UNet

January 17, 2021

1 Práctica 5

1.1 Redes generadoras: UNet

1.1.1 Marc Balle Sánchez

Se cargan aquellas librerías de interés para la práctica.

[1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv2DTranspose,␣
↪BatchNormalization, Concatenate
from keras.utils.vis_utils import plot_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,␣
↪img_to_array
```

[2]:
```
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

Se procede a descargar el conjunto de datos y organizarlo en una estructura de directorios apta para el desarrollo del problema.

[5]:
```
!cd /content
!wget https://github.com/miquelmn/visio_per_computador/raw/master/in/DL/data.zip
```

```
--2021-01-17 15:46:05--
https://github.com/miquelmn/visio_per_computador/raw/master/in/DL/data.zip
Resolving github.com (github.com)… 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: https://raw.githubusercontent.com/miquelmn/visio_per_computador/master
/in/DL/data.zip [following]
--2021-01-17 15:46:05--  https://raw.githubusercontent.com/miquelmn/visio_per_co
```

```
mputador/master/in/DL/data.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
151.101.0.133, 151.101.64.133, 151.101.128.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|151.101.0.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 13539260 (13M) [application/zip]
Saving to: 'data.zip'

data.zip            100%[===================>]  12.91M  --.-KB/s    in 0.09s

2021-01-17 15:46:06 (139 MB/s) - 'data.zip' saved [13539260/13539260]
```

[6]:
```
!unzip data.zip
```

```
Archive:  data.zip
   creating: test/
  inflating: test/0.png
  inflating: test/1.png
  inflating: test/10.png
  inflating: test/11.png
  inflating: test/12.png
  inflating: test/13.png
  inflating: test/14.png
  inflating: test/15.png
  inflating: test/16.png
  inflating: test/17.png
  inflating: test/18.png
  inflating: test/19.png
  inflating: test/2.png
  inflating: test/20.png
  inflating: test/21.png
  inflating: test/22.png
  inflating: test/23.png
  inflating: test/24.png
  inflating: test/25.png
  inflating: test/26.png
  inflating: test/27.png
  inflating: test/28.png
  inflating: test/29.png
  inflating: test/3.png
  inflating: test/4.png
  inflating: test/5.png
  inflating: test/6.png
  inflating: test/7.png
  inflating: test/8.png
  inflating: test/9.png
```

```
 creating: train/
 creating: train/imgs/
inflating: train/imgs/0.png
inflating: train/imgs/1.png
inflating: train/imgs/10.png
inflating: train/imgs/11.png
inflating: train/imgs/12.png
inflating: train/imgs/13.png
inflating: train/imgs/14.png
inflating: train/imgs/15.png
inflating: train/imgs/16.png
inflating: train/imgs/17.png
inflating: train/imgs/18.png
inflating: train/imgs/19.png
inflating: train/imgs/2.png
inflating: train/imgs/20.png
inflating: train/imgs/21.png
inflating: train/imgs/22.png
inflating: train/imgs/23.png
inflating: train/imgs/24.png
inflating: train/imgs/25.png
inflating: train/imgs/26.png
inflating: train/imgs/27.png
inflating: train/imgs/28.png
inflating: train/imgs/29.png
inflating: train/imgs/3.png
inflating: train/imgs/4.png
inflating: train/imgs/5.png
inflating: train/imgs/6.png
inflating: train/imgs/7.png
inflating: train/imgs/8.png
inflating: train/imgs/9.png
 creating: train/labels/
inflating: train/labels/0.png
inflating: train/labels/1.png
inflating: train/labels/10.png
inflating: train/labels/11.png
inflating: train/labels/12.png
inflating: train/labels/13.png
inflating: train/labels/14.png
inflating: train/labels/15.png
inflating: train/labels/16.png
inflating: train/labels/17.png
inflating: train/labels/18.png
inflating: train/labels/19.png
inflating: train/labels/2.png
inflating: train/labels/20.png
inflating: train/labels/21.png
```

```
  inflating: train/labels/22.png
  inflating: train/labels/23.png
  inflating: train/labels/24.png
  inflating: train/labels/25.png
  inflating: train/labels/26.png
  inflating: train/labels/27.png
  inflating: train/labels/28.png
  inflating: train/labels/29.png
  inflating: train/labels/3.png
  inflating: train/labels/4.png
  inflating: train/labels/5.png
  inflating: train/labels/6.png
  inflating: train/labels/7.png
  inflating: train/labels/8.png
  inflating: train/labels/9.png
```

[7]:
```
mkdir data_unet
```

[8]:
```
!mv test train data_unet
```

[12]:
```
!cd /content/data_unet/train/imgs/
```

[14]:
```
mkdir /content/data_unet/train/imgs/images
```

[17]:
```
!mv /content/data_unet/train/imgs/*.png /content/data_unet/train/imgs/images
```

[18]:
```
!mkdir /content/data_unet/train/labels/etiquetas
```

[19]:
```
!mv /content/data_unet/train/labels/*.png /content/data_unet/train/labels/
 ↪etiquetas
```

Se diseña la arquitectura del modelo. En concreto la parte del *decoder*. En este caso se ha obtado por emplear la convolución traspuesta en lugar del *upsampling* + convolución. Esto es así únicamente por un mayor entendimiento teórico de la convolución traspuesta.

