**Universidad de Granada**

Máster Universitario Oficial en Ciencia de Datos e Ingeniería de Computadores

Biología Computacional con Big Data-Omics e Ingeniería Biomédica

# Guión III: Clasificación con imágenes biomédicas

*Profesores:*
Francisco Carrillo Perez
Daniel Castillo Secilla
Ignacio Rojas Ruiz

9 de abril de 2021

# Índice

# 1. Objetivos

Durante el desarrollo de este guión se pretenden alcanzar los siguientes objetivos principales:

- Familiarizarse con el formato SVS de las Whole-Slide-Images (WSI).
- Conocer y utilizar Python para la lectura y transformación de las imágenes.
- Uso de algoritmos de clasificación con las imágenes obtenidas.
- Familiarizarse con Google Colab para el uso de GPUs en el entrenamiento de redes convolucionales.

# 2. Requisitos

Es necesario disponer de las siguientes herramientas, librerías y datos para la correcta realización de este guión:

- Python 3.6.
- Descarga de las principales librerías para trabajar con datos.
- Disponer de los ficheros SVS siguiendo los pasos en el Guión I.

# 3. Introducción

El principal objetivo del presente guión es la familiarización del trabajo con imágenes de tejidos cancerígenos y sanos para la posterior clasificación de los mismos utilizando algoritmos de aprendizaje automático.

El uso de imágenes digitales para la detección o diagnóstico de patologías ha incrementado rápidamente en los últimos años. Desde hace varias décadas se llevan usando estas imágenes digitales para, una vez obtenidas y procesadas, crear modelos de clasificación. Un ejemplo serían las imágenes de resonancia magnética, que se han utilizado ampliamente en la literatura para la creación de modelos de clasificación de imágenes neurodegenerativas [1].

Un tipo de imagen utilizado ampliamente en los últimos años son las Whole-Slide-Images (WSI). La obtención de estas imágenes consta de dos pasos. El primero hace uso de un hardware especializado (en este caso un escáner) el cuál digitaliza una lámina de cristal. Para la obtención de la imagen no se realiza una única captura, ya que ello nos daría una resolución inferior de la que necesitaríamos. Lo interesante de esta metodología es que utiliza una técnica de unión de imágenes. Para obtener

imágenes de alta resolución lo que se realiza es unir tiras de imágenes, produciendo incluso solapamiento, de forma que se puedan posteriormente unir como si fuesen un mosaico. Una vez obtenidas las imágenes el segundo paso es la visualización y obtención de las imágenes con el uso de software.

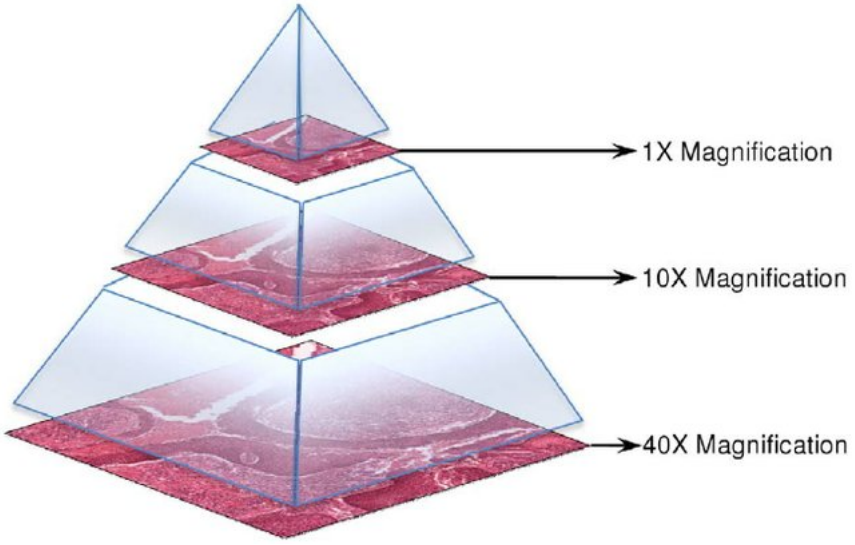


Figura 1: Ejemplo de pirámide de maximizaciones en una imágen WSI [3].

En el caso del cáncer, esta técnica se utiliza para visualizar cortes de tejidos (pueden ser de tejido tumoral o no). Para poder visualizar mejor las partes que nos interesan del tejido se realiza una técnica que se conoce como tinción hematoxilina-eosina. El método supone la aplicación de la tinción de hematoxilina, que por ser catiónica o básica, tiñe estructuras ácidas (basófilas) en tonos azul y púrpura,como por ejemplo los núcleos celulares; y el uso de eosina que tiñe componentes básicos (acidófilos) en tonos de color rosa, gracias a su naturaleza aniónica o ácida, como el citoplasma [2].

