!
2. qué clase va con qué características (**head**)!

Todo el proceso es algo como esto:

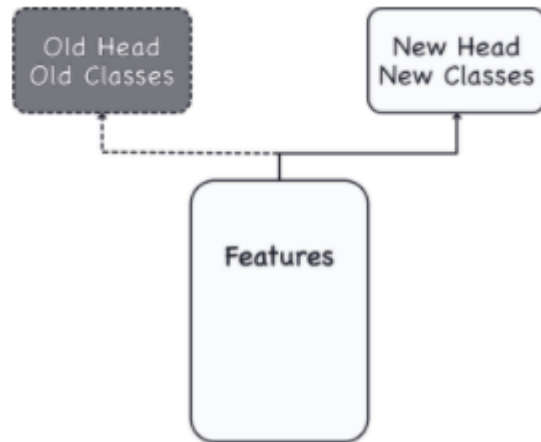


Lesson 1.2: Training the Classifier

En estos días, los **convnets** rara vez se entrenan desde cero. Más a menudo, reutilizamos la base de un modelo previamente entrenado.

A la base previamente entrenada adjuntamos un **head** no entrenada. En otras palabras, reutilizamos la parte de una red que ya ha aprendido a hacer.

1. Extraer entidades y adjuntarle algunas capas nuevas para aprender
2. Clasificar.



Debido a que el encabezado generalmente consta de solo unas pocas capas densas, se pueden crear clasificadores muy precisos a partir de relativamente pocos datos.

Reutilizar un modelo previamente entrenado es una técnica conocida como **transfer learning**. Es tan eficaz que casi todos los clasificadores de imágenes de estos días lo utilizarán.

Lesson 1.3: Example – Train a Convnet/CNN Classifier

Nuestro conjunto de datos consta de aproximadamente 10,000 imágenes de varios automóviles, alrededor de la mitad de los automóviles y la mitad de los camiones. Vamos a resolver el siguiente problema: ¿es esta una imagen de un automóvil o de un camión?

Paso 1: Load Data

Primero se importó algunas librerías y se configuró el data pipeline.

Tenemos una división de entrenamiento llamada **ds_train** y una división de validación llamada **ds_valid**.

Data Explorer



80.85 MB

- ▼ train
 - ▶ Car
 - ▶ Truck
- ▶ valid



Summary

- ▶ 10.2k files

< train (2 directories)

 Car 2961 files	 Truck 2156 files
--	--

< valid (2 directories)

 Car 2922 files	 Truck 2129 files
---	---

Lesson 1.3: Example – Train a Convnet/CNN Classifier

Nuestro conjunto de datos consta de aproximadamente 10,000 imágenes de varios automóviles, alrededor de la mitad de los automóviles y la mitad de los camiones. Vamos a resolver el siguiente problema: ¿es esta una imagen de un automóvil o de un camión?

Paso 2: Define Pre-trained Base

El conjunto de datos más utilizado para el entrenamiento previo es ImageNet, un gran conjunto de datos de muchos tipos de imágenes naturales.

Keras incluye una variedad de modelos previamente entrenados en ImageNet en su módulo de aplicaciones. El modelo preentrenado que usaremos se llama VGG16.

```
pretrained_base = tf.keras.models.load_model(
    '../input/cv-course-models/cv-course-models/vgg16-pretrained-base',
)
pretrained_base.trainable = False
```

Lesson 1.3: Example – Train a Convnet/CNN Classifier

Nuestro conjunto de datos consta de aproximadamente 10,000 imágenes de varios automóviles, alrededor de la mitad de los automóviles y la mitad de los camiones. Vamos a resolver el siguiente problema: ¿es esta una imagen de un automóvil o de un camión?

Paso 3: Attach Head

A continuación, adjuntamos clasificador **head**. Para este ejemplo, usaremos una capa de unidades ocultas (la primera capa Densa) seguida de una capa para transformar las salidas en una puntuación de probabilidad para la clase 1, **Camión**.