En esta parte es interesante comentar que en la capa de salida se seleccionan dos filtros de convolución 1x1. Al principio se escogió solamente uno, siguiendo el ejemplo de algunos modelos encontrados por internet. Sin embargo el entrenamiento del modelo era pésimo. Observando la representación gráfica típica de la UNet del artículo original de Ronneberger et al. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, se comprobó que realmente son dos filtros en la capa final. Al modificar dicho número todo funcionó mucho mejor. Entiendo que este número se debe a que la imagen de salida es binaria, clasificando los píxeles como 0 o 1.

[3]:
```
def unet(n_filters=16, bn=True, dilation_rate=1, input_size=(512, 512, 1),
         output_channels=3, loss_func="categorical_crossentropy"):
  #loss_func, output_channels se podrían quitar de la funcion
    """
    Creates the U-Net Model.
```

```python
    The U-Net neural network is a model introduced by Ronneberger et al. at
    2015. This method builds the neural network. This network is an
    encoder-decoder network, also known as a FCN network. We introduce as a
    method to improve the learning process multiple layers of batch
    normalization.
    TODO:
        Add the decoder, the part that creates the new image.
    Returns:
        Model object with all the layers.
    """
    # Define input batch shape
    inputs = keras.Input(input_size)

    conv1 = Conv2D(n_filters * 1, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(inputs)
    if bn:
        conv1 = BatchNormalization()(conv1)

    conv1 = Conv2D(n_filters * 1, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(conv1)
    if bn:
        conv1 = BatchNormalization()(conv1)

    pool1 = MaxPooling2D(pool_size=(2, 2), data_format='channels_last')(conv1)

    conv2 = Conv2D(n_filters * 2, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(pool1)
    if bn:
        conv2 = BatchNormalization()(conv2)

    conv2 = Conv2D(n_filters * 2, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(conv2)
    if bn:
        conv2 = BatchNormalization()(conv2)

    pool2 = MaxPooling2D(pool_size=(2, 2), data_format='channels_last')(conv2)

    conv3 = Conv2D(n_filters * 4, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(pool2)
    if bn:
        conv3 = BatchNormalization()(conv3)

    conv3 = Conv2D(n_filters * 4, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(conv3)
    if bn:
        conv3 = BatchNormalization()(conv3)
```

```
    pool3 = MaxPooling2D(pool_size=(2, 2), data_format='channels_last')(conv3)

    conv4 = Conv2D(n_filters * 8, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(pool3)
    if bn:
        conv4 = BatchNormalization()(conv4)

    conv4 = Conv2D(n_filters * 8, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(conv4)
    if bn:
        conv4 = BatchNormalization()(conv4)

    pool4 = MaxPooling2D(pool_size=(2, 2), data_format='channels_last')(conv4)

    conv5 = Conv2D(n_filters * 16, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(pool4)
    if bn:
        conv5 = BatchNormalization()(conv5)

    conv5 = Conv2D(n_filters * 16, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(conv5)
    if bn:
        conv5 = BatchNormalization()(conv5)


    # Decoder

    tconv6 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2), padding =␣
↪'same')(conv5)
    tconv6 = Concatenate()([tconv6, conv4]) #en teoria se concatenan en el eje Z
  #  u6 = Dropout(dropout)(u6)
    conv6 = Conv2D(n_filters * 8, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(tconv6)
    if bn:
        conv6 = BatchNormalization()(conv6)

    conv6 =  Conv2D(n_filters * 8, (3, 3), activation='relu', padding='same',
                   dilation_rate=dilation_rate)(conv6)

    if bn:
        conv6 = BatchNormalization()(conv6)

    tconv7 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2), padding =␣
↪'same')(conv6)
    tconv7 = Concatenate()([tconv7, conv3])
  #  u6 = Dropout(dropout)(u6)
    conv7 = Conv2D(n_filters * 4, (3, 3), activation='relu', padding='same',
```

```
                    dilation_rate=dilation_rate)(tconv7)
    if bn:
        conv7 = BatchNormalization()(conv7)

    conv7 =  Conv2D(n_filters * 4, (3, 3), activation='relu', padding='same',
                    dilation_rate=dilation_rate)(conv7)

    if bn:
        conv7 = BatchNormalization()(conv7)


    tconv8 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2), padding =␣
↪'same')(conv7)
    tconv8 = Concatenate()([tconv8, conv2])
 #  u6 = Dropout(dropout)(u6)
    conv8 = Conv2D(n_filters * 2, (3, 3), activation='relu', padding='same',
                    dilation_rate=dilation_rate)(tconv8)
    if bn:
        conv8 = BatchNormalization()(conv8)

    conv8 =  Conv2D(n_filters * 2, (3, 3), activation='relu', padding='same',
                    dilation_rate=dilation_rate)(conv8)

    if bn:
        conv8= BatchNormalization()(conv8)


    tconv9 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2), padding =␣
↪'same')(conv8)
    tconv9 = Concatenate()([tconv9, conv1])
 #  u6 = Dropout(dropout)(u6)
    conv9 = Conv2D(n_filters * 1, (3, 3), activation='relu', padding='same',
                    dilation_rate=dilation_rate)(tconv9)
    if bn:
        conv9 = BatchNormalization()(conv9)

    conv9 =  Conv2D(n_filters * 1, (3, 3), activation='relu', padding='same',
                    dilation_rate=dilation_rate)(conv9)

    if bn:
        conv9= BatchNormalization()(conv9)


    outputs = Conv2D(output_channels, (1, 1), padding = 'same',␣
↪activation='sigmoid')(conv9) #salida
    model = keras.Model(inputs, outputs)
    return model
```

Se compila el modelo escogiendo como función de pérdida la *binary cross entropy* por tratarse de imágenes binarias en la salida de la red.