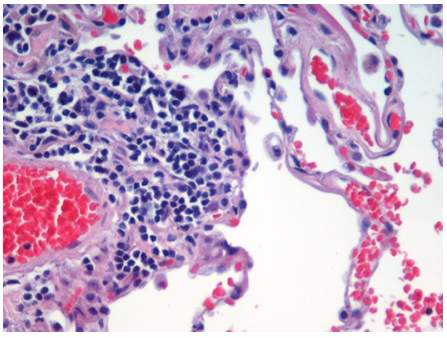

Figura 2: Ejemplo de la tinción hematoxilina-eosina usada sobre un tejido [4].

# 4. Preparación del entorno de trabajo de Python

**Paso 1**

De base Python viene instalado en sistemas GNU/Linux y MacOS. En el caso de utilizar Windows instalar el entorno Python descargándolo mediante su sitio web oficial.

La instalación de los paquetes que se van a utilizar durante la realización de esta práctica son los siguientes:

```
matplotlib==3.1.1
numpy==1.17.2
opencv-python==4.1.1.26
openslide-python==1.1.1
palettable==3.3.0
pandas==0.25.1
Pillow==6.2.0
psutil==5.6.3
pycparser==2.19
pylibmc==1.6.1
pyparsing==2.4.2
pyproj==2.4.0
python-dateutil==2.8.0
pytz==2019.3
pyvips==2.1.8
scikit-image==0.16.2
scikit-learn==0.22
scipy==1.3.3
six==1.12.0
sklearn==0.0
tqdm==4.36.1
```

Para facilitar la instalación de los paquetes, se proporciona un fichero *requirements.txt*. Para la instalación de los paquetes mediante este fichero se debe realizar el siguiente paso.

**Paso 2**

PIP ya viene de serie con las versiones de Python 3.4+ por tanto no sería necesario instalarlo. Si aun así se están utilizando versiones anteriores de Python 3:
Para la instalación de sistemas GNU/Linux la instalación de PIP se realiza con los siguientes comandos:

$ sudo apt update
$ sudo apt install python3-pip

Antes de instalar los requerimientos de Python debemos instalar la librería Openslide. En linux:

$ sudo apt-get install openslide-tools
$ sudo apt-get install python3-openslide

A continuación, instalaremos el paquete large-image con el siguiente comando:
$ pip install large-image[all] girder-large-image-annotation[tasks] –find-links https://girder.github.io/large_image_wheels
Una vez realizada la instalación, ejecutamos el siguiente comando dónde se encuentre el fichero requirements.txt:

$ pip3 install -r requirements.txt

# 5. Conversión de las WSI en formato SVS a otros formatos

Para poder trabajar con este tipo de imágenes es necesario realizar la conversión del formato en que se obtienen (SVS) a otros formatos con los que se pueda trabajar de manera más sencilla. Para ello vamos a hacer uso de la librería de Python **Openslide**, en concreto el paquete *large-image*[5] de Python. SVS es un formato privativo construido utilzando como base el formato TIFF. Con la librería Openslide seremos capaces de leer la imagen SVS eligiendo el tamaño de magnificación con el que deseamos obtener la imagen saliente (hemos de recordar que lo interesante de las imágenes WSI es que obtienen la digitalización del corte haciendo uso de la unión de las imágenes a distas escalas de maginificación).

Para ello comenzamos cargando las librerías necesarias y declarando los directorios a utilizar junto con las etiquetas de las clases:

```
1 """Script to convert SVS images to PNG and save them to a class folder.
```

```
    """
import large_image # para trabajar con las imágenes SVS
import cv2 # para el guardado de las imágenes en formato PNG
from pathlib import Path # para explorar los directorios
from tqdm import tqdm # para visualizar el progreso
import pandas as pd # para crear el CSV con la información

path_healthy = "../Data/SolidTissueNormal/"
path_tumor = "../Data/PrimaryTumor/"
label_healthy = "SolidTissueNormal"
label_tumor = "PrimaryTumor"
```

A continuación vamos a recorrer las imágenes que tenemos en los directorios guardadas. A la vez que realizamos esto, vamos a ir guardando en dos arrays toda la información para luego crear un CSV que nos pueda servir de consulta:

```
paths = []
labels = []

# Recorremos recursivamente los directorios donde se encuntran
# las imágenes
for filename in tqdm(Path('../').glob('**/*.svs')):
    f = str(filename) # nombre del fichero
    save_name = f.split('/')[-1].split('.')[0] # cogemos todo menos el
    .svs
    label = f.split('/')[-1].split('-')[3] # cuál es la clase de la
    imagen
    ts = large_image.getTileSource(f) # cargamos esa imagen SVS con el
    paquete

    # Ahora lo que hacemos es decir con qué magnificación queremos la
    imagen.
    # En este caso la estamos obteniendo con una magnificación de 2.5,
    sobre todo
    # por costes computacionales y le inifcamos que nos la devuelva
    como un
    # numpy ndarray.
    im_low_res, _ = ts.getRegion(
        scale=dict(magnification=2.5),
        format=large_image.tilesource.TILE_FORMAT_NUMPY
    )

    # Comprobamos a qué clase pertence la imagen siento 01A o 01Z
    Primary Tumor
    # y 11A Solid Tissue Normal. Guardamos la imagen en su respectivo
    directorio
    # y añadimos la información  a los arrays.
```

```
    if(label == '01A' or label == '01Z'):
        cv2.imwrite(path_tumor + save_name + '.png', im_low_res)
        paths.append(path_tumor + save_name + '.png')
        labels.append(label_tumor)
    elif(label == '11A'):
        cv2.imwrite(path_healthy + save_name + '.png', im_low_res)
        paths.append(path_healthy + save_name + '.png')
        labels.append(label_healthy)

# Creamos el DataFrame de Pandas con la información y guardamos el CSV.
data = pd.DataFrame()
data['Img_Paths'] = paths
data['Label'] = labels

data.to_csv('../Data/info.csv', sep=',', index=False)
```

