Informe Short Project

Adrián Núñez Valenzuela

Gerard Queralt Ferré

Índex

Creació de la base de dades

Modificació de l'script de Python per descarregar imatges

El primer objectiu del projecte va ser poblar la base de dades d'imatges. Per fer-ho es va partir del codi de Python que se'ns va proporcionar, al que li vam fer unes lleugeres modificacions per facilitar-nos les execucions repetides [Codi 1].

El que vam fer va ser permetre-li acceptar arguments de la línia de comandes que li indiquessin quin tipus d'imatges volíem – ulls o no ulls – i, si cal, quantes. Un canvi senzill però important per nosaltres, que treballem en Windows, va ser canviar el nom temporal de l'arxiu que crea el mètode *save_image* d'"aux" a "auxiliary", ja que el sistema operatiu no permet que es creïn arxius amb el primer nom.

Crida de l'script de Python des de Matlab

Un cop modificat l'script, i per tal d'evitar que el codi de Matlab pogués executar-se per error amb la base de dades buida, vam fer un script que invocava el codi de Python vist al Codi 1 [Codi 2][Codi 3].

El funcionament d'aquest script de Matlab és el següent: es crida una funció amb un paràmetre que li indica quin tipus d'imatges volem. Després, es troba quantes imatges falten a la base de dades respecte les requerides a l'enunciat (300 ulls, 400 no ulls); en cas de faltar-ne, s'executa l'script de Python amb els paràmetres pertinents utilitzant la llibreria de Python de Matlab, inclosa per defecte.

Quan estan totes les imatges a la base de dades s'invoca un altre script que retalla les imatges de cares amb un rectangle trobat empíricament, de manera que només en queden els ulls[Codi 4].

Extracció de característiques i creació del model

Extracció de característiques

Aquesta és la part més important del projecte, i com a tal, en la que més s'ha iterat. Hem provat una gran varietat de descriptors amb resultats diversos, però finalment ens hem quedat amb els quatre que es poden apreciar al Codi 5: l'histograma de nivell de gris; la suma de nivells de gris de les files (possible gràcies a que redimensionem la imatge); la suma de la magnitud del gradient de les files; i la suma de la direcció del gradient de les files.

Un operador que hem descartat però volem destacar és la suma de nivells de gris de les columnes. En un primer moment ens va aportar una lleugera millora en la precisió, però a l'hora de provar el model ens vam adonar que només detectava els ulls drets. En principi vam pensar en duplicar les imatges de ulls i invertir-les, però ens vam adonar que l'únic descriptor que canviaria en fer això era precisament aquest sumatori de columnes, de manera que vam provar d'eliminar-lo. Efectivament el model detecta correctament els dos ulls, amb un cost de precisió teòrica inferior a l'1%. Per quest motiu les sumes del gradient es fan només respecte les files, i no les columnes.

Hem provat alguns altres descriptors, com per exemple la mitjana del gradient, el nivell de gris amb més aparicions, la mitjana de l'histograma, però en tots els casos suposaven una millora imperceptible, de manera que vam decidir prescindir-ne.

Fent algunes mesures utilitzant el Codi 6, el temps de càlcul d'aquestes característiques no supera mai la dècima de segon; el pitjor cas observat tarda aproximadament 0. 07 segons.

```
>> ComputeCharExtractionSpeed
Elapsed time is 0.069969 seconds.
```

Figura 1 Pitjor cas de computació de característiques

Creació del model

Un cop vam haver implementat els scripts i funcions necessaris per a extreure les característiques de la imatge, vam preparar l'script per crear el model [Codi 8]. Aquest codi crea una taula amb les característiques de les imatges de la base de dades. Un cop les etiquetades les guarda a un fitxer *.csv* i obri automàticament el *Classification Learner* de Matlab. Com es demana a l'enunciat, el model es de tipus SVM lineal, i utilitza la validació creuada 5 cops.