[12]:
```
model = unet(n_filters=16, bn=True, output_channels = 2, dilation_rate=1,␣
 ↪input_size=(512, 512, 1))
model.compile(loss="binary_crossentropy", metrics=["accuracy"])
```

[5]:
```
model.summary()
```

```
Model: "model"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 512, 512, 1) 0
________________________________________________________________________________
__________________
conv2d (Conv2D)                 (None, 512, 512, 16) 160         input_1[0][0]
________________________________________________________________________________
__________________
batch_normalization (BatchNorma (None, 512, 512, 16) 64          conv2d[0][0]
________________________________________________________________________________
__________________
conv2d_1 (Conv2D)               (None, 512, 512, 16) 2320
batch_normalization[0][0]
________________________________________________________________________________
__________________
batch_normalization_1 (BatchNor (None, 512, 512, 16) 64          conv2d_1[0][0]
________________________________________________________________________________
__________________
max_pooling2d (MaxPooling2D)    (None, 256, 256, 16) 0
batch_normalization_1[0][0]
________________________________________________________________________________
__________________
conv2d_2 (Conv2D)               (None, 256, 256, 32) 4640
max_pooling2d[0][0]
________________________________________________________________________________
__________________
batch_normalization_2 (BatchNor (None, 256, 256, 32) 128         conv2d_2[0][0]
________________________________________________________________________________
__________________
conv2d_3 (Conv2D)               (None, 256, 256, 32) 9248
batch_normalization_2[0][0]
________________________________________________________________________________
__________________
batch_normalization_3 (BatchNor (None, 256, 256, 32) 128         conv2d_3[0][0]
________________________________________________________________________________
```

```
------------------
max_pooling2d_1 (MaxPooling2D)  (None, 128, 128, 32) 0
batch_normalization_3[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_4 (Conv2D)               (None, 128, 128, 64) 18496
max_pooling2d_1[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_4 (BatchNor (None, 128, 128, 64) 256         conv2d_4[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_5 (Conv2D)               (None, 128, 128, 64) 36928
batch_normalization_4[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_5 (BatchNor (None, 128, 128, 64) 256         conv2d_5[0][0]
--------------------------------------------------------------------------------
------------------
max_pooling2d_2 (MaxPooling2D)  (None, 64, 64, 64)   0
batch_normalization_5[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_6 (Conv2D)               (None, 64, 64, 128)  73856
max_pooling2d_2[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_6 (BatchNor (None, 64, 64, 128)  512         conv2d_6[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_7 (Conv2D)               (None, 64, 64, 128)  147584
batch_normalization_6[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_7 (BatchNor (None, 64, 64, 128)  512         conv2d_7[0][0]
--------------------------------------------------------------------------------
------------------
max_pooling2d_3 (MaxPooling2D)  (None, 32, 32, 128)  0
batch_normalization_7[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_8 (Conv2D)               (None, 32, 32, 256)  295168
max_pooling2d_3[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_8 (BatchNor (None, 32, 32, 256)  1024        conv2d_8[0][0]
--------------------------------------------------------------------------------
------------------
```

```
conv2d_9 (Conv2D)              (None, 32, 32, 256)  590080
batch_normalization_8[0][0]

__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 32, 32, 256)  1024        conv2d_9[0][0]

__________________________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 64, 64, 128)  295040
batch_normalization_9[0][0]

__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 64, 64, 256)  0
conv2d_transpose[0][0]
batch_normalization_7[0][0]

__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 64, 64, 128)  295040
concatenate[0][0]

__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 64, 64, 128)  512         conv2d_10[0][0]

__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 64, 64, 128)  147584
batch_normalization_10[0][0]

__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 64, 64, 128)  512         conv2d_11[0][0]

__________________________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 128, 128, 64) 73792
batch_normalization_11[0][0]

__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 128, 128, 128 0
conv2d_transpose_1[0][0]
batch_normalization_5[0][0]

__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 128, 128, 64) 73792
concatenate_1[0][0]

__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 128, 128, 64) 256         conv2d_12[0][0]

__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 128, 128, 64) 36928
batch_normalization_12[0][0]
```

