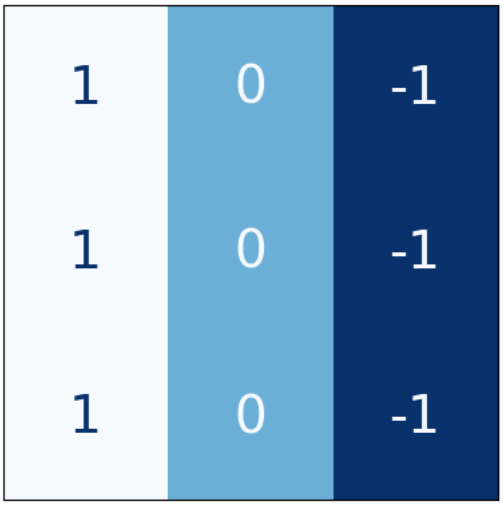
Ejercicios sobre convolución

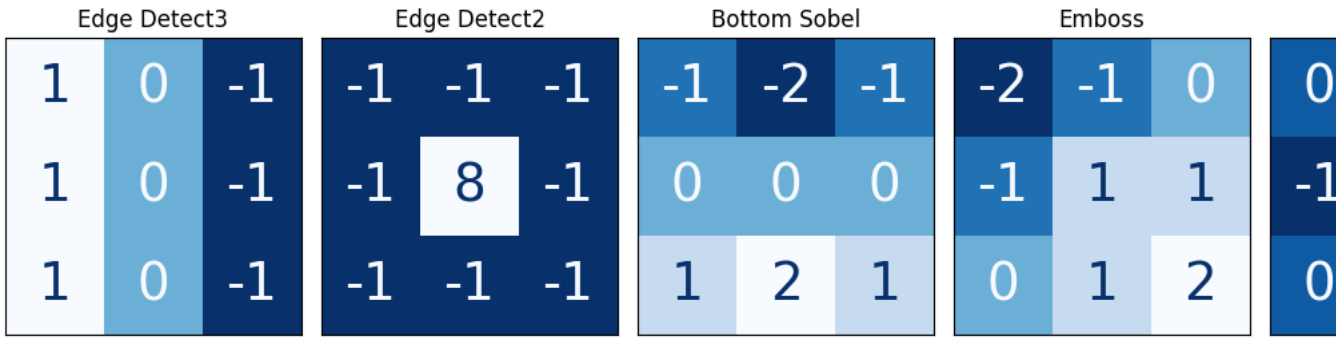
```
#imports
import numpy as np
import matplotlib.pyplot as plt
from itertools import product
import tensorflow as tf

# función útil
def mostrar_kernel(kernel, label=True, digits=None, text_size=28):
    # Format kernel
    kernel = np.array(kernel)
    if digits is not None:
        kernel = kernel.round(digits)

    # Plot kernel
    cmap = plt.get_cmap('Blues_r')
    plt.imshow(kernel, cmap=cmap)
    rows, cols = kernel.shape
    thresh = (kernel.max()+kernel.min())/2
    # Optionally, add value labels
    if label:
        for i, j in product(range(rows), range(cols)):
            val = kernel[i, j]
            color = cmap(0) if val > thresh else cmap(255)
            plt.text(j, i, val,
                     color=color, size=text_size,
                     horizontalalignment='center', verticalalignment='center')
    plt.xticks([])
    plt.yticks([])


kernel3 = tf.constant(
    [[1, 0, -1],
     [1, 0, -1],
     [1, 0, -1]],
)

mostrar_kernel(kernel3)
# kernel5 = tf.constant(
#     [[1, 1, 0, -1,-1],
#      [1, 1, 0, -1,-1],
#      [1, 1, 0, -1,-1],
#      [1, 1, 0, -1,-1],
#      [1, 1, 0, -1,-1]],
# )
# mostrar_kernel(kernel5)
```



Sección 1: Creación de kernels o filtros

```
# Edge detection
edge0 = tf.constant(
    [[1, 0, -1],
     [1, 0, -1],
     [1, 0, -1]],
```

```
)

edge = tf.constant(
    [[-1, -1, -1],
     [-1, 8, -1],
     [-1, -1, -1]],
)

# Blur
blur = tf.constant(
    [[0.0625, 0.125, 0.0625],
     [0.125, 0.25, 0.125],
     [0.0625, 0.125, 0.0625]],
)

# Bottom sobel
bottom_sobel = tf.constant(
    [[-1, -2, -1],
     [0, 0, 0],
     [1, 2, 1]],
)

# Emboss South-East
emboss = tf.constant(
    [[-2, -1, 0],
     [-1, 1, 1],
     [0, 1, 2]],
)

# Sharpen
sharpen = tf.constant(
    [[0, -1, 0],
     [-1, 5, -1],
     [0, -1, 0]],
)

kernels = [edge0,edge, bottom_sobel, emboss, sharpen]
names = ["Edge Detect3","Edge Detect2", "Bottom Sobel", "Emboss", "Sharpen"]

plt.figure(figsize=(12, 12))
for i, (kernel, name) in enumerate(zip(kernels, names)):
    plt.subplot(1, 5, i+1)
    mostrar_kernel(kernel)
    plt.title(name)
plt.tight_layout()
```



Sección 2: Carga de imagen de prueba

```
#cargando imagen
image_path = '/content/car_feature.jpg'
imagen_prueba = tf.io.read_file(image_path)
imagen_prueba = tf.io.decode_jpeg(imagen_prueba)
plt.figure(figsize=(6, 6))
plt.imshow(tf.squeeze(imagen_prueba), cmap='gray')
plt.axis('off')
plt.show();
```