## 5.1. Separación de dos o más tejidos dentro de una imagen

Es usual que en las imágenes que obtenemos de GDC se encuentre más de un corte de tejido de este paciente. Separar esta imagen en varias (una por corte de tejido) es de gran utilidad ya que, no solo aumentaríamos el número de imágenes que tenemos, si no que hacemos que el conjunto sea más homogéneo (teniendo un único corte en cada imagen).

Para ello vamos a hacer uso del histograma, para saber en que puntos podemos hacer los cortes, y una búsqueda binaria del punto final.

Primero importamos las librerías necesarias y declaramos algunos parámetros:

```
import numpy as np
from skimage import filters
import cv2
import matplotlib.pyplot as plt

eps = 0
eps_pc = 0.05
smooth_pc = 0.025
min_area = 0.15
filter_h_pc = 0
filter_w_pc =0
```

Creamos las funciones necesarias para poder calcular tanto valores del histograma, como funciones complementarias para reducción de ruido, poner valores del histograma a cero:

```
def reduce_noise_rec(img, num=3, sz=20):
    """Reducción del ruido de la imagen."""
    final = img
    for i in range(num-1):
        final = cv2.morphologyEx(final, cv2.MORPH_CLOSE, np.ones((sz,sz),np.uint8))
        final = cv2.morphologyEx(final, cv2.MORPH_OPEN, np.ones((sz-10,sz-10),np.uint8))
        sz += 10
    final = cv2.morphologyEx(final, cv2.MORPH_CLOSE, np.ones((sz,sz),np.uint8))
    return final

def put_zeros(hist, ind1, ind2):
    """Igualar a cero los valores del histograma entre dos índices."""
    hist[ind1:ind2] = 0
    while True:
        ind1 -= 1
        if ind1 < 0:
            break
        if hist[ind1] > eps:
            hist[ind1] = 0
        else:
            #break
            if ind1-1 > 0:
                if hist[ind1-1] < hist[ind1]:
                    hist[ind1] = 0
                else:
                    break
            else:
                break
    while True:
        if ind2 >= hist.shape[0]:
            break
        if hist[ind2] > eps:
            hist[ind2] = 0
        else:
            #break
            if ind2+1 < hist.shape[0]:
                if hist[ind2] > hist[ind2+1]:
                    hist[ind2] = 0
                else:
                    break
            else:
                break
        ind2 += 1