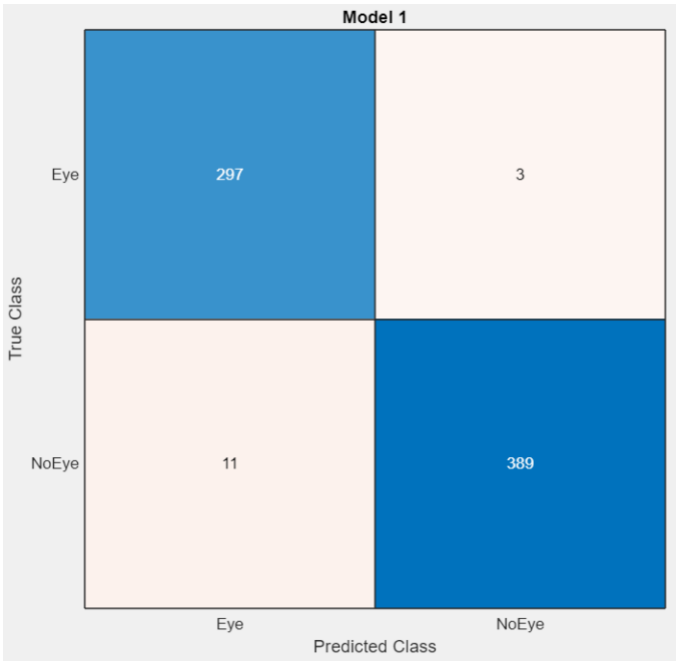

Figura 2 Matriu de confusió

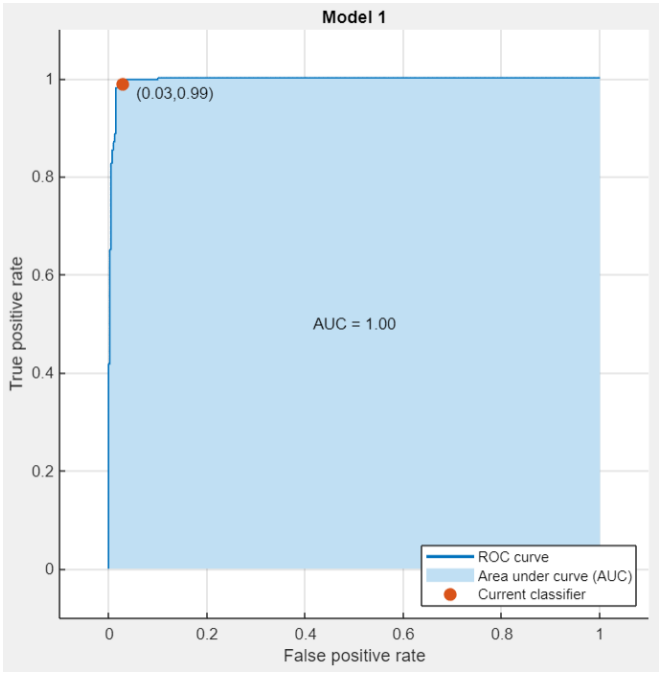

Figura 3 Curva ROC

Aquest model entrenat es guarda com una variable de l'entorn de treball amb l'opció *Export Model*, que s'utilitza en la resta del sistema.

Càlcul del mèrit

Per últim, hem de calcular el mèrit amb la fórmula que se'ns proporciona:

$$Mèrit = \sqrt{(1 - \frac{\text{Mida del vector de característiques}}{\text{Mida de la imatge}})^2 + Accuracy^2}$$

En aquesta fórmula hi ha un valor conegut, que és la mida de la imatge (48x32, com indica l'enunciat), i un valor molt fàcil de calcular, que és la mida del vector de característiques. Amb tres característiques, el rati de la mida del vector i la de la imatge és:

$$\frac{4}{48 \times 32} = \frac{4}{1536} = 0{,}0026$$

Hem hagut de calcular, doncs, la precisió del model. Per fer-ho fem una predicció i comparem els resultats amb l'etiqueta esperada. Això obté un vector de zeros i uns, on els zeros indiquen una predicció incorrecta i els uns una de correcta; per tant si sumem aquests valors i els dividim pel nombre total d'elements obtindrem la precisió del model[Codi 10].

El resultat d'executar el Codi 10 mostra una precisió al voltant del 99% (superior a la vista al *Classification Learner*) i un mèrit d'1,9971.

```
>> accuracy

accuracy =

    0.9943

>> merit

merit =

    1.9971
```

Figura 4 Precisió i mèrit del model

Demostració

Un cop creat i avaluat el model, només quedava provar-lo amb imatges reals. Per fer-ho es va escriure una funció que llegeix un vídeo *frame* a *frame* i, utilitzant el model i les funcions fetes en el capítol anterior, hi detecta els ulls i els marca. A partir de les imatges amb els ulls marcats es construeix un nou vídeo per poder comprovar el funcionament del model[Codi 11].

Detecció de les cares

Per cada *frame* del vídeo comencem detectant les cares que hi ha. Això, tal com recomana l'enunciat, ho fem amb un detector predefinit de Matlab que obté les capses contenidores d'allò que detecta com a cares. A partir d'aquestes capses nosaltres retallem el *frame* per crear un vector de possibles cares[Codi 12].