Sección 3: Aplicación de convolución

```
kernel_ejemplo = tf.constant([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1],
])
image = tf.image.convert_image_dtype(imagen_prueba, dtype=tf.float32)
image = tf.expand_dims(image, axis=0)
kernel = tf.reshape(kernel_ejemplo, [*kernel_ejemplo.shape, 1, 1])
kernel = tf.cast(kernel, dtype=tf.float32)
image_filter = tf.nn.conv2d(
      input=image,
      filters=kernel,
      strides=1,
      padding='VALID',
  )
imagen_resultado = tf.squeeze(image_filter)
plt.figure(figsize=(6, 6))
plt.imshow(imagen_resultado)
plt.axis('off')
plt.show();
```

Pregunta 1: Para que es usado el método expand_dims en las lineas de código previas, muestre las dimensiones de la imagen antes y después de ejecutar la función expand_dims.

```
# image = tf.expand_dims(image, axis=0):
#
# Utilizando la función expand_dims de TensorFlow, se agrega una dimensión adicional al tensor de imagen en el eje 0.
# Esto es necesario para que la función 'conv2d' de TensorFlow pueda aplicar el filtro de convolución correctamente.
```

Pregunta 2: Para que es usado el método reshape en las lineas de código previas, muestre las dimensiones del kernel antes y después de ejecutar la función reshape.

```
# kernel = tf.reshape(kernel_ejemplo, [*kernel_ejemplo.shape, 1, 1]):
#
# Se utiliza la función reshape de TensorFlow para darle al kernel la forma requerida
# por la función 'conv2d'. Se agrega una dimensión adicional al kernel en los ejes 2 y 3
```

Pregunta 3: Para que es usado el método squeeze en las lineas de código previas, muestre las dimensiones de la imagen antes y después de ejecutar la función squeeze.

```
# imagen_resultado = tf.squeeze(image_filter):
#
# Utilizando la función squeeze de TensorFlow, se elimina la dimensión adicional agregada anteriormente al tensor de imagen filtrada.
```

```
print(imagen_prueba.shape)
```

```
(361, 421, 1)
```

```
def ejecutar_convolucion_en_imagen(image,kernel):
  #organizando los datos en las dimensiones correctas
  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)
  #aplicando un convolucion
  image_filter = tf.nn.conv2d(
      input=image,
      filters=kernel,
      strides=1,
      padding='VALID',
  )
  return image_filter
```

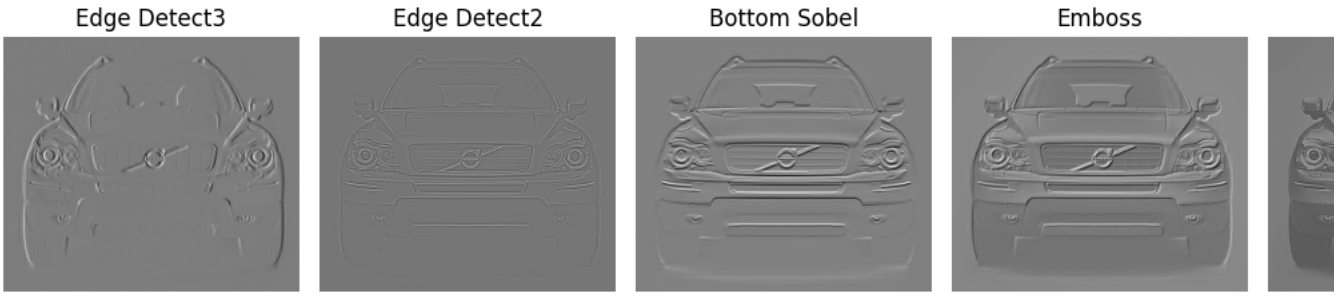
```
#ejemplo de como usar la función "ejecutar_convolucion_en_imagen"
kernel_ejemplo = tf.constant([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1],
])

imagen_resultante = ejecutar_convolucion_en_imagen(imagen_prueba,kernel_ejemplo)
plt.figure(figsize=(6, 6))
plt.imshow(tf.squeeze(imagen_resultante))
plt.axis('off')
plt.show();
```

```
print(imagen_prueba.shape)
```

```
(361, 421, 1)
```

Pregunta 4: Use la función ejecutar_convolucion_en_imagen y aplique los 5 filtros presentados en la "sección 1: Creación de kernels o filtros" a la imagen de prueba y muestre los resultados.

```
# kernels = [edge0,edge, bottom_sobel, emboss, sharpen]
# names = ["Edge Detect3","Edge Detect2", "Bottom Sobel", "Emboss", "Sharpen"]

plt.figure(figsize=(12, 12))
for i, (kernel, name) in enumerate(zip(kernels, names)):
    # Aplicar convolución a la imagen
    filtered_image = ejecutar_convolucion_en_imagen(imagen_prueba, kernel)

    # Eliminar dimensión adicional y convertir imagen resultante a numpy array
    filtered_image = tf.squeeze(filtered_image).numpy()

    # Mostrar imagen resultante
    plt.subplot(1, 5, i+1)
    plt.imshow(filtered_image, cmap='gray')
    plt.title(name)
    plt.axis('off')

plt.tight_layout()
plt.show()
```