```

```
45      return ind1+1, ind2
46
47  def check_partition(hist, num, indices):
48      hist2 = np.copy(hist)
49      # Reduce peaks
50      # Hay que tener en cuenta que puede haber varios picos muy juntos
        de la misma subimagen, asi que hay que
51      # reducirlos todos juntos
52      # Si el siguiente pico ya esta reducido cambiar indices y reducir
        en 1 num
53
54      i = 1
55      while i < len(indices):
56          if hist2[indices[i]] == 0:
57              #indices[i-1] = indices[i+1]
58              del indices[i:i+2]
59              num -= 1
60              continue
61          ind1, ind2 = put_zeros(hist2, indices[i], indices[i+1])
62          indices[i] = ind1
63          indices[i+1] = ind2
64          i += 2
65
66      # Integrate again
67      area = np.sum(hist2)
68      # Check if > 15%
69      if area > min_area:
70          return False, num, indices
71      else:
72          return True, num, indices
73
74
75  def b_search(hist, bottom, top):
76      """Búsqueda binaria."""
77      mid = (bottom + top) / 2
78      hist2 = np.copy(hist)
79      hist2[hist2 > mid] = 1
80      hist2[hist2 <= mid] = 0
81      #print('top: ' + str(top))
82      #print('mid: ' + str(mid))
83
84      indices = [-1]
85      n_bottom = 1
86      n_top = 0
87      is_bottom = True
88
89      for i in range(hist2.shape[0]):
90          if is_bottom:
91              if hist2[i] == 1:
```

```
92                  n_top += 1
93                  is_bottom = False
94                  indices += [i]
95          else:
96              if hist2[i] == 0:
97                  n_bottom += 1
98                  is_bottom = True
99                  indices += [i]
100     if not is_bottom:
101         n_bottom += 1
102         indices += [hist2.shape[0]]
103 
104     ok, n_top, indices = check_partition(hist, n_top, indices)
105 
106     if ok:
107         return n_top, indices[1:]
108     else:
109         if n_top > 4:
110             bottom = mid
111             return b_search(hist, bottom, top)
112         else:
113             top = mid
114             return b_search(hist, bottom, top)
115 
116 
117 def get_num_images_hist(hist):
118     global eps
119 
120     hist = filters.gaussian(hist, sigma=smooth_pc*6000)
121     min_val = 0
122     max_val = np.amax(hist)
123     eps = eps_pc * max_val
124 
125     return b_search(hist, min_val, max_val)
126 
127 
128 def get_num_images(img):
129     # Luminance and binarization
130     L = 0.299*img[:,:,0] + 0.587*img[:,:,1] + 0.114*img[:,:,2]
131     threshold = filters.threshold_otsu(L)
132     binarized = (L < threshold).astype(np.uint8)
133     final = reduce_noise_rec(binarized)
134 
135     # Get histogram in y axis
136     hist = np.sum(final / np.sum(final), axis=0)
137     return get_num_images_hist(hist)
138 
139 
140 def get_images_inds(img, num, indices):
```

```
141     images = []
142 
143     walls = [0]
144     for i in range(num-1):
145         walls += [int((indices[2*i+1] + indices[2*i+2]) / 2)]
146     walls += [img.shape[1]]
147 
148     for i in range(num):
149         images += [img[:,walls[i]:walls[i+1],:]]
150 
151     return images
152 
153 
154 def get_images(img):
155     if img.shape[0] >= 2*img.shape[1]:
156         img = cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)
157     num, indices = get_num_images(img)
158     images = get_images_inds(img, num, indices)
159     return images
```

Y por último recorremos todas las imágenes dividiéndolas o no dependiendo de cuántos cortes se encuentren:

```
1  import numpy as np
2  from skimage import filters
3  import cv2
4  import matplotlib.pyplot as plt
5  import splitter
6  import os
7  import os.path
8 
9 
10 DataDir = '../Data'
11 OutDir = '../DataSplitted'
12 if not os.path.exists(OutDir):
13     os.mkdir(OutDir)
14 
15 
16 for dirpath, dirnames, filenames in os.walk(DataDir):
17     for filename in [f for f in filenames if f.endswith(".png")]:
18         print(filename)
19         folder = dirpath.split(os.sep)[-1]
20         name = filename.split('.')[0]
21         ImgPath = os.path.join(dirpath, filename)
22         OutPath = os.path.join(OutDir, folder)
23         if not os.path.exists(OutPath):
24             os.mkdir(OutPath)
```

```
25
26          print(ImgPath)
27          img = cv2.imread(ImgPath)
28          images = splitter.get_images(img)
29
30          for i in range(len(images)):
31              outname = name + "_" + str(i) + ".png"
32              OutFile = os.path.join(OutPath, outname)
33              cv2.imwrite(OutFile, images[i])
```

## 5.2. Creación de arrays para una validación cruzada en k segmentos

Si tuviésemos un conjunto de datos suficientemente grande, para la creación de un modelo solo nos haría falta hacer una partición de entrenamiento-validación-test y con eso tendríamos una buena representación de cómo funcionaría el modelo en otro conjunto de datos.

Sin embargo, en los conjuntos de datos de imágenes biomédicas no se suele tener una gran cantidad de imágenes debido a lo costoso que es obtenerlas. Es por ello que se suele utilizar la técnica de validación cruzada en k segmentos. Con esta técnica creamos k conjuntos de entrenamiento-validación. Para ello el conjunto de datos se separa en k conjuntos distintos. En la primera iteración se utiliza el primer segmento como validación y el resto como entrenamiento, en la segunda el segundo segmento como validación y el resto como entrenamiento, así sucesivamente hasta que hemos realizado k entrenamientos. Normalmente, antes de realizar la validación cruzada, habremos extraído un conjunto de test que solo será utilizado para validar el modelo finalmente elegido en la validación.

Una característica que tenemos en nuestro conjunto de datos es que podemos tener más de una imagen del mismo paciente. Si, por ejemplo, tenemos dos imágenes del paciente y una está en el conjunto de entrenamiento y otra en el conjunto de test, se podría dar una filtración de información entre el conjunto de entrenamiento y el conjunto de validación. Es por ello que un mismo paciente solo puede estar en uno de los conjuntos, ya sea este validación, entrenamiento o test.

Para no tener que crear estos conjuntos cada vez que ejecutemos el entrenamiento, lo que realizaremos será crear los arrays para cada iteración de antemano. De esta forma lo único que haremos será leer el array que corresponda a esa iteración de la validación cruzada en lugar de leer las imágenes cada vez.

