Anexo A

Ficheros creados para la realización de este proyecto

A.1 Ficheros del preprocesado de las bases de datos

En este primera parte del anexo se incluyen los ficheros empleados para el preprocesado de las bases de datos utilizadas durante la realización de este trabajo.

El código del primero de los ficheros que se utilizó se puede ver en el código A.1, con dicho fichero se extrajo de la base de datos la lista de los nombres de los sujetos que la componen (*faceid.list*), el nombre de las imágenes que hay (*col*) y el nombre de las imágenes que hay ordenadas por orden alfabético en función de a que sujeto corresponden.

Listing A.1: Tratando la base de datos 1

```
#!/bin/bash

cd ../lfw-deepfunneled/
ls -1 > ../example_crop_opencvLFW/faceid.list
find . | grep "\.jpg" > ../example_crop_opencvLFW/col
paste ../example_crop_opencvLFW/col | sort > ../
```

```
    example_crop_opencvLFW/colFinal
```

En el código A.2 se puede observar como se realizó el proceso de detección de los rostros en las imágenes y recorte de dichas imágenes en un tamaño de 128x128.

Listing A.2: Tratando la base de datos 2

```
import cv #Opencv
import Image #Image from PIL
import glob
import os,sys
import numpy as np
import argparse

parser=argparse.ArgumentParser()
parser.add_argument("-l","--lista", help="Lista parte imagenes")
args=parser.parse_args()

def cargar_datos(args):
        lista=args.lista
        return lista

lista=cargar_datos(args)

def DetectFace(image, faceCascade, returnImage=False):
    # This function takes a grey scale cv image and finds
    # the patterns defined in the haarcascade function

    #variables
    min_size = (20,20)
    haar_scale = 1.1
    min_neighbors = 3
    haar_flags = 0

    # Equalize the histogram
    cv.EqualizeHist(image, image)
```

```
    # Detect the faces
    faces = cv.HaarDetectObjects(
            image, faceCascade, cv.CreateMemStorage(0),
            haar_scale, min_neighbors, haar_flags, min_size
        )

    # If faces are found
    if faces and returnImage:
        for ((x, y, w, h), n) in faces:
            # Convert bounding box to two CvPoints
            pt1 = (int(x), int(y))
            pt2 = (int(x + w), int(y + h))
            cv.Rectangle(image, pt1, pt2, cv.RGB(255, 0, 0), 5, 8, 0)

    if returnImage:
        return image
    else:
        return faces

def pil2cvGrey(pil_im):
    # Convert a PIL image to a greyscale cv image
    pil_im = pil_im.convert('L')
    cv_im = cv.CreateImageHeader(pil_im.size, cv.IPL_DEPTH_8U, 1)
    cv.SetData(cv_im, pil_im.tostring(), pil_im.size[0]  )
    return cv_im

def cv2pil(cv_im):
    # Convert the cv image to a PIL image
    return Image.fromstring("L", cv.GetSize(cv_im), cv_im.tostring())

def imgCrop(image, cropBox, boxScale=1):
    # Crop a PIL image with the provided box [x(left), y(upper), w(
        width), h(height)]

    # Calculate scale factors
    xDelta=max(cropBox[2]*(boxScale-1),0)
    yDelta=max(cropBox[3]*(boxScale-1),0)
```

```
    # Convert cv box to PIL box [left, upper, right, lower]
    PIL_box=[cropBox[0]-xDelta, cropBox[1]-yDelta, cropBox[0]+cropBox
        [2]+xDelta, cropBox[1]+cropBox[3]+yDelta]

    return image.crop(PIL_box)

def faceCrop(imagePattern,boxScale=1):
    faceCascade = cv.Load('/extra/scratch03/vmingote/keras/dataBase/
        example_crop_opencvLFW/haarcascade_frontalface_alt.xml')

    imgList=glob.glob(imagePattern)
    if len(imgList)<=0:
        print 'No Images Found'
        return

    for img in imgList:
        pil_im=Image.open(img)
        cv_im=pil2cvGrey(pil_im)
        faces=DetectFace(cv_im,faceCascade)
        if faces:
            n=1
            for face in faces:
                croppedImage=imgCrop(pil_im, face[0],boxScale=
                    boxScale)
                image=croppedImage.resize((128,128)) #Reescalo las
                    imagenes al tamano que deseo
                fname,ext=os.path.splitext(img)
                image.save(fname+'_crop'+str(n)+ext)
                n+=1
        else:
            print 'No faces found:', img

#Funcion que lee el archivo con los modelos.
def leerarchivo(nArchivo):
    archivo = open (nArchivo,'r')
    Ltotal = []
```

```
    for linea in archivo.readlines(): # lee todas las lineas del
        archivo, conserva saltos de linea
        partes_linea = linea.split()
        Ltotal.append(' '.join(partes_linea))
    return Ltotal

os.chdir('/extra/scratch03/vmingote/keras/dataBase/
    example_crop_opencvLFW/')
print("Directorio:", os.getcwd())
col2Final=leerarchivo(lista)

os.chdir('data/')
# Crop all jpegs in a folder. Note: the code uses glob which follows
    unix shell rules.
# Use the boxScale to scale the cropping area. 1=opencv box, 2=2x the
    width and height
for i in range (len(col2Final)):
        archivo=col2Final[i]
        print(archivo)
        print("Directorio:", os.getcwd())
        faceCrop(archivo,boxScale=1)
        #os.remove(archivo)
```