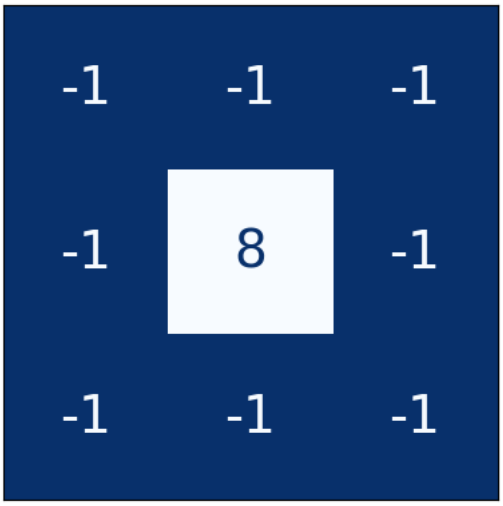
Pregunta 5: Explique las principales diferencias perceptibles entre los resultados obtenidos en la pregunta previa.

```
# DIFERENCIAS
#
# Edge Detect3: Este filtro resalta los bordes en la imagen. Los bordes verticales aparecen resaltados en blanco y negro.
#
# Edge Detect2: Similar al filtro anterior, este también resalta los bordes en la imagen. Sin embargo, los bordes se muestran con mayor g
#
# Bottom Sobel: Este filtro resalta los bordes horizontales en la imagen. Los bordes horizontales aparecen resaltados en blanco y negro.
#
# Emboss: Este filtro crea un efecto de relieve en la imagen. Genera un efecto tridimensional resaltando los bordes de los objetos y crea
#
# Sharpen: Este filtro acentúa los detalles y bordes en la imagen, produciendo un efecto de mayor nitidez.
```

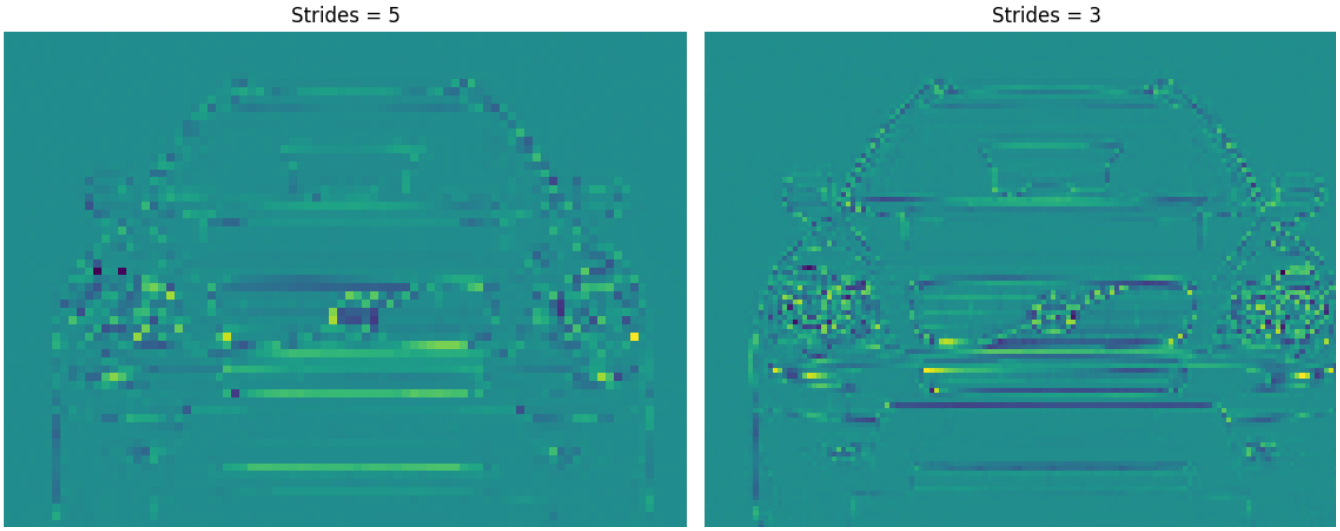
Pregunta 6: Usando la imagen de prueba y el filtro de ejemplo muestre los resultados de la función ejecutar_convolucion_en_imagen variando el parametro strides del método tf.nn.conv2d, pruebe con valores de strides: 5, 3 y 1 e explique las diferencias de los resultados.

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

```
kernel_ejemplo = tf.constant([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1],
])
mostrar_kernel(kernel_ejemplo)
```




```
strides_values = [5, 3, 1]

def ejecutar_convolucion_en_imagen_add_stride(image,kernel,_stride_):
  #organizando los datos en las dimensiones correctas
  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)
  #aplicando un convolucion
  image_filter = tf.nn.conv2d(
      input=image,
      filters=kernel,
      strides=_stride_,
      padding='VALID',
  )
  return image_filter

plt.figure(figsize=(18, 6))
for i, strides in enumerate(strides_values):
    imagen_resultante = ejecutar_convolucion_en_imagen_add_stride(imagen_prueba, kernel_ejemplo, strides)

    plt.subplot(1, 3, i+1)
    plt.imshow(tf.squeeze(imagen_resultante))
    plt.axis('off')
    plt.title(f"Strides = {strides}")

plt.tight_layout()
plt.show()
```