Para facilitar la reutilización de parámetros, vamos a crear un fichero parameters.py para declarar los valores:

```
"""Global parameters for experiments."""

# WIDTH and HEIGHT
width = 299
height = 299
# PATHS AND VARIABLES
path_img_splitted = '../DataSplitted/'
path_plots = '../plots/'
splits = 5
path_input = '../inputs/inputs_labels_split'
classes = ['PrimaryTumor', 'SolidTissueNormal']
# CNN hyper-parameters
epochs = 15
lr = 0.001
```

Primero cargamos las librerías que van a ser necesarias:

```
"""Reading Breast Cancer images to create dataset."""
import cv2
import pandas as pd
import numpy as np # librería para trabajar con vectores y matrices
import random
import glob
import parameters
from tqdm import tqdm
random.seed(69)

HEIGHT = 299 # tamaño al que queremos reducir las imágenes
WIDTH = 299
```

Como hemos sacado de una misma imagen varias imágenes en el caso de que tuviésemos más de un tejido en la imagen, no todas ellas tienen las mismas dimensiones. Una restricción a la hora de luego entrenar redes neuronales convolucionales es que todas las imágenes deben tener las mismas dimensiones. Por ello, necesitamos declarar una función que reduzca las imágenes a unas mismas dimensiones pero sin crear una aberración en la imagen. Para ello, para conservar el ratio de la imagen, añadimos píxeles blancos a la imagen en el caso de que se reduzca demasiado el tamaño:

```
def image_resize(image, width=None, height=None, inter=cv2.INTER_AREA):
    """Resized and pad image to certain dimensions."""
    # initialize the dimensions of the image to be resized and
    # grab the image size
```

```
    dim = None
    (h, w) = image.shape[:2]

    # if both the width and height are None, then return the
    # original image
    if width is None and height is None:
        return image

    # check to see if the width is None
    if width is None:
        # calculate the ratio of the height and construct the
        # dimensions
        r = height / float(h)
        dim = (int(w * r), height)

    # otherwise, the height is None
    else:
        # calculate the ratio of the width and construct the
        # dimensions
        r = width / float(w)
        dim = (width, int(h * r))

    # resize the image
    resized = cv2.resize(image, dim, interpolation=inter)

    if(resized.shape[0] < HEIGHT):
        pad_ratio = int((HEIGHT - resized.shape[0]) / 2)

        padded = cv2.copyMakeBorder(resized, pad_ratio, pad_ratio, 0,
    0,
                                    cv2.BORDER_CONSTANT, value=[255,
    255, 255])

        if(padded.shape[0] != HEIGHT):
            padded = cv2.copyMakeBorder(resized, pad_ratio, pad_ratio +
     1, 0, 0,
                                        cv2.BORDER_CONSTANT, value
    =[255, 255, 255])
    else:
        padded = cv2.resize(resized, (width, HEIGHT), interpolation=
    inter)
    # return the resized image
    return padded
```