Una vez recortadas todas las imágenes y colocadas junto a las originales se tuvo que obtener el nombre de dichas imágenes recortadas para luego poderlas pasar una carpeta con la misma estructura que donde se encontraban las originales pero en la que se tuviera solo las recortadas. Este proceso se realizó haciendo uso del código que se puede ver en A.3, en el cual se obtuvieron las rutas donde se encuentran todas las imágenes recortadas (*ficherosCrop.txt*, los nombres de los sujetos a los que corresponde cada imagen recortada (*identidades*), los nombres de las imágenes recortadas (*fotos*) y la lista de las rutas donde se debía tener cada una de las imágenes recortadas (*lfw.trn*).

Listing A.3: Tratando la base de datos 3

```
#!/bin/bash

find . -name "*_crop1.jpg" -type f | sort > ficherosCrop.txt
file=$1
size_list=`wc -l ficherosCrop.txt | awk '{ print $1}'`
cat ficherosCrop.txt | sed "s/\// /g" | awk '{print $3}' >
   identidades
cat ficherosCrop.txt | sed "s/\// /g" | awk '{print $4}' > fotos
paste -d '/' identidades fotos | sort > lfw.trn
```

A.2 Ficheros utilizados para crear los sistemas

En este sección se encuentran los ficheros creados para implementar los diversos modelos que se utilizaron para realizar los experimentos.

A.2.1 Iniciar modelos

En primer lugar se incluye el fichero usado para crear el primer sistema de los implementados en este trabajo. Con el código A.4 se creó el modelo que se componía de 2 capas convolucionales y 4 capas densas.

Listing A.4: Creando el modelo 2C+4D

```
from __future__ import print_function
import os, sys, stat
import numpy as np
import shutil

np.random.seed(1337)  # for reproducibility

from keras.datasets import funcionesCASIAUnificado
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
```

```
from keras.optimizers import SGD
from keras.utils import np_utils
from keras.callbacks import ModelCheckpoint
from keras.models import model_from_json

batch_size = 32
nb_classes = 10576
nb_epoch = 100

# dimensiones de entrada de las imagenes
img_rows, img_cols = 128, 128
img_channels = 1

def create_model():

        model = Sequential()
        model.add(Convolution2D(32, 11, 11, border_mode='same',
            dim_ordering='tf',
                                input_shape=( img_rows, img_cols,
                                    img_channels)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(3, 3),strides=(2,2),
            dim_ordering='tf'))
        model.add(Convolution2D(16, 9, 9,dim_ordering='tf'))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2),
            dim_ordering='tf'))
        model.add(Convolution2D(16, 9, 9,dim_ordering='tf'))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2),
            dim_ordering='tf'))
        model.add(Convolution2D(16, 7, 7,dim_ordering='tf'))
        model.add(Activation('relu'))
        model.add(Flatten())
        model.add(Dense(1024))
        model.add(Activation('relu'))
        model.add(Dense(1024))
```

```
        model.add(Activation('relu'))
        model.add(Dense(512))
        model.add(Activation('relu'))
        model.add(Dropout(0.5))
        model.add(Dense(nb_classes))
        model.add(Activation('softmax'))
        return model

model=create_model()
model.compile(loss='categorical_crossentropy', optimizer='sgd',
    metrics=["accuracy"])

nuevaruta = r'modelBDCASIA'
os.chdir('/extra/scratch03/vmingote/keras/modelos/')
print("Directorio:", os.getcwd())

if not os.path.exists(nuevaruta):
        os.mkdir(nuevaruta)
        os.chdir(nuevaruta)

        json_string=model.to_json()
        open('my_model_architecture.json', 'w').write(json_string)

        print ("Guardado el modelo")

        archivo = open ('archivo.txt','a')
        os.chmod('archivo.txt',0744)

        model.save_weights('model.h5')

else:
        print ("Ya esta creado el modelo inicial.")
```