La capa Flatten transforma las salidas bidimensionales de la base en las entradas unidimensionales que necesita el **head**.

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

model = keras.Sequential([
    pretrained_base,
    layers.Flatten(),
    layers.Dense(6, activation='relu'),
    layers.Dense(1, activation='sigmoid'),
])
```

Flatten: Esta capa convierte la imagen de tres dimensiones a una sola.

Lesson 1.3: Example – Train a Convnet/CNN Classifier

Nuestro conjunto de datos consta de aproximadamente 10,000 imágenes de varios automóviles, alrededor de la mitad de los automóviles y la mitad de los camiones. Vamos a resolver el siguiente problema: ¿es esta una imagen de un automóvil o de un camión?

Paso 4: Train

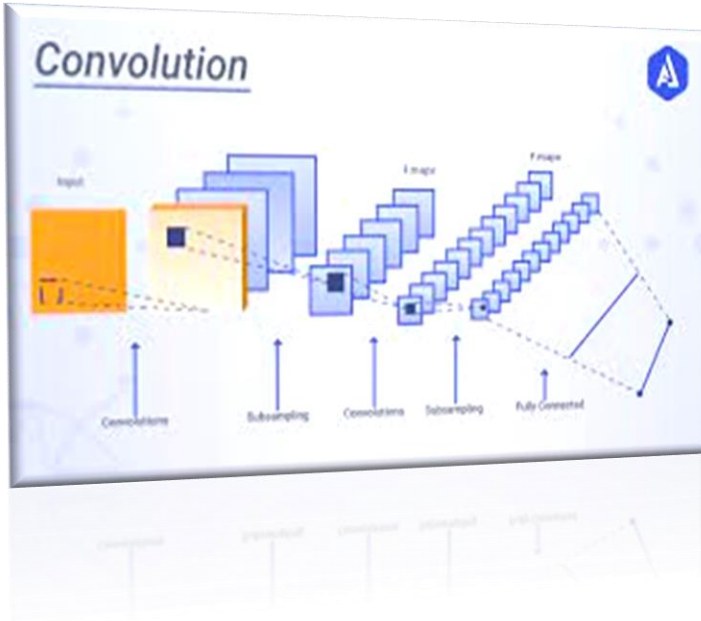
Finalmente, entrenemos el modelo. Dado que este es un problema de dos clases, usaremos las versiones binarias de **cross-entropy** (entropía cruzada) y **accuracy**. El optimizador **adam** generalmente funciona bien, por lo que también lo elegiremos.

```
model.compile(  
    optimizer='adam',  
    loss='binary_crossentropy',  
    metrics=['binary_accuracy'],  
)  
  