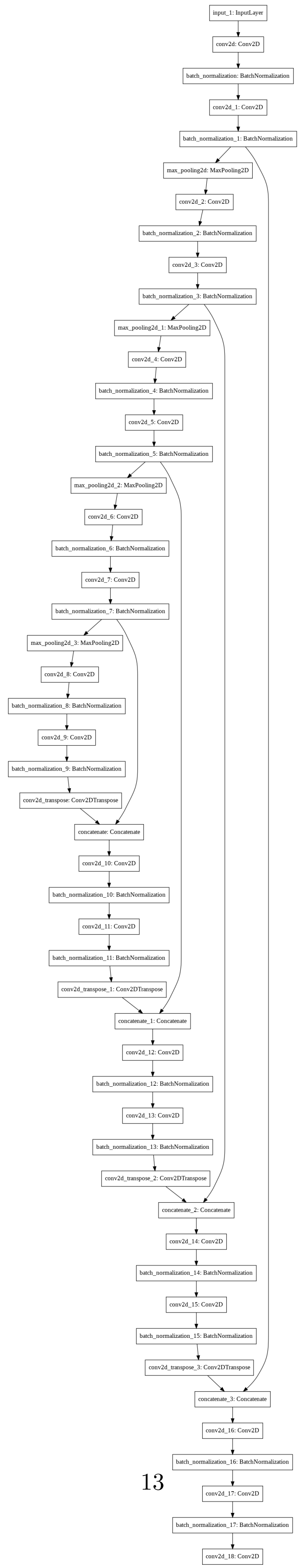
```
----------------------------------------------------------------------------------
------------------
batch_normalization_13 (BatchNo (None, 128, 128, 64) 256         conv2d_13[0][0]
----------------------------------------------------------------------------------
------------------
conv2d_transpose_2 (Conv2DTrans (None, 256, 256, 32) 18464
batch_normalization_13[0][0]
----------------------------------------------------------------------------------
------------------
concatenate_2 (Concatenate)     (None, 256, 256, 64) 0
conv2d_transpose_2[0][0]
batch_normalization_3[0][0]
----------------------------------------------------------------------------------
------------------
conv2d_14 (Conv2D)              (None, 256, 256, 32) 18464
concatenate_2[0][0]
----------------------------------------------------------------------------------
------------------
batch_normalization_14 (BatchNo (None, 256, 256, 32) 128         conv2d_14[0][0]
----------------------------------------------------------------------------------
------------------
conv2d_15 (Conv2D)              (None, 256, 256, 32) 9248
batch_normalization_14[0][0]
----------------------------------------------------------------------------------
------------------
batch_normalization_15 (BatchNo (None, 256, 256, 32) 128         conv2d_15[0][0]
----------------------------------------------------------------------------------
------------------
conv2d_transpose_3 (Conv2DTrans (None, 512, 512, 16) 4624
batch_normalization_15[0][0]
----------------------------------------------------------------------------------
------------------
concatenate_3 (Concatenate)     (None, 512, 512, 32) 0
conv2d_transpose_3[0][0]
batch_normalization_1[0][0]
----------------------------------------------------------------------------------
------------------
conv2d_16 (Conv2D)              (None, 512, 512, 16) 4624
concatenate_3[0][0]
----------------------------------------------------------------------------------
------------------
batch_normalization_16 (BatchNo (None, 512, 512, 16) 64          conv2d_16[0][0]
----------------------------------------------------------------------------------
------------------
conv2d_17 (Conv2D)              (None, 512, 512, 16) 2320
batch_normalization_16[0][0]
----------------------------------------------------------------------------------
------------------
```

```
batch_normalization_17 (BatchNo (None, 512, 512, 16) 64          conv2d_17[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_18 (Conv2D)              (None, 512, 512, 2)  34
batch_normalization_17[0][0]
================================================================================
==================
Total params: 2,164,322
Trainable params: 2,161,378
Non-trainable params: 2,944
--------------------------------------------------------------------------------
------------------
```

Se hace uso de esta nueva herramienta *plot_model* para mostrar un esquema de la red construida.

[6]:
```
plot_model(model)
```

[6]:


input_1: InputLayer
conv2d: Conv2D
batch_normalization: BatchNormalization
conv2d_1: Conv2D
batch_normalization_1: BatchNormalization
max_pooling2d: MaxPooling2D
conv2d_2: Conv2D
batch_normalization_2: BatchNormalization
conv2d_3: Conv2D
batch_normalization_3: BatchNormalization
max_pooling2d_1: MaxPooling2D
conv2d_4: Conv2D
batch_normalization_4: BatchNormalization
conv2d_5: Conv2D
batch_normalization_5: BatchNormalization
max_pooling2d_2: MaxPooling2D
conv2d_6: Conv2D
batch_normalization_6: BatchNormalization
conv2d_7: Conv2D
batch_normalization_7: BatchNormalization
max_pooling2d_3: MaxPooling2D
conv2d_8: Conv2D
batch_normalization_8: BatchNormalization
conv2d_9: Conv2D
batch_normalization_9: BatchNormalization
conv2d_transpose: Conv2DTranspose
concatenate: Concatenate
conv2d_10: Conv2D
batch_normalization_10: BatchNormalization
conv2d_11: Conv2D
batch_normalization_11: BatchNormalization
conv2d_transpose_1: Conv2DTranspose
concatenate_1: Concatenate
conv2d_12: Conv2D
batch_normalization_12: BatchNormalization
conv2d_13: Conv2D
batch_normalization_13: BatchNormalization
conv2d_transpose_2: Conv2DTranspose
concatenate_2: Concatenate
conv2d_14: Conv2D
batch_normalization_14: BatchNormalization
conv2d_15: Conv2D
batch_normalization_15: BatchNormalization
conv2d_transpose_3: Conv2DTranspose
concatenate_3: Concatenate
conv2d_16: Conv2D
batch_normalization_16: BatchNormalization
conv2d_17: Conv2D
batch_normalization_17: BatchNormalization
conv2d_18: Conv2D