A.2.2 Entrenar modelos

A continuación se incluyen los ficheros utilizados para llevar a cabo el entrenamiento tanto al tener que utilizar la función generadora como al no tener que usarla.

Listing A.5: Entrenamiento del sistema basado en una red convolucional de una sola rama

```
from __future__ import print_function
from __future__ import absolute_import
import os,sys,stat
import argparse
import numpy as np
import h5py
import shutil

np.random.seed(1337)  # for reproducibility

from keras.datasets import funcionesLFWUnificado
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.optimizers import SGD
from keras.utils import np_utils,visualize_util
from keras.callbacks import ModelCheckpoint, Callback
from keras.models import model_from_json
from matplotlib import pyplot as plt

parser=argparse.ArgumentParser()
parser.add_argument("-m","--model", help="Modelo a entrenar")
parser.add_argument("-b","--batch_size",help="Tamano batch")
parser.add_argument("-e","--nb_epoch",help="Numero de epocas")
parser.add_argument("-c","--nb_classes",help="Numero de clases")
args=parser.parse_args()
```

```
#Cambiar a la carpeta pasada por parametro
if args.model:
        nuevaruta=args.model
        os.chdir('/extra/scratch03/vmingote/keras/modelos/')
        os.chdir(nuevaruta)
        print ("Cambiado de directorio")
        print("Directorio:", os.getcwd())

def cargar_datos(args):
        batch_size=args.batch_size
        nb_epoch=args.nb_epoch
        nb_classes=args.nb_classes
        return int(batch_size),int(nb_epoch),int(nb_classes)

[batch_size,nb_epoch,nb_classes]=cargar_datos(args)

color=False
x_train,x_test,y_train,y_test,matrizId=funcionesLFWUnificado.
    load_dataOpenCV(nuevaruta,color)

X_train=x_train.reshape(x_train.shape[0],128,128,1)
X_test=x_test.reshape(x_test.shape[0],128,128,1)

Y_train = np_utils.to_categorical(y_train,nb_classes)
Y_test = np_utils.to_categorical(y_test,nb_classes)

#Cargo el modelo de la red creado para cada caso.
model = model_from_json(open('my_model_architecture.json').read())
model.compile(loss='categorical_crossentropy', optimizer='sgd',
    metrics=["accuracy"])

#Leo el fichero donde esta la lista de modelos ya creados y los paso
    a una lista de python.
Lista=funcionesLFWUnificado.leerarchivo('archivo.txt')
lon=len(Lista)

#Si es la primera vez que se lanza carga los creados con el fichero
```

```
    de iniciar.
if (lon==0):
    model.load_weights('model.h5')
    epoch=0
    print("Pesos iniciales cargados")

#Si ya existe algun modelo se carga el ultimo que esta en la lista.
else:
    ruta=Lista[lon-1]
    print ("Pesos que se van a cargar: ",ruta)
    model.load_weights(ruta)
    print ("Pesos cargados")
    epoch=lon

#Funcion para ir guardando los modelos en cada epoch.
save_model=ModelCheckpoint("model.{epoch:02d}.h5",modelos=Lista,
    verbose=1,save_best_only=False)

#Voy entrenando el modelo.
model.fit(X_train, Y_train, batch_size=batch_size,
              nb_epoch=nb_epoch, verbose=2, shuffle=True,callbacks=[
                  save_model],epoch=epoch)

visualize_util.plot(model,to_file='model.png')
```