Y ya declaramos la función que nos genera los numpy arrays y los guarda como tal:

```
def kfold(paths, splits=5):
```

```
    """Creating a k Fold cross validation dataset."""
    data = pd.read_csv(paths)
    paths = list(data['Img_Paths'])
    random.shuffle(paths)

    patient_id = []
    for path in paths:
        filename = path.split('/')[3].split('.')[0]
        filename_split = filename.split('-')
        id_ = filename_split[0] + "-" + filename_split[1] + '-' + \
            filename_split[2]
        if(id_ not in patient_id):
            patient_id.append(id_)

    x_test = []
    y_test = []
    test_per = int(len(patient_id) * 0.8)
    path_test = patient_id[test_per:]
    patient_id_train = patient_id[:test_per]

    # Leemos los datos
    for path in tqdm(path_test):
        new_path = parameters.path_img_splitted + parameters.classes[0]
    + '/' + path
        for img in glob.glob(new_path + "*.png"):
            img_ = cv2.imread(img)
            img_ = image_resize(img_, width=parameters.width)
            if(img_.shape != (parameters.height, parameters.width, 3)):
                print(img_.shape)
                break
            x_test.append(img_)
            y_test.append(parameters.classes[0])

        new_path = parameters.path_img_splitted + parameters.classes[1]
    + '/' + path
        for img in glob.glob(new_path + "*.png"):
            img_ = cv2.imread(img)
            img_ = image_resize(img_, width=parameters.width)
            if(img_.shape != (parameters.height, parameters.width, 3)):
                print(img_.shape)
                break
            x_test.append(img_)
            y_test.append(parameters.classes[1])

    print("Number of samples in test: {}".format(len(x_test)))

    dict_split = {
            'x_test': np.asarray(x_test),
            'y_test': np.asarray(y_test)
```

```
49          }
50
51      np.save('../inputs/inputs_labels_test' + '-' + str(parameters.
    height) + "x" + str(parameters.width) + '.npy',
52              dict_split)
53
54      window = int(len(patient_id_train) / splits)
55
56      for i in tqdm(range(splits)):
57          start = window * i
58          end = window * (i + 1)
59          path_val = patient_id_train[start:end]
60          path_train = []
61          for j in range(splits):
62              # if it is the test split
63              start_j = window * j
64              end_j = window * (j + 1)
65              if(j != i):
66                  path_train.append(patient_id[start_j:end_j])
67
68          path_train = [item for sublist in path_train for item in
    sublist]
69          x_train = []
70          y_train = []
71          x_val = []
72          y_val = []
73          for path in tqdm(path_train):
74              new_path = parameters.path_img_splitted + parameters.
    classes[0] + '/' + path
75              for img in glob.glob(new_path + "*.png"):
76                  img_ = cv2.imread(img)
77                  img_ = image_resize(img_, width=parameters.width)
78                  if(img_.shape != (parameters.height, parameters.width,
    3)):
79                      print(img_.shape)
80                      break
81                  x_train.append(img_)
82                  y_train.append(parameters.classes[0])
83
84              new_path = parameters.path_img_splitted + parameters.
    classes[1] + '/' + path
85              for img in glob.glob(new_path + "*.png"):
86                  img_ = cv2.imread(img)
87                  img_ = image_resize(img_, width=parameters.width)
88                  if(img_.shape != (parameters.height, parameters.width,
    3)):
89                      print(img_.shape)
90                      break
91                  x_train.append(img_)
```

```
92                  y_train.append(parameters.classes[1])
93
94          print("Number of samples in train: {}".format(len(x_train)))
95
96          for path in tqdm(path_val):
97              new_path = parameters.path_img_splitted + parameters.
    classes[0] + '/' + path
98              for img in glob.glob(new_path + "*.png"):
99                  img_ = cv2.imread(img)
100                 img_ = image_resize(img_, width=parameters.width)
101                 if(img_.shape != (parameters.height, parameters.width,
    3)):
102                     print(img_.shape)
103                     break
104                 x_val.append(img_)
105                 y_val.append(parameters.classes[0])
106
107             new_path = parameters.path_img_splitted + parameters.
    classes[1] + '/' + path
108             for img in glob.glob(new_path + "*.png"):
109                 img_ = cv2.imread(img)
110                 img_ = image_resize(img_, width=parameters.width)
111                 if(img_.shape != (parameters.height, parameters.width,
    3)):
112                     print(img_.shape)
113                     break
114                 x_val.append(img_)
115                 y_val.append(parameters.classes[1])
116
117         print("Number of samples in val: {}".format(len(x_val)))
118
119         print("Total number of samples in the splits: {}".format(len(
    x_train) + len(x_val)))
120
121         dict_split = {
122             'x_train': np.asarray(x_train),
123             'y_train': np.asarray(y_train),
124             'x_val': np.asarray(x_val),
125             'y_val': np.asarray(y_val)
126         }
127
128         np.save('../inputs/inputs_labels_split' + str(i) + '-' + str(
    parameters.height) + "x" + str(parameters.width) + '.npy',
129                 dict_split)
```

# 6. Clasificación usando características de la función Wavelet como entrada a Support Vector Machines (SVM)

Una vez que ya hemos leído las imágenes y tenemos nuestros conjuntos preparados podemos entrenar un clasificador con ellas. Como una aproximación inicial vamos a utilizar un algoritmo clásico como son los SVM [11].

Las imágenes como tal se podrían utilizar como entrada para la clasificación, pero esto es extremadamente poco eficiente en el caso del SVM. Es por ello que vamos a extraer características de las imágenes usando la función de wavelet y posteriormente usarlas como entrada. La función de wavelet es un tipo especial de transformada matemática que representa una señal en términos de versiones trasladadas y dilatadas de una onda finita (denominada óndula madre). Cada una nos servirá como característica de entrada para la clasificación usando el SVM.

Primero importamos las librerías necesarias:

```
import numpy as np
import matplotlib.pyplot as plt
import pywt
import os

import itertools
from skimage import filters
from sklearn.cluster import KMeans
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
```