Se crean los distintos generadores de datos, los cuales nos van a permitir realizar el *data augmentation* . En esta práctica, es aun más necesario recurrir a esta técnica ya que solo se disponen de 30 imágenes para entrenar el modelo, a diferencia de las 2000 de la anterior.

En este ejercicio, el *data augmentation* presenta una complicación. Las transformaciones deben realizarse tanto sobre la imagen original como sobre su máscara binaria. Para ello va a utilizarse el parámetro *seed*.

[14]:
```python
# we create two instances with the same arguments
data_gen_args = dict(featurewise_center=True,
                     featurewise_std_normalization=True,
                     rotation_range=90.,
                     width_shift_range=0.1,
                     height_shift_range=0.1,
                     zoom_range=0.2)
image_datagen = ImageDataGenerator(**data_gen_args)
mask_datagen = ImageDataGenerator(**data_gen_args)


seed = 1 # Para alterar tanto la imagen como su máscara

image_generator = image_datagen.flow_from_directory(
    '/content/data_unet/train/imgs',
    target_size=(512, 512),
    color_mode = 'grayscale', #imágenes en escala de grises -> un solo canal
    batch_size=10,
    class_mode=None,
    seed=seed)

mask_generator = mask_datagen.flow_from_directory(
    '/content/data_unet/train/labels',
    target_size=(512, 512),
    color_mode = 'grayscale',
    batch_size=10,
    class_mode=None,
    seed=seed)
```

```
Found 30 images belonging to 1 classes.
Found 30 images belonging to 1 classes.
```

Esta función será el generador del modelo. Básicamente a cada llamada del entrenamiento devuelve un *batch* de imágenes y sus respectivas máscaras. Cada pareja imagen-máscara habra sido sometida a las mismas transformaciones. En este caso se emplea *yield* en lugar de *return* ya que este primero no fuerza la salida de la función.

[10]:
```python
def train_generator ():
  train_gen = zip (image_generator, mask_generator)

  for (x,y) in train_gen:
    yield (x,y)
```

Se entrena el modelo. El número de *epochs* y *batches* por *epoch* es relativamente bajo. Esto ha sido así ya que con estos valores se han obtenido resultados relativamente buenos en un tiempo reducido.

[15]:
```python
history = model.fit_generator(generator = train_generator(),
                    steps_per_epoch = 15,
                    epochs = 50)
```

```
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/keras/engine/training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
/usr/local/lib/python3.6/dist-
packages/keras_preprocessing/image/image_data_generator.py:720: UserWarning:
This ImageDataGenerator specifies `featurewise_center`, but it hasn't been fit
on any training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-
packages/keras_preprocessing/image/image_data_generator.py:728: UserWarning:
This ImageDataGenerator specifies `featurewise_std_normalization`, but it hasn't
been fit on any training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
```

```
Epoch 1/50
15/15 [==============================] - 11s 532ms/step - loss: -78.9832 -
accuracy: 0.4406
Epoch 2/50
15/15 [==============================] - 8s 537ms/step - loss: -206.9899 -
accuracy: 0.5804
Epoch 3/50
15/15 [==============================] - 8s 540ms/step - loss: -265.7517 -
accuracy: 0.6509
Epoch 4/50
15/15 [==============================] - 8s 541ms/step - loss: -325.8661 -
accuracy: 0.6956
Epoch 5/50
15/15 [==============================] - 8s 541ms/step - loss: -373.0863 -
accuracy: 0.7166
Epoch 6/50
15/15 [==============================] - 8s 541ms/step - loss: -423.3761 -
accuracy: 0.7231
```

```
Epoch 7/50
15/15 [==============================] - 8s 536ms/step - loss: -475.8874 -
accuracy: 0.7191
Epoch 8/50
15/15 [==============================] - 8s 534ms/step - loss: -529.0097 -
accuracy: 0.7215
Epoch 9/50
15/15 [==============================] - 8s 534ms/step - loss: -583.1932 -
accuracy: 0.7182
Epoch 10/50
15/15 [==============================] - 8s 541ms/step - loss: -638.7064 -
accuracy: 0.7202
Epoch 11/50
15/15 [==============================] - 8s 542ms/step - loss: -702.6253 -
accuracy: 0.7222
Epoch 12/50
15/15 [==============================] - 8s 538ms/step - loss: -758.4969 -
accuracy: 0.7188
Epoch 13/50
15/15 [==============================] - 8s 538ms/step - loss: -821.8916 -
accuracy: 0.7210
Epoch 14/50
15/15 [==============================] - 8s 536ms/step - loss: -887.8904 -
accuracy: 0.7256
Epoch 15/50
15/15 [==============================] - 8s 542ms/step - loss: -953.2229 -
accuracy: 0.7286
Epoch 16/50
15/15 [==============================] - 8s 543ms/step - loss: -1021.8696 -
accuracy: 0.7291
Epoch 17/50
15/15 [==============================] - 8s 540ms/step - loss: -1093.1558 -
accuracy: 0.7312
Epoch 18/50
15/15 [==============================] - 8s 537ms/step - loss: -1163.4853 -
accuracy: 0.7312
Epoch 19/50
15/15 [==============================] - 8s 539ms/step - loss: -1239.6684 -
accuracy: 0.7354
Epoch 20/50
15/15 [==============================] - 8s 548ms/step - loss: -1320.9810 -
accuracy: 0.7356
Epoch 21/50
15/15 [==============================] - 8s 547ms/step - loss: -1396.2175 -
accuracy: 0.7341
Epoch 22/50
15/15 [==============================] - 8s 546ms/step - loss: -1476.9945 -
accuracy: 0.7389
```