Listing A.6: Entrenamiento del sistema basado en una red convolucional de una sola rama

```
from __future__ import print_function
from __future__ import absolute_import
import os,sys,stat
import argparse
import numpy as np
import h5py
import shutil
import scipy.io
```

```
np.random.seed(1337)  # for reproducibility

from keras.datasets import funcionesCASIAUnificado
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers.local import LocallyConnected2D
from keras.optimizers import SGD
from keras.utils import np_utils, visualize_util
from keras.callbacks import ModelCheckpoint, Callback
from keras.models import model_from_json
from matplotlib import pyplot as plt
from sklearn.utils import shuffle

parser=argparse.ArgumentParser()
parser.add_argument("-m","--model", help="Modelo a entrenar")
parser.add_argument("-b","--batch_size",help="Tamano batch")
parser.add_argument("-e","--nb_epoch",help="Numero de epocas")
parser.add_argument("-c","--nb_classes",help="Numero de clases")
parser.add_argument("-l","--lista", help="Lista parte imagenes")
args=parser.parse_args()

#Cambiar a la carpeta pasada por parametro
if args.model:
        nuevaruta=args.model
        os.chdir('/extra/scratch03/vmingote/keras/modelos/')
        os.chdir(nuevaruta)
        print ("Cambiado de directorio")
        print("Directorio:", os.getcwd())

def cargar_datos(args):
        batch_size=args.batch_size
        nb_epoch=args.nb_epoch
        nb_classes=args.nb_classes
        lista=args.lista
        return int(batch_size),int(nb_epoch),int(nb_classes),lista
```

```
[batch_size,nb_epoch,nb_classes,lista]=cargar_datos(args)

#Leo los ficheros con la informacion de los ficheros .mat que he
    creado anteriormente y que luego voy a tener que cargar.
listaDatos=funcionesCASIAUnificado.leerarchivo('/extra/scratch03/
    vmingote/keras/tfmv2/ficherosBDCASIA/ficherosCASIA-mia/listaDatos2
    .txt')
listaEtiquetas=funcionesCASIAUnificado.leerarchivo('/extra/scratch03/
    vmingote/keras/tfmv2/ficherosBDCASIA/ficherosCASIA-mia/
    listaEtiquetas2.txt')
listaLongitudes=funcionesCASIAUnificado.leerarchivo('/extra/scratch03
    /vmingote/keras/tfmv2/ficherosBDCASIA/ficherosCASIA-mia/lonDatos2.
    txt')

samples_per_epoch=0
#Lectura de los ficheros para calcular el numero total de samples per
     epoch
for i in range (len(listaLongitudes)):
     lon=listaLongitudes[i]
     samples_per_epoch=samples_per_epoch+int(lon)

print("Directorio:", os.getcwd())

#Generador para la carga del total de ficheros en cada iteracion
def myGenerator():
    while 1:
        ind=np.random.permutation(len(listaDatos))
        inds=[]
        trozos=4 #numero de trozos que cojo para unir
        for l in range(0,len(ind),trozos):
                if l+(trozos-1)<len(ind):
                        inds+=[ind[l:l+trozos]]

                else:
                        if l==(len(ind)-1):
                                inds+=[[ind[l]]]
```

```
                    else:
                            lon=len(ind)-1
                            inds+=[[ind[1:lon]]]

    for i in range(len(inds)):
            indi=inds[i]
            X_train=[]
            y_train=[]
            if (len(indi))<trozos:
                    print("BREAK")
                    break
            for k in range (trozos):
                    if k==0:
                            arch=indi[k]

                            fichero=listaDatos[arch]
                            Xdata={}
                            scipy.io.loadmat(fichero,Xdata)