Declaramos las distintas funciones que necesitamos para extraer las características de una imagen:

```
def get_features_image_rgb(image, level=3, wavelets=['haar']):
    """ Función para extraer las características de una imagen."""
    features = []

    # wavelet
    for wavelet in wavelets:
        coeffs = pywt.wavedec2(image, wavelet, level=5, axes=(0,1))
        ca, cs = pywt.coeffs_to_array(coeffs, axes=(0,1))
        features += [ca[cs[0]].flatten()]
        for i in range(level):
```

```
            features += [ca[cs[i+1]['da']].flatten()]
            features += [ca[cs[i+1]['ad']].flatten()]
            features += [ca[cs[i+1]['dd']].flatten()]

    return np.concatenate(features)

def get_features_set(image_set):
    """Extraer las características de todo un conjunto de imágenes."""
    features = []
    for image in image_set:
        features += [get_features_image_rgb(image)]
    return np.stack(features, axis=0)


def get_labels_set(label_set):
    """Obtener el array de etiquetas."""
    global labels
    lbls = []
    for lbl in label_set:
        lbls += [labels.index(lbl)]
    return np.asarray(lbls)

# Función para sacar la matriz de confusión
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    #classes = classes[unique_labels(y_true, y_pred)]
    classes = [classes[i] for i in unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
```

```
60
61      fig, ax = plt.subplots()
62      im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
63      #plt.show()
64      #im = ax.matshow(cm, interpolation='nearest', cmap=cmap)
65      ax.figure.colorbar(im, ax=ax)
66      # We want to show all ticks...
67      ax.set(xticks=np.arange(cm.shape[1]),
68             yticks=np.arange(cm.shape[0]),
69             #yticks=np.array([-0.5, 1.5]),
70             #yticks=np.arange(cm.shape[0])/cm.shape[0] + 1.0/(2*cm.shape
    [0]),
71             #xlim=(0, cm.shape[1]+0.5),
72             ylim=(cm.shape[0]-0.5, -0.5),
73             # ... and label them with the respective list entries
74             xticklabels=classes, yticklabels=classes,
75             title=title,
76             ylabel='True label',
77             xlabel='Predicted label')
78
79      # Rotate the tick labels and set their alignment.
80      plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
81               rotation_mode="anchor")
82
83      # Loop over data dimensions and create text annotations.
84      fmt = '.2f' if normalize else 'd'
85      thresh = cm.max() / 2.
86      for i in range(cm.shape[0]):
87          for j in range(cm.shape[1]):
88              ax.text(j, i, format(cm[i, j], fmt),
89                      ha="center", va="center",
90                      color="white" if cm[i, j] > thresh else "black")
91      fig.tight_layout()
92      return ax
```

Por último, declaramos la función para el entrenamiento del SVM:

```
1  def train_svm(path='../inputs', splits=5, lbls=['PrimaryTumor', '
       SolidTissueNormal'], show_plot=False):
2      global labels
3      labels = lbls
4
5      # for dic in npy
6      for i in range(splits):
7          image_set = np.load(os.path.join(path, (part1 + str(i) + part2)
       ), allow_pickle=True)
8
```

```
        # create classifier
        clf = svm.SVC(C=1.0, kernel='linear', gamma='auto')

        # train
        print('training...')
        clf.fit(get_features_set(image_set['x_train']), get_labels_set(
    image_set['y_train']))

        # predict
        print('predicting...')
        y_pred = clf.predict(get_features_set(image_set['x_val']))
        y_val = get_labels_set(image_set['y_val'])

        correct = y_pred == y_val
        correct = correct.astype(np.int32)
        acc = np.sum(correct) / correct.shape[0]
        print(acc)

        plot_confusion_matrix(y_val, y_pred, classes=labels, title='
    Confusion Matrix')
        if show_plot:
            plt.show()
```

# 7. Clasificación usando Redes Neuronales Convolucionales

Las redes neuronales convolucionales son un tipo específico de red neuronal artificial donde se aplica la operación de convolución. Hay una pequeña discusión sobre la autoría de las redes convolucionales pero uno de los primeros trabajos donde fueron utilizadas fue en [6], donde se utilizaban para el reconocimiento de caracteres escritos.

Lo que hace que las redes neuronales convolucionales sean tan convenientes con respecto a los algoritmos clásicos de visión por computador es que realizan una extracción automática de características de la imagen. Anteriormente era necesario realizar una extracción manual de estas características (bordes, puntos de interés, etc.) para luego ser usadas como entrada para el clasificador. Sin embargo, gracias a la operación de convolución, la red aprende filtros que son capaces de detectar estas características. Esto se consigue ajustando los pesos de los filtros (que no es más que una matriz de valores) con respecto al error que se obtiene en la predicción de los datos de entrenamiento gracias al algoritmo de backpropagation que se utiliza en las redes neuronales artificiales. En el caso de querer ahondar más en el funcionamiento de las redes neuronales hay muchos recursos que son muy útiles [7], [8].

Entrenar una red desde cero es poco común, ya que normalmente no se tiene el

número de datos suficientes como para que la red sea capaz de aprender las características de forma eficiente. Es por ello que se suele utilizar una técnica conocida como Transfer Learning. En ella lo que hacemos es utilizar arquitecturas ya entrenadas en otro conjunto de datos y las utilizamos como extractores de características. Para ello, sustituimos su capa/capas finales por unas no entrenadas y solo entrenamos esas, haciendo uso de las características que han sido aprendidas por la red en el otro conjunto de datos.