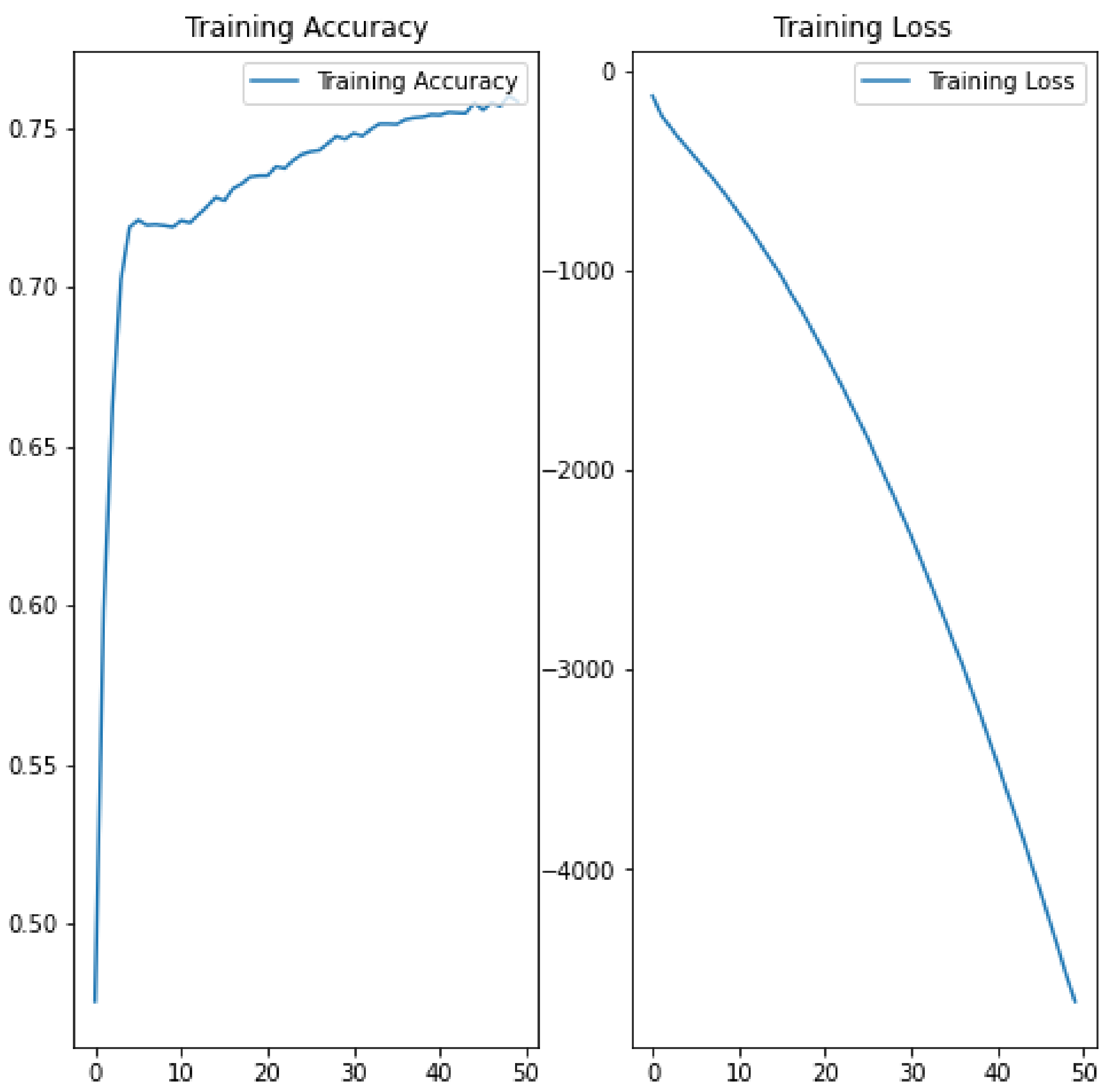
```
Epoch 23/50
15/15 [==============================] - 8s 543ms/step - loss: -1565.6238 -
accuracy: 0.7369
Epoch 24/50
15/15 [==============================] - 8s 542ms/step - loss: -1649.2079 -
accuracy: 0.7391
Epoch 25/50
15/15 [==============================] - 8s 539ms/step - loss: -1735.7644 -
accuracy: 0.7400
Epoch 26/50
15/15 [==============================] - 8s 538ms/step - loss: -1823.6068 -
accuracy: 0.7415
Epoch 27/50
15/15 [==============================] - 8s 538ms/step - loss: -1924.2551 -
accuracy: 0.7426
Epoch 28/50
15/15 [==============================] - 8s 540ms/step - loss: -2017.6353 -
accuracy: 0.7433
Epoch 29/50
15/15 [==============================] - 8s 541ms/step - loss: -2111.9837 -
accuracy: 0.7486
Epoch 30/50
15/15 [==============================] - 8s 543ms/step - loss: -2206.0631 -
accuracy: 0.7465
Epoch 31/50
15/15 [==============================] - 8s 547ms/step - loss: -2309.4192 -
accuracy: 0.7493
Epoch 32/50
15/15 [==============================] - 8s 536ms/step - loss: -2416.4174 -
accuracy: 0.7471
Epoch 33/50
15/15 [==============================] - 8s 538ms/step - loss: -2524.1440 -
accuracy: 0.7499
Epoch 34/50
15/15 [==============================] - 8s 540ms/step - loss: -2632.3120 -
accuracy: 0.7510
Epoch 35/50
15/15 [==============================] - 8s 541ms/step - loss: -2736.2550 -
accuracy: 0.7505
Epoch 36/50
15/15 [==============================] - 8s 543ms/step - loss: -2848.1102 -
accuracy: 0.7532
Epoch 37/50
15/15 [==============================] - 8s 540ms/step - loss: -2955.6590 -
accuracy: 0.7539
Epoch 38/50
15/15 [==============================] - 8s 540ms/step - loss: -3071.6465 -
accuracy: 0.7544
```

```
Epoch 39/50
15/15 [==============================] - 8s 540ms/step - loss: -3192.4058 -
accuracy: 0.7543
Epoch 40/50
15/15 [==============================] - 8s 548ms/step - loss: -3325.0229 -
accuracy: 0.7552
Epoch 41/50
15/15 [==============================] - 8s 539ms/step - loss: -3437.1041 -
accuracy: 0.7533
Epoch 42/50
15/15 [==============================] - 8s 541ms/step - loss: -3569.2638 -
accuracy: 0.7544
Epoch 43/50
15/15 [==============================] - 8s 543ms/step - loss: -3693.4068 -
accuracy: 0.7541
Epoch 44/50
15/15 [==============================] - 8s 540ms/step - loss: -3815.4340 -
accuracy: 0.7548
Epoch 45/50
15/15 [==============================] - 8s 555ms/step - loss: -3951.6358 -
accuracy: 0.7574
Epoch 46/50
15/15 [==============================] - 8s 556ms/step - loss: -4086.3543 -
accuracy: 0.7565
Epoch 47/50
15/15 [==============================] - 8s 553ms/step - loss: -4216.2557 -
accuracy: 0.7575
Epoch 48/50
15/15 [==============================] - 8s 551ms/step - loss: -4362.8468 -
accuracy: 0.7569
Epoch 49/50
15/15 [==============================] - 8s 556ms/step - loss: -4497.1670 -
accuracy: 0.7616
Epoch 50/50
15/15 [==============================] - 8s 558ms/step - loss: -4638.1631 -
accuracy: 0.7590
```