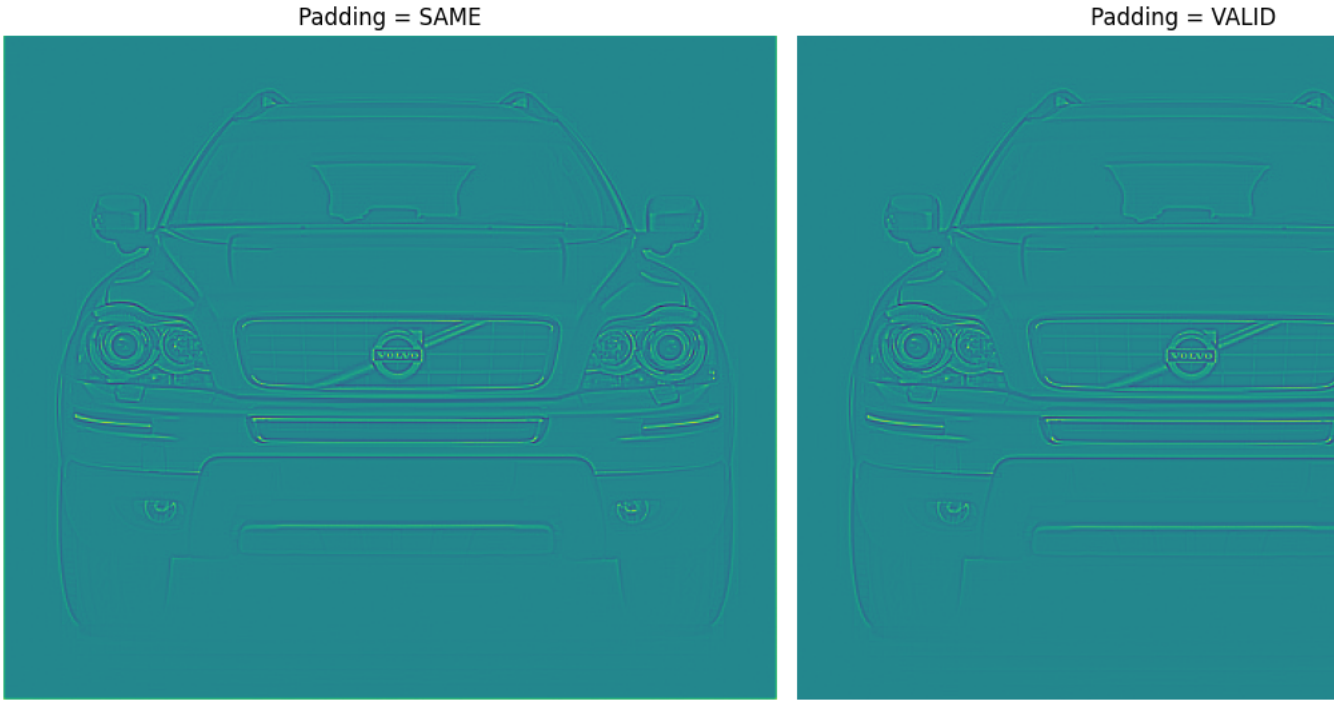



```
# Como notamos entre la diferencia entre cada imagen es que cada imagen es mas distorcionada por lo que podriamos inferir que
# Un valor de strides mayor producirá un mayor espaciado entre las regiones convolucionadas, lo que puede resultar en una pérdida de deta
# y resolución en la imagen resultante.
```

Pregunta 7: Usando la imagen de prueba y el filtro de ejemplo muestre los resultados de la función ejecutar_convolucion_en_imagen variando el parametro padding del método tf.nn.conv2d, pruebe con valores de padding: 'SAME','VALID' e explique las diferencias de los resultados.

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

kernel_ejemplo = tf.constant([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1],
])
mostrar_kernel(kernel_ejemplo)
```

```
padding_values = ['SAME','VALID']

def ejecutar_convolucion_en_imagen_add_padding(image,kernel,_padding_):
  #organizando los datos en las dimensiones correctas
  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)
  #aplicando un convolucion
  image_filter = tf.nn.conv2d(
      input=image,
      filters=kernel,
      strides=1,
      padding=_padding_,
  )
  return image_filter

plt.figure(figsize=(18, 6))
for i, padd in enumerate(padding_values):
    imagen_resultante = ejecutar_convolucion_en_imagen_add_padding(imagen_prueba, kernel_ejemplo, padd)

    plt.subplot(1, 3, i+1)
    plt.imshow(tf.squeeze(imagen_resultante))
    plt.axis('off')
    plt.title(f"Padding = {padd}")

plt.tight_layout()
plt.show()
```




```
# Padding = 'SAME': En este caso, se aplica un relleno para asegurar que el tamaño de la imagen resultante sea el mismo que el tamaño de
# Se añaden píxeles de valor cero alrededor de la imagen antes de la convolución. Esto tiene el efecto de mantener la resolución espacial
# preservar los bordes y detalles.

# Padding = 'VALID': En este caso, no se aplica ningún relleno a la imagen antes de la convolución. La convolución se realiza solo en las
# que tienen suficientes píxeles para alinear completamente el kernel. Esto puede resultar en una reducción del tamaño de la imagen resul
# con la imagen original.
```

Sección 4: Aplicación de ReLu

1. En las siguientes lineas de código aplicaremos la función ReLu a la imagen resultante de la convolución.

```
kernel_ejemplo = tf.constant([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1],
])
imagen_resultante_convolucion = ejecutar_convolucion_en_imagen(imagen_prueba,kernel_ejemplo)

#aplicando ReLu
imagen_resultante_relu = tf.nn.relu(imagen_resultante_convolucion)

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

Pregunta 8: Compare el valores almacenados en las imagenes: "imagen_resultante_convolucion" y "imagen_resultante_relu" logra identificar como afecta el uso de la relu en la imagen?

```
imagen_resultante_convolucion
```

```
<tf.Tensor: shape=(1, 359, 419, 1), dtype=float32, numpy=
array([[[[-2.7451336e-02],
         [ 1.7881393e-07],
         [-3.9213300e-03],
         ...,
         [-7.8430772e-03],
         [-3.9215684e-03],
         [-3.9215684e-03]],

        [[ 7.8427196e-03],
         [ 1.9607663e-02],
         [ 3.9213896e-03],
         ...,
         [-3.9215088e-03],
         [ 2.3529470e-02],
         [ 1.5686452e-02]],

        [[-3.9218068e-03],
         [ 7.8431964e-03],
         [ 3.1372547e-02],
         ...,
         [-1.1764526e-02],
         [-1.5686035e-02],
         [ 1.5686512e-02]],

        ...,

        [[-1.1764705e-02],
         [-1.1764705e-02],
         [-1.1764705e-02],
         ...,
         [-1.1920929e-07],
         [-1.1920929e-07],
```

```
               [-1.1920929e-07]],

              [[ 1.1764884e-02],
               [ 1.1764884e-02],
               [ 1.1764884e-02],
               ...,
               [-1.1920929e-07],
               [ 3.9214492e-03],
               [ 7.8429580e-03]],

              [[-3.5294056e-02],
               [-3.5294056e-02],
               [-3.5294056e-02],
               ...,
               [-3.9216280e-03],
               [ 7.8429580e-03],
               [-2.3529649e-02]]]], dtype=float32)>
```