Para este guión vamos a hacer uso de la librería de Deep Learning para Python, Keras [9]. Keras es una API de alto nivel sobre Tensorflow. Esto lo que nos permite es hacer más rápido el pipeline a la hora de entrenar la red, perdiendo un poco el control de los aspectos de más bajo nivel del entrenamiento.

Uno de los problemas que tienen las redes convolucionales es la cantidad de computación requerida para su entrenamiento. Es por ello que se suelen entrenar haciendo uso de las GPUs, debido al paralelismo de datos que estas proporcionan. En 2017 Google publicó una herramienta interna que se basaba en los Jupyter Notebooks, Google Colab [10]. Los Jupyter Notebooks son una herramienta de scripting interactiva que permite la ejecución por celdas y la visualización de resultados en tiempo real, tal como si fuese un cuaderno. Lo interesante de Google Colab es que nos permite usar las GPUs de Google a coste cero. Se ponen restricciones de espacio y memoria, pero son suficientes para nuestro proyecto.

## 7.1. Ejecutar nuestro scripts en una GPU en Google Colab

El notebook de ejemplo para el entrenamiento de la red neuronal con los datos que ya hemos preparado se encuentra en el siguiente **enlace**.

Una vez creado vuestro notebook debemos seleccionar que queremos que se ejecute en la GPU. Para ello realizamos los siguientes pasos:

1. En la barra de herramientas seleccionamos **Entorno de ejecución**.
2. Seleccionamos **Cambiar tipo de entorno de ejecución**.
3. En **Acelerador por hardware** seleccionamos GPU y le damos a guardar.

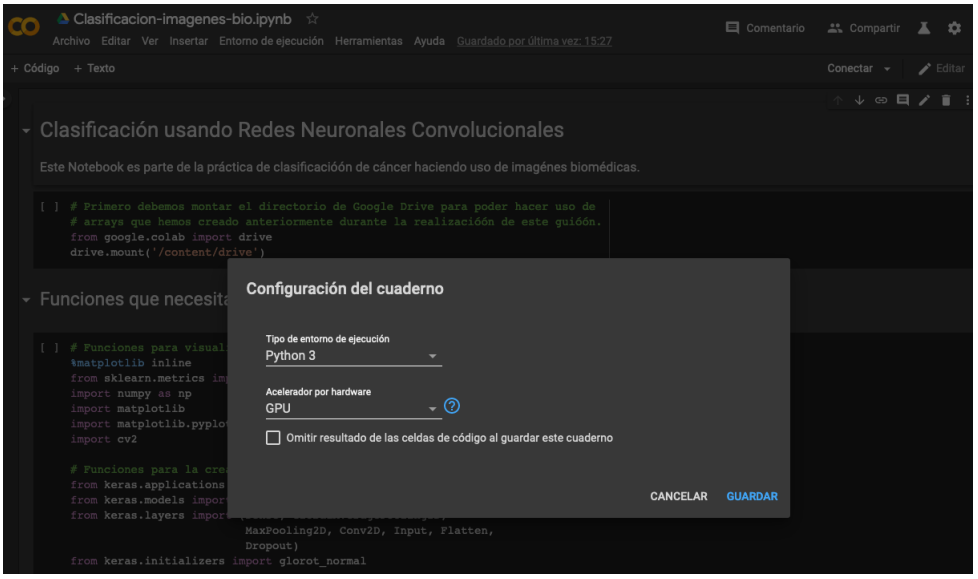


Figura 3: Como deben quedar las opciones para que se ejecute el código en GPU en Google Colab.

# Referencias

[1] Herrera, L. J., Rojas, I., Pomares, H., Guillén, A., Valenzuela, O., Baños, O. (2013, September). Classification of MRI images for Alzheimer's disease detection. In 2013 International Conference on Social Computing (pp. 846-851). IEEE.

[2] Jocelyn H. Bruce-Gregorios, M.D. Técnicas Histopatológicas. JMC Press Inc., Quezon City, Filipinas. 1974.

[3] Wang, Y., Williamson, K. E., Kelly, P. J., James, J. A., Hamilton, P. W. (2012). SurfaceSlide: a multitouch digital pathology platform. PloS one, 7(1), e30783.

[4] Rogers, A. (2004). T cells cause lung damage in emphysema. PLoS Med, 1(1), e25.

[5] large-image, Kitware, Inc. `https://pypi.org/project/large-image/`

[6] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. (1998).

[7] Ian Goodfellow, Yoshua Bengio & Aaron Courville Deep Learning. MIT Press Book. `http://www.deeplearningbook.org/`

[8] deeplearning.ai Coursera Programa Especializado Aprendizaje Profundo `https://es.coursera.org/specializations/deep-learning`

[9] Chollet, François and others. Keras, 2015 `https://keras.io`

[10] Google. Google Colab `https://colab.research.google.com`

[11] Boser, B. E., Guyon, I. M., Vapnik, V. N. (1992, July). A training algorithm for optimal margin classifiers. In Proceedings of the fifth annual workshop on Computational learning theory (pp. 144-152). ACM.