A continuación se dibujan unas gráficas que permiten observar el entrenamiento del modelo en términos de *accuracy* y error.

[16]:
```
acc = history.history['accuracy']

loss = history.history['loss']

epochs = 50
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
```

```
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.legend(loc='upper right')
plt.title('Training Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.legend(loc='upper right')
plt.title('Training Loss')
plt.show()
```



En cuanto al *accuracy* se observa como este sube bruscamente en las primeras *epochs* y después tiende a a permanecer casi constante en valores entre 0.75 y 0.8. Este es uno de los motivos por los cuales tampoco se han escogido realizar más *epochs* durante el entrenamiento, ya que la tendencia de la curva deja ver que no va a mejorar mucho el modelo en términos de *accuracy*. Además, podría existir el peligro de que si se sigue entrenando, se pueda caer en *overfitting*. La probabilidad de esto último aumenta al no haber incluido algunas medidas contra ello en el modelo, como por ejemplo *dropout*.

Se procede a ordenar los datos del conjunto de test de una forma válida para realizar las posteriores predicciones.

[18]:
```
!mkdir /content/data_unet/test/test_imgs
```

[19]:
```
!mv /content/data_unet/test/*.png  /content/data_unet/test/test_imgs/
```

Para realizar las predicciones, se crea un generador vacío, con todos los valores nulos, de tal forma que no se realice ningún tipo de transformación sobre las imágenes cargadas.

Al principio se optó por cargar las imágenes una a una empleando la función de matplotlib *imread* más un reajuste del tamaño de la imagen con tal de que este encajase en el modelo. Sin embargo este método daba problemas cuando se llamaba a *predict*. Al final se probó con el generador y funcionó correctamente.