imagen_resultante_relu

```
     <tf.Tensor: shape=(1, 359, 419, 1), dtype=float32, numpy=
     array([[[[0.0000000e+00],
              [1.7881393e-07],
              [0.0000000e+00],
              ...,
              [0.0000000e+00],
              [0.0000000e+00],
              [0.0000000e+00]],

             [[7.8427196e-03],
              [1.9607663e-02],
              [3.9213896e-03],
              ...,
              [0.0000000e+00],
              [2.3529470e-02],
              [1.5686452e-02]],

             [[0.0000000e+00],
              [7.8431964e-03],
              [3.1372547e-02],
              ...,
              [0.0000000e+00],
              [0.0000000e+00],
              [1.5686512e-02]],

             ...,

             [[0.0000000e+00],
              [0.0000000e+00],
              [0.0000000e+00],
              ...,
              [0.0000000e+00],
              [0.0000000e+00],
              [0.0000000e+00]],

             [[1.1764884e-02],
              [1.1764884e-02],
              [1.1764884e-02],
              ...,
              [0.0000000e+00],
              [3.9214492e-03],
              [7.8429580e-03]],

             [[0.0000000e+00],
              [0.0000000e+00],
              [0.0000000e+00],
              ...,
              [0.0000000e+00],
              [7.8429580e-03],
              [0.0000000e+00]]]], dtype=float32)>
```

```
# En este caso, la imagen resultante de la convolución se pasa a través de la función ReLU para resaltar los píxeles que tienen valores p
# y descartar los píxeles con valores negativos. El resultado es una imagen que resalta aún más los bordes y características distintivas
# en la imagen original.
#
# A comparación de la imagen original que considera los pixeles negativos
#
# Podemos observar cómo se realzan los bordes y detalles en la imagen, ya que los píxeles negativos se han establecido en cero.
```

Sección 4: Aplicación de pooling

```
imagen_resultante_pooling = tf.nn.pool(
    input=imagen_resultante_relu,
```

```
    window_shape=(2,2),
    pooling_type='MAX',
    strides=(2, 2),
    padding='SAME',
)

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

Pregunta 9: Compare el valores almacenados en las imagenes: "imagen_resultante_relu" e "imagen_resultante_pooling" logra identificar como afecta el uso del pooling en la imagen?

```
imagen_resultante_relu
```

```
    <tf.Tensor: shape=(1, 359, 419, 1), dtype=float32, numpy=
    array([[[[0.0000000e+00],
             [1.7881393e-07],
             [0.0000000e+00],
             ...,
             [0.0000000e+00],
             [0.0000000e+00],
             [0.0000000e+00]],

            [[7.8427196e-03],
             [1.9607663e-02],
             [3.9213896e-03],
             ...,
             [0.0000000e+00],
             [2.3529470e-02],
             [1.5686452e-02]],

            [[0.0000000e+00],
             [7.8431964e-03],
             [3.1372547e-02],
             ...,
             [0.0000000e+00],
             [0.0000000e+00],
             [1.5686512e-02]],

            ...,

            [[0.0000000e+00],
             [0.0000000e+00],
             [0.0000000e+00],
             ...,
             [0.0000000e+00],
             [0.0000000e+00],
             [0.0000000e+00]],

            [[1.1764884e-02],
             [1.1764884e-02],
             [1.1764884e-02],
             ...,
             [0.0000000e+00],
```

```
               [3.9214492e-03],
               [7.8429580e-03]],

              [[0.0000000e+00],
               [0.0000000e+00],
               [0.0000000e+00],
               ...,
               [0.0000000e+00],
               [7.8429580e-03],
               [0.0000000e+00]]]], dtype=float32)>
```

```
imagen_resultante_pooling
```

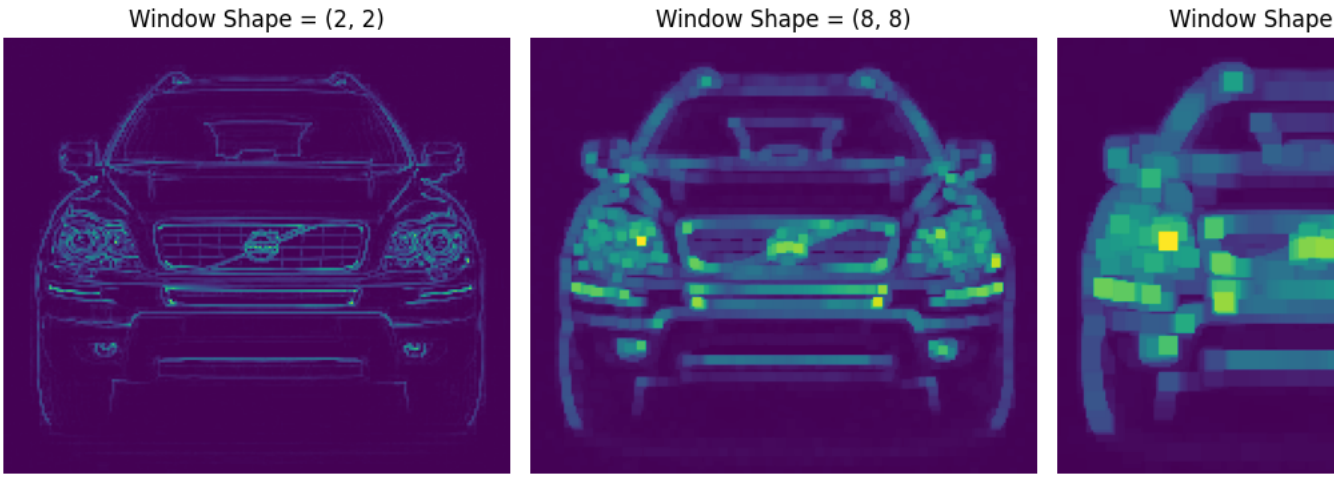
```
     <tf.Tensor: shape=(1, 180, 210, 1), dtype=float32, numpy=
     array([[[[0.01960766],
              [0.00392139],
              [0.00784314],
              ...,
              [0.        ],
              [0.02352947],
              [0.01568645]],