                            X_train=Xdata.get("X_train")
                            X_train = X_train.reshape(X_train.
                                shape[0], 128, 128,1)

                            fichero2=listaEtiquetas[arch]
                            Ydata={}
                            scipy.io.loadmat(fichero2,Ydata)
                            y_train=Ydata.get("y_train")
                            y_train=np.transpose(y_train)

                    else:
                            arch=indi[k]

                            fichero=listaDatos[arch]
                            Xdata={}
                            scipy.io.loadmat(fichero,Xdata)
                            X_train1=Xdata.get("X_train")
                            X_train1 = X_train1.reshape(X_train1.
                                shape[0], 128, 128,1)
```

```
                        X_train=np.vstack((X_train,X_train1))

                        fichero2=listaEtiquetas[arch]
                        Ydata={}
                        scipy.io.loadmat(fichero2,Ydata)
                        y_train1=Ydata.get("y_train")
                        y_train1=np.transpose(y_train1)
                        y_train=np.vstack((y_train,y_train1))

            Y_train = np_utils.to_categorical(y_train, nb_classes
                )

            X_train, Y_train,apariciones = shuffle(X_train,
                Y_train, random_state=0)
            nbatches=X_train.shape[0]/batch_size
            print("Nbatches:",nbatches)

            for j in range(nbatches):
                    longitud=X_train.shape[0]

                    if (((j+1)*batch_size>(int(longitud)-
                        batch_size)) and ((j+1)*batch_size!=int(
                        longitud))):
                            Xtr = X_train[(j+1)*batch_size:int(
                                longitud)]
                            ytr = Y_train[(j+1)*batch_size:int(
                                longitud)]
                            yield [Xtr,ytr]

                    else:
                            Xtr = X_train[j*batch_size:(j+1)*
                                batch_size]
                            ytr = Y_train[j*batch_size:(j+1)*
                                batch_size]
                            yield [Xtr,ytr]
```

```
#Cargo el modelo de la red creado para cada caso.
model = model_from_json(open('my_model_architecture.json').read())
model.compile(loss='categorical_crossentropy', optimizer='sgd',
    metrics=["accuracy"])
model.summary()

#Leo el fichero donde esta la lista de modelos ya creados y los paso
    a una lista de python.
Lista=funcionesCASIAUnificado.leerarchivo('archivo.txt')
lon=len(Lista)

#Si es la primera vez que se lanza carga los creados con el fichero
    de iniciar.
if (lon==0):
    model.load_weights('model.h5')
    epoch=0
    print("Pesos iniciales cargados")

#Si ya existe algun modelo se carga el ultimo que esta en la lista.
else:
    ruta=Lista[lon-1]
    print ("Pesos que se van a cargar: ",ruta)
    model.load_weights(ruta)
    print ("Pesos cargados")
    epoch=lon

#Funcion para ir guardando los modelos en cada epoch.
save_model=ModelCheckpoint("model.{epoch:02d}.h5",modelos=Lista,
    verbose=1,save_best_only=False)

# fit the model on the batches generated by datagen.flow()
model.fit_generator(myGenerator(),
                    samples_per_epoch=samples_per_epoch,
                    nb_epoch=nb_epoch,verbose=2,
                    callbacks=[save_model],epoch=epoch)

visualize_util.plot(model,to_file='model.png')
```