Nótese que en este caso se ha tenido que recurrir al parámetro *shuffle* para que posteriormente se puedan mostrar correctamente los pares imagen-máscara. Este parámetro permite seleccionar *batches* de imágenes de forma no aleatoria.

[34]:
```
test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow_from_directory('/content/data_unet/test',
                                                  target_size=(512, 512),
                                                  color_mode = 'grayscale',
                                                  batch_size=30,
                                                  class_mode=None,
                                                  shuffle = False, #para␣
 ↪evitar que se ploteen imágenes distintas
                                                  seed = 1)
```

Found 30 images belonging to 1 classes.

Se realiza la predicción. En este caso se realiza un solo *step* ya que el *batch_size* seleccionado es igual al número de imágenes de test disponibles.

[35]:
```
output = model.predict(test_generator,
                       steps = 1)
```

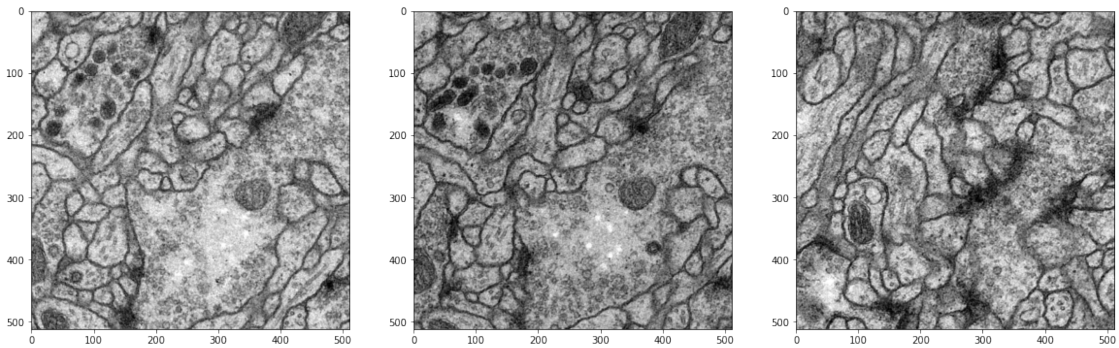
Así pues en *output* se tienen todas las máscaras.

[40]:
```
output.shape
```

[40]: (30, 512, 512, 2)

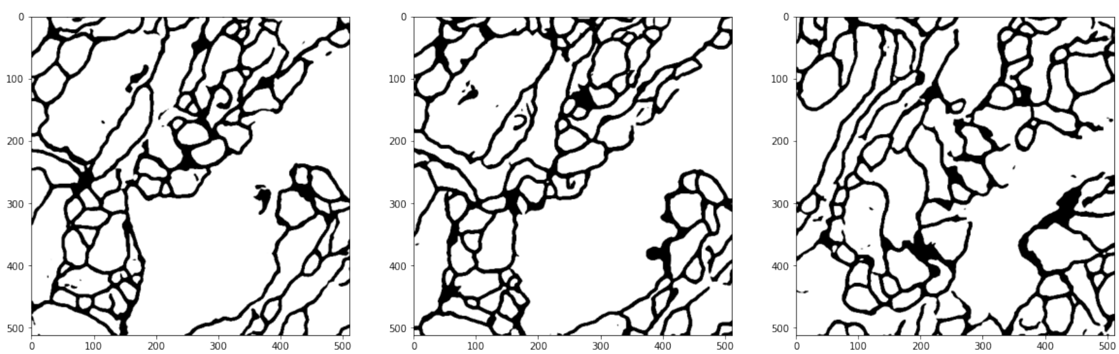
```python
[36]: orig_test_img = next(test_generator)
fig, ax = plt.subplots(1,3, figsize=(20, 10))
ax[0].imshow(orig_test_img[0].reshape((512,512)), cmap='gray')
ax[1].imshow(orig_test_img[1].reshape((512,512)), cmap='gray')
ax[2].imshow(orig_test_img[2].reshape((512,512)), cmap='gray')
```

```
[36]: <matplotlib.image.AxesImage at 0x7fa997a7b400>
```



```python
[37]: fig, ax = plt.subplots(1,3, figsize=(20, 10))
ax[0].imshow(output[0][:,:,0].reshape((512,512)), cmap='gray')
ax[1].imshow(output[1][:,:,0].reshape((512,512)), cmap='gray')
ax[2].imshow(output[2][:,:,0].reshape((512,512)), cmap='gray')
```

```
[37]: <matplotlib.image.AxesImage at 0x7fa995ffecf8>
```



Puede apreciarse que los resultados son buenos, aunque mejorables. Especialmente esto se aprecia en algunos bordes inconexos.

Por otra parte podría decirse que no existen indicios de *overfitting*. Así a ojo, podría decirse que el *accuracy* de test ronda también el 75-80 %, al igual que en el conjunto de entrenamiento.

Como posibles mejoras, existe la opción de entrenar la red con un mayor número de *epochs*. Por otro lado, podrías modificarse algunos parámetros internos, como el núemero de filtros empleados en cada una de las capas. Idealmente, lo mejor sería disponer de un mayor número de imágenes para entrenar.