             [[0.0078432 ],
              [0.03137255],
              [0.03137249],
              ...,
              [0.0078432 ],
              [0.0078432 ],
              [0.01568651]],

             [[0.01568609],
              [0.00784314],
              [0.01568615],
              ...,
              [0.01568639],
              [0.02352947],
              [0.00392187]],

             ...,

             [[0.01176435],
              [0.01176435],
              [0.01176435],
              ...,
              [0.00784314],
              [0.01568615],
              [0.        ]],

             [[0.01176488],
              [0.01176488],
              [0.01176488],
              ...,
              [0.01568615],
              [0.00392145],
              [0.00784296]],

             [[0.        ],
              [0.        ],
              [0.        ],
              ...,
              [0.00392139],
              [0.00784296],
              [0.        ]]]], dtype=float32)>
```

```
# imagen_resultante_relu : Resaltan los píxeles que tienen valores positivos y descartar los píxeles con valores negativos.
# El resultado es una imagen que resalta aún más los bordes y características distintivas presentes  en la imagen original.

# imagen_resultante_polling : El resultado es una imagen con una resolución espacial reducida, ya que se han tomado los valores máximos d
# El pooling máximo tiende a resaltar las características más prominentes en la imagen, preservando la información más relevante mientras

# Por último Puedes observar cómo la resolución espacial se ha reducido y cómo se han conservado las características distintivas de la im
```

Pregunta 10: Compare los resultados obtenidos variando el parametro window_shape, pruebe con valores de (2,2), (8,8), (16,16), como estos parametros afectan la apariencia y el tamanio de la imagen_resultante_pooling.

```
window_shapes = [(2,2), (8,8), (16,16)]

plt.figure(figsize=(12, 4))
for i, window_shape in enumerate(window_shapes):
    imagen_resultante_pooling = tf.nn.pool(
```

```
        input=imagen_resultante_relu,
        window_shape=window_shape,
        pooling_type='MAX',
        strides=(2, 2),
        padding='SAME',
    )

    # Mostrar imagen resultante
    plt.subplot(1, len(window_shapes), i+1)
    plt.imshow(tf.squeeze(imagen_resultante_pooling))
    plt.axis('off')
    plt.title(f"Window Shape = {window_shape}")

plt.tight_layout()
plt.show()
```




```
# A medida que el valor de window_shape aumenta, la imagen resultante tendrá una menor resolución espacial y un tamaño más pequeño.
# Si queremos mas detalle deberemos de tener una ventana de pooling más pequeño, se conservará más detalle y nivel de información en la i
```

Pregunta 11: Compare los resultados obtenidos variando el parametro pooling_type, pruebe con valores 'MAX', 'AVG' como estos parametros afectan la apariencia de la imagen_resultante_pooling.

```
pooling_type_values = ['MAX','AVG']

plt.figure(figsize=(12, 4))
for i, pool_type in enumerate(pooling_type_values):
    imagen_resultante_pooling = tf.nn.pool(
        input=imagen_resultante_relu,
        window_shape=(2,2),
        pooling_type=pool_type,
        strides=(2, 2),
        padding='SAME',
    )

    # Mostrar imagen resultante
    plt.subplot(1, len(window_shapes), i+1)
    plt.imshow(tf.squeeze(imagen_resultante_pooling))
    plt.axis('off')
    plt.title(f"Pooling type = {pool_type}")

plt.tight_layout()
plt.show()
```


```
# Pooling máximo ('MAX'): Con el pooling máximo, se selecciona el valor máximo dentro de cada región de la ventana de pooling.
# Esto tiende a resaltar características prominentes y bordes en la imagen. En la imagen resultante, se pueden observar regiones más
# oscuras o de alto contraste donde se encuentran las características más distintivas.

# Pooling promedio ('AVG'): Con el pooling promedio, se calcula el promedio de los valores dentro de cada región de la ventana de pooling
# Esto suaviza la imagen y reduce el contraste entre las características. En la imagen resultante, las transiciones entre las caracterís
# son más suaves y se observa una apariencia más uniforme en la imagen.
```

Sección 5: Primera red convolucional

para completar el laboratorio vamos a usar el conjunto de digitos de mnist para crear una red convolucional simple, incluyendo todos los componentes estudiados

```
#instalando el paquete que tiene imagenes
%pip install mnist
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: mnist in /usr/local/lib/python3.10/dist-packages (0.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from mnist) (1.22.4)
```

```
import numpy as np
import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.utils import to_categorical

train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()

# Normalizizando las imagenes.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

# Redimensionando las imagenes.
train_images = np.expand_dims(train_images, axis=3)
test_images = np.expand_dims(test_images, axis=3)

num_filters = 8
filter_size = 3
pool_size = 2

# Construyendo el modelo.
model1 = Sequential([
  Conv2D(num_filters, filter_size, input_shape=(28, 28, 1)),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(10, activation='softmax'),
])

# Configurando el model.
model1.compile(
  'adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)
```

Pregunta 12: cual es la dimensión de los conjuntos de entrenamiento y test (train_images, test_images)?

```
print("La dimension es : " , train_images.shape)
print("La dimension es : " , test_images.shape)
```

```
La dimension es :  (60000, 28, 28, 1)
La dimension es :  (10000, 28, 28, 1)
```

Pregunta 13: Cuantas imagenes tiene en conjunto de entrenamiento y el conjunto de test, y de que tamanio son las imagenes?