history = model.fit(  
    ds_train,  
    validation_data=ds_valid,  
    epochs=30,  
    verbose=0,  
)
```

Note: Como recomendación al entrenar una red neuronal, siempre es una buena idea examinar las gráficas de pérdidas y métricas.

LABORATORIO 1

Lesson #2

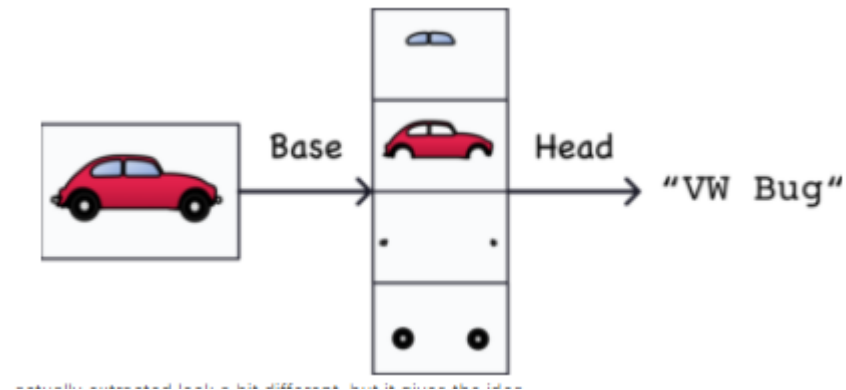


Convolution and ReLU

Lesson 2.0: Introduction

Review: En la lesson 1, vimos que un clasificador convolucional tiene 2 partes: una **base** convolucional y **head** de capas densas. Recordemos que el trabajo de la **base** es extraer *features* visuales de una imagen, que luego **head** usaría para clasificar la imagen.

Todo el proceso es algo como esto:



En estas lecciones, aprenderemos sobre los 2 tipos más importantes de capas que normalmente se encuentra en la base de un clasificador de imágenes convolucional. Estas son la **capa convolucional** de **activación de ReLU** y la capa de **maximum pooling** (agrupación máxima).

Lesson 2.0: Introduction

Capa de convolución:

Esta capa procesará la salida de neuronas que están conectadas en regiones locales (píxeles cercanos) de entrada, calculando el producto escalar entre sus pesos y una pequeña región a la que están conectados en el volumen de entrada.

Pooling:

Es una operación que permite analizar el contenido de una imagen por bloques o regiones para extraer la información más representativa de las mismas.

Maximum 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.

Función de activación "ReLU – Rectified Lineal Unit"

La función ReLU transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran.



Características:

- Activación Sparse – solo se activa si son positivos
- No está acotada
- Se pueden morir demasiadas neuronas
- Se comportan bien con imágenes
- Buen desempeño en redes convolucionales

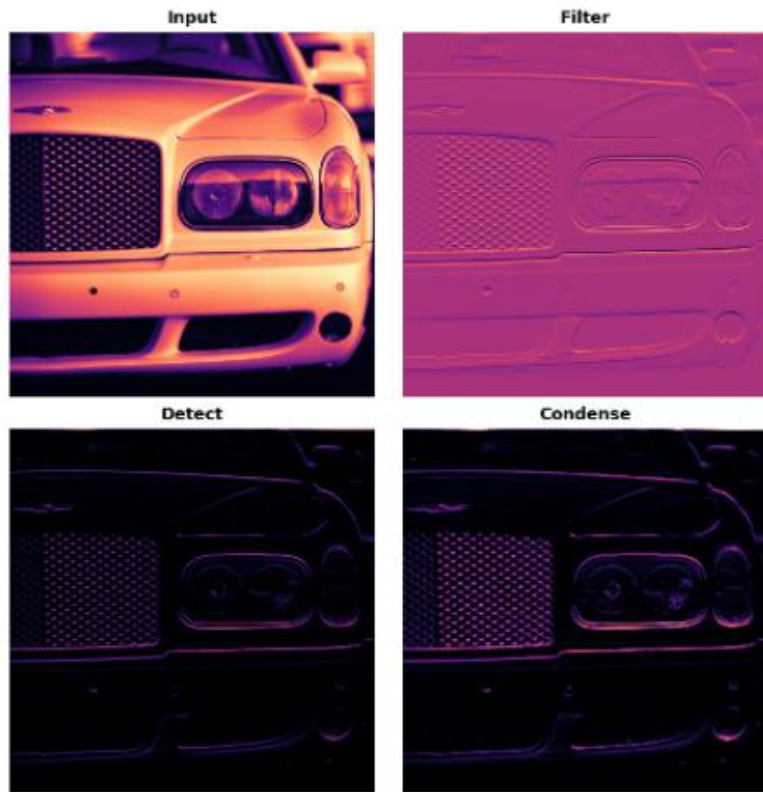
Capa ReLU:

Aplicará la función de activación en los elementos de la matriz

Lesson 2.1: Feature extraction

La extracción de características realizada por la **base** consta de tres operaciones básicas:

1. **Filter** una imagen para una característica en particular (**convolution**)
2. **Detect** que característica está dentro de la imagen filtrada (ReLU)
3. **Condense** la imagen para mejorar las características (**máximo pooling**)



The three steps of feature extraction.

La siguiente figura ilustra este proceso.

Puedes ver cómo estas tres operaciones consiguen aislar alguna característica particular de la imagen original (en este caso, líneas horizontales).

Lesson 2.2: Filter with Convolution

Una capa convolucional realiza el paso de filtrado. Puede definir una capa convolucional en un modelo de Keras algo como esto:

```
In [2]: from tensorflow import keras
        from tensorflow.keras import layers

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

filters= Puede considerar que cada filtro es responsable de extraer algún tipo de característica de la imagen en bruto. En la práctica, están en número de 64,128,256, 512 etc

Kernel_size = Es el tamaño de estos filtros de convolución. 3x3

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

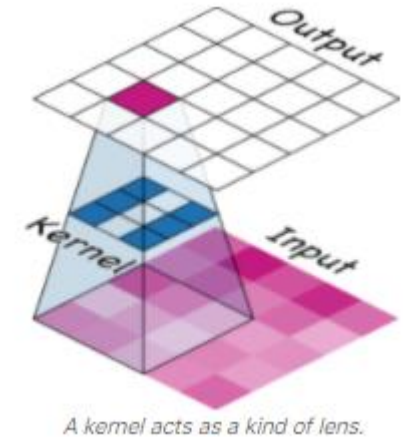
kernel

Lesson 2.3: Filter with Convolution - weights

los **weights** que aprende un convnet durante el entrenamiento están contenidos principalmente en sus capas convolucionales. Estos pesos los llamamos **kernels**. Podemos representarlos como pequeños arreglos:

-1	2	-1
-1	2	-1
-1	2	-1

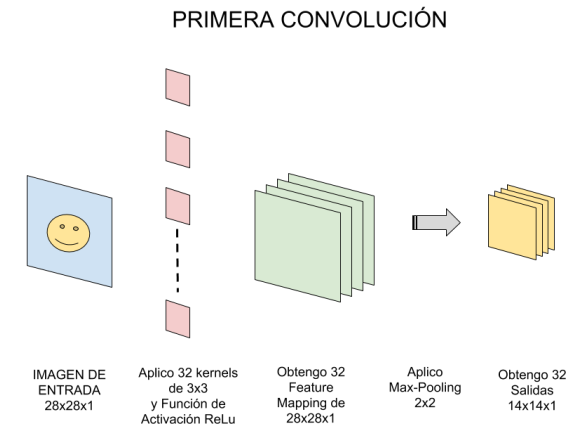
Un **kernel** funciona escaneando una imagen y produciendo una suma ponderada de valores de píxeles. De esta manera, un kernel actuará como una especie de lente polarizada, enfatizando o restando importancia a ciertos patrones de información.



Los **kernels** definen cómo se conecta una capa convolucional a la capa que sigue.

Al establecer las dimensiones de los núcleos con `kernel_size`, le está diciendo al convnet cómo formar estas conexiones. La mayoría de las veces, un kernel tendrá dimensiones impares, como `kernel_size = (3, 3)` o `(5, 5)`, de modo que un solo píxel se encuentra en el centro.

