Hoja informativa: Redes neuronales convolucionales

Práctica

```
# convolución unidimensional

def convolve(sequence, weights):
    convolution = np.zeros(len(sequence) - len(weights) + 1)
    for i in range(convolution.shape[0]):
        convolution[i] = np.sum(weights * sequence[i:i + len(weights)])
```

```
# capa convolucional bidimensional en keras
# filters: el número de filtros que corresponde al tamaño del tensor de output.
# kernel_size: el tamaño espacial del filtro K.
# strides: determina cuán lejos se desplaza el filtro sobre la matriz de input (establecid
o en 1 de forma predeterminada.)
# padding: define el ancho del padding cero.
# Hay dos tipos de padding: valid (válido) y **same* (igual).
# El tipo predeterminado de padding es valid y es igual a cero.
# Same establece el tamaño del padding de forma automática,
# de modo que el ancho y alto del tensor de output sea igual al ancho y alto del tensor de
input.
# activation: esta función se aplica inmediatamente después de la convolución.

keras.layers.Conv2D(filters, kernel_size, strides, padding, activation)
```

```
# la capa Flatten hace que el tensor multidimensional sea unidimensional

keras.layers.Flatten()
```

```
# Capa Pooling
# pool_size: tamaño de pooling (agrupación).
# strides: un paso (o stride) determina cuán lejos se desplaza el filtro sobre la matriz d
e input. Si se especifica *None*, el paso es igual al tamaño de pooling.
# padding: define el ancho del padding cero.

keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', ...)
keras.layers.AveragePooling2D(pool_size=(2, 2), strides=None, padding='valid', ...)
```

```
# generador de datos

from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator()

# extraer datos de una carpeta
datagen_flow = datagen.flow_from_directory(
    # carpeta con el conjunto de datos
    '/dataset/',
    # tamaño de la imagen objetivo
    target_size=(150, 150),
    # tamaño del lote
    batch_size=16,
    # modo de clase
    class_mode='sparse',
    # indica que este es el generador de datos para el conjunto de entrenamiento (si es ne
cesario)
    subset='training',
    # indica que este es el generador de datos para el conjunto de validación (si es neces
ario)
    subset='validation',
    # configurar un generador de números aleatorios
    seed=12345)

# obtener el siguiente lote
features, target = next(datagen_flow)

# entrenar el modelo mediante generadores de datos
# utilizando el conjunto de datos completo
model.fit(datagen_flow, steps_per_epoch=len(datagen_flow))

# solo con conjuntos de entrenamiento y validación
model.fit(train_datagen_flow,
          validation_data=val_datagen_flow,
          steps_per_epoch=len(train_datagen_flow),
          validation_steps=len(val_datagen_flow))
```

```
# Agregar aumento en generadores de datos
# horizontal_flip: rotación horizontal
# vertical_flip: rotación vertical

datagen = ImageDataGenerator(validation_split=0.25,
                             rescale=1./255,
                             horizontal_flip=True,
                             vertical_flip=True)
```

```
# Implementación de la arquitectura ResNet
# input_shape: tamaño de la imagen de input
# classes=1000: el número de neuronas en la última capa totalmente conectada donde sucede
 la clasificación
# weights='imagenet': la inicialización de los pesos
# ImageNet: el nombre de una gran base de datos de imágenes que
# se usaba para entrenar a la red para clasificar imágenes en 1000 clases
# Escribe weights=None para inicializar los pesos al azar.
# include_top=True: indica que hay dos capas al final de ResNet:
# GlobalAveragePooling2D y Dense. Si la configuras en False, estas dos capas no estarán pr
esentes.

from tensorflow.keras.applications.resnet import ResNet50

model = ResNet50(input_shape=None,
                 classes=1000,
                 include_top=True,
                 weights='imagenet')
```

```
# Creación de tu propia red basada en ResNet50

backbone = ResNet50(input_shape=(150, 150, 3),
                    weights='imagenet',
                    include_top=False)

# congela ResNet50 sin la parte superior (opcional)
backbone.trainable = False

model = Sequential()
model.add(backbone)
model.add(GlobalAveragePooling2D())
model.add(Dense(12, activation='softmax'))
```

Teoría

Convolución: una función que aplica las mismas operaciones a todos los pixeles para extraer elementos de imagen que son importantes para la clasificación.

Redes neuronales convolucionales: una clase de redes neuronales que utilizan capas convolucionales y hacen la mayor parte del cómputo dentro de la red.

Capa de convolución: una capa donde la operación de convolución se aplica a las imágenes de input.

Filtro: componente de la capa convolucional, un conjunto de pesos que se aplican a la imagen.

Padding: esta configuración coloca ceros en los bordes de la matriz (padding cero) de modo que los pixeles más alejados participen en la convolución al menos la misma cantidad de veces que los pixeles centrales.

Striding o **stride** (paso): esta configuración desplaza el filtro en más de un pixel y genera una imagen de output más pequeña.

Pooling: compactar un grupo de pixeles en un pixel usando alguna transformación; por ejemplo, calculando el máximo o la media aritmética.

Aumento: una técnica que se usa para expandir artificialmente un conjunto de datos al transformar las imágenes existentes. Los cambios solo se aplican a los conjuntos de entrenamiento, mientras que los de prueba y validación se quedan igual. El aumento transforma la imagen original, pero aún preserva sus características principales.