```
# El conjunto de entrenamiento contiene 60,000 imágenes, mientras que el conjunto de prueba contiene 10,000 imágenes.
# Estas imágenes son imágenes en escala de grises de tamaño 28x28 píxeles.
```

```
# Entrenando el modelo.
history1=model1.fit(
  train_images,
  to_categorical(train_labels),
  epochs=3,
  validation_data=(test_images, to_categorical(test_labels)),
)
```

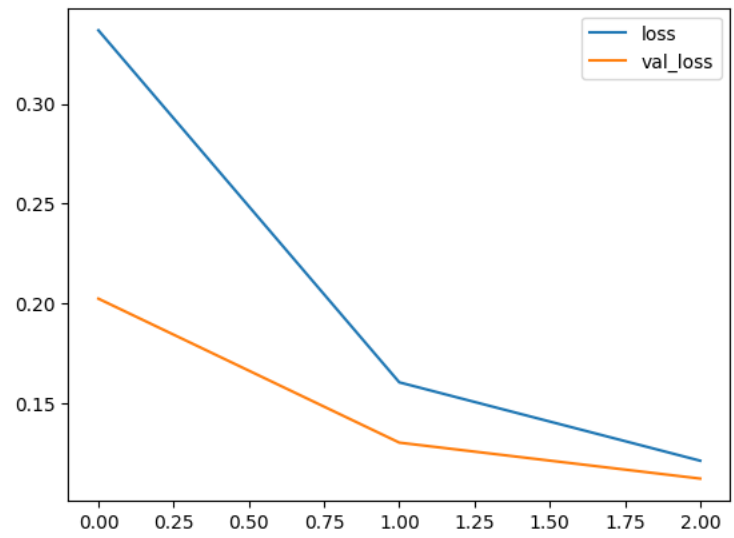
```
Epoch 1/3
1875/1875 [==============================] - 24s 12ms/step - loss: 0.3368 - accuracy: 0.9037 - val_loss: 0.2022 - val_accuracy: 0.9
Epoch 2/3
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1603 - accuracy: 0.9544 - val_loss: 0.1301 - val_accuracy: 0.9
Epoch 3/3
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1210 - accuracy: 0.9653 - val_loss: 0.1120 - val_accuracy: 0.9
```

```
import pandas as pd
#rescatando el historico del entrenamiento
history_frame = pd.DataFrame(history1.history)
history_frame
```

	loss	accuracy	val_loss	val_accuracy
0	0.336816	0.903717	0.202248	0.9407
1	0.160295	0.954400	0.130065	0.9626
2	0.120993	0.965283	0.112030	0.9657

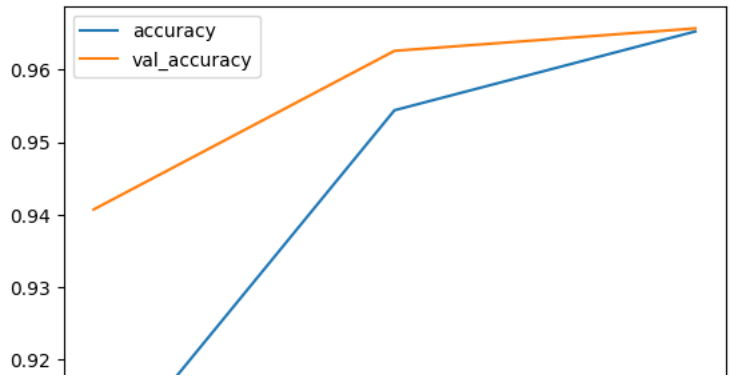
```
#mostrando el historico de loss de entrenamiento vs validación
history_frame.loc[:, ['loss','val_loss']].plot()
```

```
<Axes: >
```




```
#mostrando el historico de accuracy de entrenamiento vs validación
history_frame.loc[:, ['accuracy','val_accuracy']].plot()
```

```
<Axes: >
```



Pregunta 13: Por qué usamos el método tf.expand_dims antes de hacer la predicción en las siguientes lineas de código?

```
# Predicción usando imagenes de test

# indice de la imagem
indice=0
imagen_test =  tf.expand_dims(test_images[indice], axis=0)
prediction = model1.predict(imagen_test)

# Resultado de prediccion
print("Probabilidades")
print(prediction)

print("Mayor probabilidad")
print(np.argmax(prediction, axis=1))

# Etiqueta real de la imagen
print("Etiqueta real")
print(test_labels[indice])
```

```
1/1 [==============================] - 0s 42ms/step
Probabilidades
[[2.8309469e-06 2.4879991e-08 4.2914676e-06 6.2373321e-05 5.7485551e-07
  6.3787546e-08 1.5336048e-11 9.9992669e-01 5.0351076e-08 3.0276019e-06]]
Mayor probabilidad
[7]
Etiqueta real
7
```

```
# tf.expand_dims() se utiliza antes de hacer la predicción en las siguientes líneas de código con el fin de ajustar
# la forma de la imagen de prueba (imagen_test) para que coincida con la forma de entrada requerida por el modelo.