A.2.3 Extracción de características

En el código A.7 se puede observar como se extraen los vectores de características para todas las imágenes de la base de datos LFW que se utilizaron para evaluar el reconocimiento posterior y como se guardaba cada vector de características en diferentes ficheros de formato .txt.

Listing A.7: Extracción de características

```
from __future__ import print_function
import os,sys,stat
import argparse
import numpy as np
import h5py
import scipy.io
import matplotlib.pyplot as plt

np.random.seed(1337)  # for reproducibility

from keras.datasets import funcionesLFWUnificado
from keras.utils import np_utils, visualize_util
from keras.callbacks import ModelCheckpoint, Callback
from keras.models import model_from_json
from keras import backend as K

def get_activations(model, layer, X):
    fget_layer_output = K.function([model.layers[0].input],
                                   [model.layers[layer].output])
    layer_output = fget_layer_output([X])[0]
    return layer_output

parser=argparse.ArgumentParser()
parser.add_argument("-m","--model", help="Modelo a entrenar")
parser.add_argument("-e","--nb_epoch",help="Numero de epocas")
parser.add_argument("-l","--lista",help="Nombre lista")
args=parser.parse_args()
```

```
#Cambiar a la carpeta pasada por parametro
if args.model:
        nuevaruta=args.model
        os.chdir('/extra/scratch03/vmingote/keras/modelos/')
        os.chdir(nuevaruta)
        print ("Cambiado de directorio")
        print("Directorio:", os.getcwd())

def cargar_datos(args):
    nb_epoch=args.nb_epoch
    lista=args.lista
    return int(nb_epoch),lista

[nb_epoch,lista]=cargar_datos(args)

color=False
x_train,x_test,y_train,y_test,matrizId=funcionesLFWUnificado.
    load_dataArchivostxtOpenCV(nuevaruta,color,lista)
X_train=x_train.reshape(x_train.shape[0],128,128,1)
X_test=x_test.reshape(x_test.shape[0],128,128,1)

model = model_from_json(open('my_model_architecture.json').read())
model.summary()

Lista=funcionesLFWUnificado.leerarchivo('archivo.txt')
lon=len(Lista)

if (lon==nb_epoch):
    ruta=Lista[nb_epoch-1]
    print ("Pesos que se van a cargar: ",ruta)
    model.load_weights(ruta)
    print ("Pesos cargados")

else:
    print("El entrenamiento no se ha realizado correctamente.")
```

```
visualize_util.plot(model,'model.png')

cont=1
for i in range(len(X_test)):
    layer_output1 = get_activations(model, 7, X_test[i:i+1,:,:,:])
    id=y_test[i]
    nombre=matrizId[id-1][0]
    if id==y_test[i-1]:
        cont=cont+1

    else:
        cont=1

    archivo= nombre+'_'+str(cont)+".txt"
    f = open (archivo,'a')
    os.chmod(archivo,0744)
    np.savetxt(archivo,layer_output1)

    nombre3='yTest_'+str(lista)+'.txt'
    f = open (nombre3,'a')
    os.chmod(nombre3,0744)
    f.write('%s \n' %id)
    f.close()

    nombre4='ficherosTest_'+str(lista)+'.txt'
    f = open (nombre4,'a')
    os.chmod(nombre4,0744)
    f.write('%s \n' %archivo)
    f.close()
```

A.2.4 Comparación y reconocimiento

Con el código A.8 se realizó el cálculo de la métrica distancia coseno para cada pareja de imágenes de la base de datos LFW que se querían evaluar con el primer sistema implementado durante este trabajo.