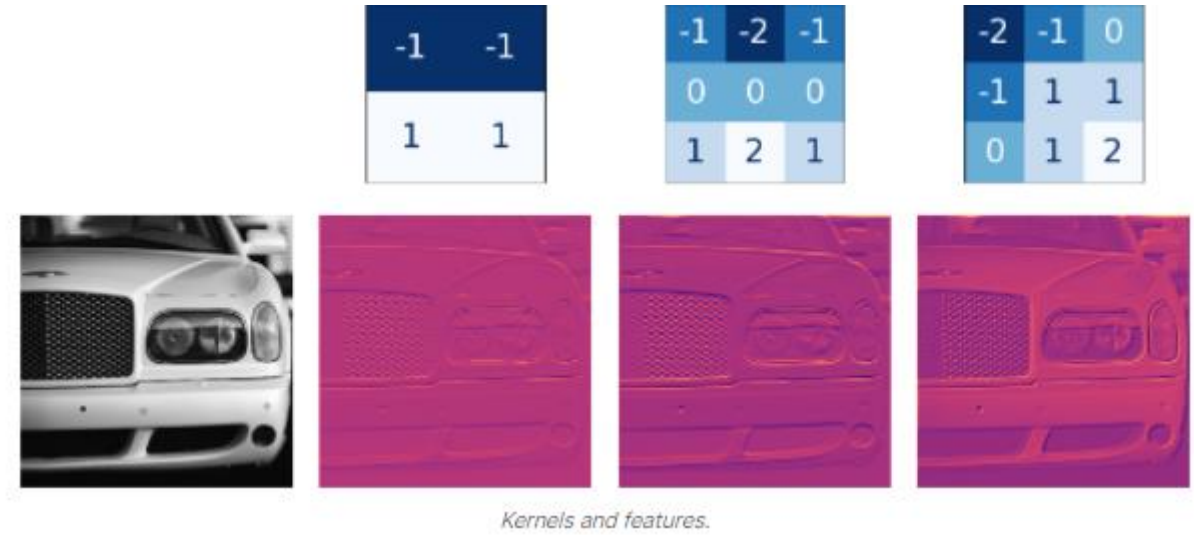
Los **kernels** en una capa convolucional determinan qué tipo de características crea.



Lesson 2.4: Filter with Convolution - activations

Las activaciones en la red las llamamos **feature maps** (mapas de características).

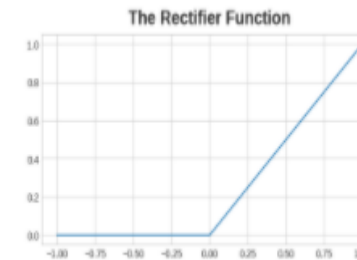
Son los que resultan cuando aplicamos un filtro a una imagen; contienen las características visuales que extrae el kernel. Aquí se muestran algunos kernels con mapas de características que produjeron.



Note: Con el parámetro de **filters** (filtros), le dice a la capa convolucional cuántos mapas de características desea que cree como salida.

Lesson 2.5: Detect with ReLU

Después del filtrado, los **feature maps** (mapas de características) pasan por la función de activación.



The graph of the rectifier function looks like a line with the negative part "rectified" to 0.

Una neurona con un rectificador adjunto se llama **rectified linear unit** (unidad lineal rectificada). Por esa razón, también podríamos llamar a la función rectificadora la **activación ReLU**.

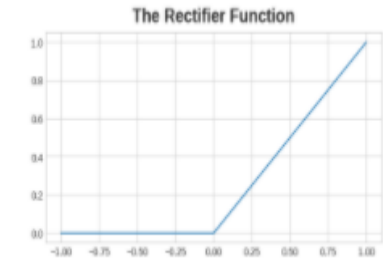
La activación de ReLU se puede definir en su propia capa de activación, pero la mayoría de las veces solo la incluirá como la función de activación de Conv2D.

```
model = keras.Sequential([
    layers.Conv2D(filters=64, kernel_size=3, activation='relu')
    # More layers follow
])
```

Lesson 2.5: Detect with ReLU

Podría pensar en la función de activación como puntuar valores de píxeles de acuerdo con alguna medida de importancia. La activación de ReLU dice que los valores negativos no son importantes y, por lo tanto, los establece en 0. ("Todo lo que no es importante es igualmente insignificante").

Aquí ReLU aplicó los **feature maps** (mapas de características) anteriores. Observe cómo logra aislar las características.



Como otras funciones de activación, la función ReLU no es lineal.

La no linealidad asegura que las características se combinarán de formas interesantes a medida que se adentren más en la red. "Se repasará en lesson 5"

LABORATORIO 2