# Al utilizar tf.expand_dims(), se agrega una dimensión adicional al tensor de la imagen de prueba en la posición especificada
# por el argumento axis. En este caso, axis=0 indica que se agrega una dimensión al principio del tensor, convirtiéndolo en un
# tensor de forma (1, height, width, channels), lo que corresponde a un tamaño de lote (batch) de 1.
```

Pregunta 14: por que no usamos el método tf.expand_dims antes de hacer la prediccines de en la siguiente linea de código?

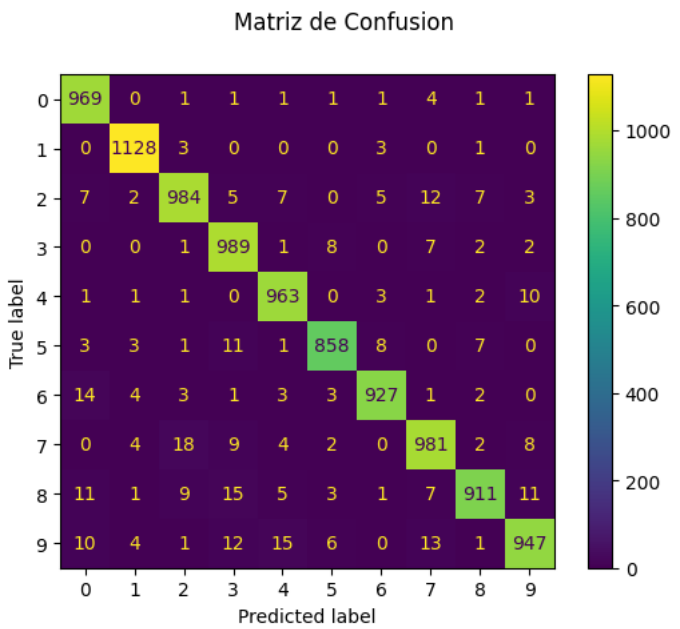
```
# Esto se debe a que test_images ya tiene la forma adecuada para la entrada al modelo.
# Por lo tanto, cuando se realiza la predicción con model1.predict(test_images), no es necesario utilizar
# tf.expand_dims() nuevamente, ya que test_images ya tiene la forma adecuada para ser ingresada al modelo.


predictions_probabilities = model1.predict(test_images)
```

```
313/313 [==============================] - 1s 4ms/step
```

Pregunta 15: Cuál es la función del método argmax en la siguiente linea de código?

```
# se obtiene un arreglo de índices que representa las etiquetas predichas para cada imagen de prueba. Estos índices se utilizan
# posteriormente en la construcción de la matriz de confusión
```

```
predicted_labels=np.argmax(predictions_probabilities, axis=1)


# mostrando matriz de confusion de los resultados de la predicción vs las etiquetas reales o verdaderas
from sklearn import metrics

disp = metrics.ConfusionMatrixDisplay.from_predictions(test_labels, predicted_labels)
disp.figure_.suptitle("Matriz de Confusion")
plt.show()
```



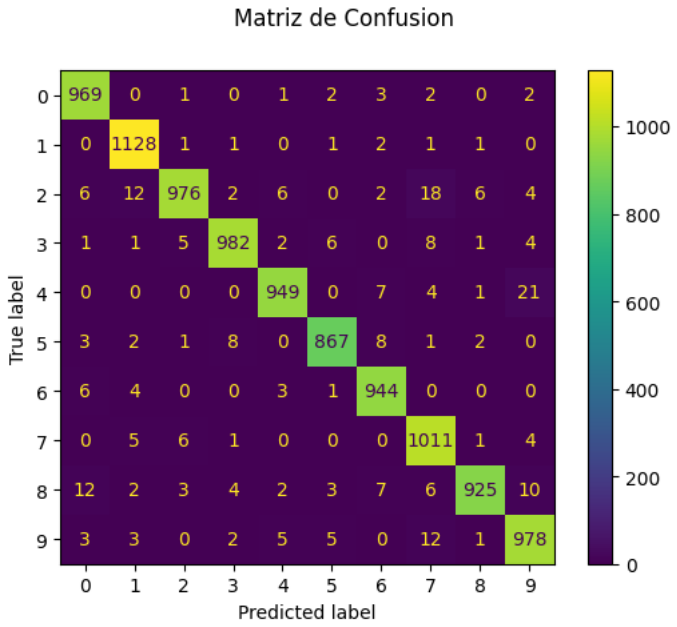
Pregunta 16: Cuál es el efecto de agregar una capa convolucional a la red previamente creada?

```
num_filters = 8
filter_size = 3
pool_size = 2

# Construyendo el modelo.
model2 = Sequential([
  Conv2D(num_filters, filter_size, input_shape=(28, 28, 1)),
  Conv2D(num_filters, filter_size),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(10, activation='softmax'),
])

# Configurando el model.
model2.compile(
  'adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)

# Entrenando el modelo.
history2=model2.fit(
  train_images,
  to_categorical(train_labels),
  epochs=3,
  validation_data=(test_images, to_categorical(test_labels)),
)
```

```
Epoch 1/3
1875/1875 [==============================] - 42s 22ms/step - loss: 0.2690 - accuracy: 0.9212 - val_loss: 0.1161 - val_accuracy: 0.9
Epoch 2/3
1875/1875 [==============================] - 25s 13ms/step - loss: 0.1123 - accuracy: 0.9671 - val_loss: 0.0847 - val_accuracy: 0.9
Epoch 3/3
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0894 - accuracy: 0.9736 - val_loss: 0.0880 - val_accuracy: 0.9
```

```
#predicción
predictions_probabilities2 = model2.predict(test_images)
predicted_labels2=np.argmax(predictions_probabilities2, axis=1)
# mostrando matriz de confusion de los resultados de la predicción vs las etiquetas reales o verdaderas
disp = metrics.ConfusionMatrixDisplay.from_predictions(test_labels, predicted_labels2)
disp.figure_.suptitle("Matriz de Confusion")
plt.show()
```

```
313/313 [==============================] - 3s 8ms/step
```




```
# La adición de una capa convolucional adicional permite que la red aprenda representaciones más complejas y sofisticadas
# de las características presentes en las imágenes de entrada. Esto puede resultar en una mayor capacidad de discriminación y
# en la capacidad de capturar patrones más finos y detalles relevantes en los datos.
```

✓ 0 s completado a las 13:30