Listing A.8: Comparacion y reconocimiento

```
from __future__ import print_function
import os,sys,stat
import argparse
import numpy as np
import h5py
import math
import scipy.spatial.distance
import matplotlib.pyplot as plt

np.random.seed(1337)  # for reproducibility

from keras.datasets import funcionesLFWUnificado
from keras.utils import np_utils,visualize_util
from keras.callbacks import ModelCheckpoint, Callback
from keras.models import model_from_json
from keras import backend as K
from PIL import Image
from sklearn import metrics
from sklearn.utils import shuffle
from math import*

parser=argparse.ArgumentParser()
parser.add_argument("-m","--model", help="Modelo a entrenar")
parser.add_argument("-e","--nb_epoch",help="Numero de epocas")
parser.add_argument("-c","--nb_classes",help="Numero de clases")
args=parser.parse_args()

batch_size=32

#Cambiar a la carpeta pasada por parametro
if args.model:
        nuevaruta=args.model
        os.chdir('/extra/scratch03/vmingote/keras/modelos/')
        os.chdir(nuevaruta)
        print ("Cambiado de directorio")
        print("Directorio:", os.getcwd())
```

```
def cargar_datos(args):
    nb_epoch=args.nb_epoch
    nb_classes=args.nb_classes
    return int(nb_epoch),int(nb_classes)

[nb_epoch,nb_classes]=cargar_datos(args)

X_test=[]
numeroImagenTe=[]
nombreImagenTe=[]

ficherosTest=funcionesLFWUnificado.leerarchivo('ficherosTest.txt')
y_test=funcionesLFWUnificado.leerarchivo('yTest.txt')

for i in ficherosTest:
   L_test=np.loadtxt(i)
   X_test.append(L_test)
   partes=i.split('_')
   parte=partes[len(partes)-1].split('.')
   numero=parte[0]

   if len(partes)==3:
        union1=(partes[0],partes[1])
        nombre="_".join(union1)

   elif len(partes)==4:
        union1=(partes[0],partes[1],partes[2])
        nombre="_".join(union1)

   elif len(partes)==5:
        union1=(partes[0],partes[1],partes[2],partes[3])
        nombre="_".join(union1)

   elif len(partes)==6:
        union1=(partes[0],partes[1],partes[2],partes[3],partes[4])
        nombre="_".join(union1)
```

```
    else:
        nombre=(partes[0])

    numeroImagenTe.append(numero)
    nombreImagenTe.append(nombre)

X_test=np.asarray(X_test)

test=True
te_pairs, te_y,etiq_te,imagenTe,nombreTe1,nombreTe2=
    funcionesLFWUnificado.
    load_data_parejasClasificadorWebCaracteristicas(nuevaruta,X_test,
    y_test,numeroImagenTe,nombreImagenTe,test)

x_test,y_test,etiquetaTest,fichTe,fichNombreTe1,fichNombreTe2=shuffle
    (te_pairs,te_y,etiq_te,imagenTe,nombreTe1,nombreTe2)

X_test=x_test[:, 0]
X2_test=x_test[:, 1]

def square_rooted(x):
        return round(sqrt(sum([a*a for a in x])),3)

def cosine_similarity(x,y):
        numerator = sum(a*b for a,b in zip(x,y))
        denominator = square_rooted(x)*square_rooted(y)
        return round(numerator/float(denominator),3)

resultados6=[]

for i in range (len(X_test)):
        caracteristica1=X_test[i][:]
        caracteristica2=X2_test[i][:]
        resultados6.append(cosine_similarity(caracteristica1,
            caracteristica2))

resultados6=np.asarray(resultados6)
```

```
media=np.mean(resultados6)
print(media)

y_pred=[]
for i in range (len(resultados6)):
        if resultados6[i]>media:
                y_pred.append(1)

        else:
                y_pred.append(0)

y_pred=np.asarray(y_pred)

fpr, tpr, _ = metrics.roc_curve(y_test,resultados6)
roc_auc=metrics.auc(fpr,tpr)

np.savetxt('fprParejasWeb.txt',fpr)
np.savetxt('tprParejasWeb.txt',tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve of class (area=%0.2f)' %roc_auc)

plt.plot( [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('ROC4ParejasWeb.png')
```

Anexo B

Comparativa tiempos de computación

En este anexo se encuentra una comparativa de los tiempos de computación al entrenar los sistemas que se implementaron durante este trabajo con distintos dispositivos. Para conseguir dicha comparativa se obtuvo el tiempo de entrenamiento haciendo uso de la CPU y las GPUs disponibles en los diversos nodos del cluster utilizado: GTX 560, GTX 980, GTX 1080 y TITAN X.

En la tabla B.1 se encuentran los resultados, en cuanto a tiempo de computación, de realizar una iteración del entrenamiento de las bases de datos LFW y CASIA con los distintos dispositivos mencionados anteriormente. En dicha tabla se puede apreciar la gran diferencia en el tiempo de computación que existe al utilizar los diferentes dispositivos disponibles para la realización de este trabajo.

Tabla B.1: Tabla de tiempos de computación de una iteración del entrenamiento al utilizar el primer sistema implementado.

Base de datos	Dispositivo utilizado	Modelo	Tiempo iteración
LFW	CPU	E5-2620 (i7)	7664" (0h 13' 50")
	GPU	GTX 560[1]	212" (0h 3' 32")
		GTX 980[2]	48" (0h 0' 48")
		GTX 1080[3]	26" (0h 0' 26")
		TITAN X[4] [5]	59" (0h 0' 59")
CASIA	CPU	E5-2620 (i7)	238262" (66h 11' 2")
	GPU	GTX 560	10562" (2h 56' 2")
		GTX 980	4250" (1h 10' 50")
		GTX 1080	1813" (0h 30' 13")
		TITAN X	4349" (1h 12' 29")

[1]url:http://www.nvidia.es/object/product-geforce-gtx-560-es.html

[2]url:http://www.nvidia.es/object/geforce-gtx-980-es.html#pdpContent=2

[3]url:https://www.nvidia.es/graphics-cards/geforce/pascal/gtx-1080/

[4]url:http://www.nvidia.es/object/geforce-gtx-titan-x-es.html#pdpContent=2

[5]We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.