Detecció d'ulls

A partir d'aquest vector de possibles cares volem detectar els ulls que hi hagin en cadascuna d'elles. D'això se n'encarrega la funció *FindEyes*[Codi 13], que processa cada cara per blocs i calcula una imatge binària on els píxels blancs seran aquells blocs que pertanyin a un ull.

Marcatge d'ulls i creació del vídeo

Per últim, es redimensionen aquestes imatges binàries – on cada píxel representa un bloc – per a que tinguin la mida de la cara. Després filtrem la imatge binària per deixar-hi només l'element més gran, per evitar errors provocats pel soroll de la imatge, i n'obtenim la capsa contenidora. Finalment la marquem en el *frame*, que s'escriu en el vídeo resultant[Codi 11]

Annex

Índex de codis

Script de Python per descarregar imatges

```
import os
import requests
from PIL import Image
import imagehash
import time
import sys

PATH = "images"
Set = set()
Counter = 0


def initialize_set():
    s = set()
    try:
        with open('hashes.txt') as f:
            lines = f.readlines()
        for line in lines:
            s.add(line)
    except:
        print("It seems it is your first execution, no problem :)")
    finally:
        print("Set initialized to avoid repetitions from previous executions")
    return s


def save_image(picture):
    file = os.path.join(PATH, "auxiliary.jpg")

    with open(file, "wb") as f:  # write byte
        f.write(picture)
    img = Image.open(file)
    hashfile = str(imagehash.average_hash(img))
    if hashfile not in Set:
        Set.add(hashfile)
        img.save(os.path.join(PATH, str(hashfile) + ".jpg"))

        with open('hashes.txt', "a") as f2:  # to append
            f2.write(str(hashfile) + "\n")
        f2.close()

        global Counter
        Counter += 1
        print("Successfully Saved" + " -- Image #: " + str(Counter))
```

```
    else:
        print("Already Existed")
    img.close()
    os.remove(file)
    # cerramos y eliminamos aux


def get_image_from_web():
    if eyeImages:
        url = "https://thispersondoesnotexist.com/image"
    else:
        url = "https://picsum.photos/200/300"
    r = requests.get(url).content
    return r


def create_dir():
    try:
        os.mkdir(PATH)
    except OSError:
        print("Creation of the directory %s failed" % PATH)
    else:
        print("Successfully created the directory %s " % PATH)


def objective_reached(num_images):
    global Counter
    return Counter >= num_images


def main(num_dif_images = 0):
    create_dir()
    global Set
    Set = initialize_set()

    if num_dif_images == 0:
        if eyeImages:
            num_dif_images = 300
        else:
            num_dif_images = 400
    while not objective_reached(num_dif_images):
        img = get_image_from_web()
        save_image(img)
        time.sleep(0.5)
    return
```

```
if __name__ == "__main__":
    eyeImages = True
    if len(sys.argv) > 1:
        secondArg = sys.argv[1]
        if secondArg == "false" or secondArg == "False" or secondArg == "noEyes":
            eyeImages = False
        elif secondArg != "true" and secondArg != "True" and secondArg != "eyes":
            print('Argument ', secondArg, ' not recognised; assuming eye images wanted')
    if len(sys.argv) > 2:
        main(int(sys.argv[2]))
    else:
        main()
```

Codi 1 get_images

Script per poblar la base de dades

```
PopulateDatabaseFolder(true);
PopulateDatabaseFolder(false);
ExtractEyes;
```

Codi 2 PopulateDatabase

Funció per poblar un arxiu concret de la base de dades

```
function [] = PopulateDatabaseFolder(eyes)
    if eyes
        path = what('Eyes').path;
        eyesArg = 'true';
        target = 300;
    else
        path = what('NoEyes').path;
        eyesArg = 'false';
        target = 400;
    end
    images = dir(append(path,'\*.jpg'));
    L = length(images);
    target = target - L;
    if target > 0
        getImagesPath = what('GetImages').path;
        oldDir = cd(getImagesPath);
        pythonCall = append('get_images.py ', eyesArg, ' ', string(target));
        pyrunfile(pythonCall);
        movefile(append('images\*.jpg'), path);
        rmdir('images');
        cd(oldDir);
    end
end
```

Codi 3 PopulateDatabaseFolder

Script per retallar les imatges de cares per extreure els ulls

```
% rectangle found empirically
rect = [283.112, 446.532, 185.909, 82.459];
fullPath = what('Eyes').path;
eyes = dir(append(fullPath,'\*.jpg'));
L = length(eyes);
for i = 1:L
    I = imread(eyes(i).name);
    [r, c] = size(I);
    if r > 187 && c > 87 % not already cropped
        I = imcrop(I, rect);
        fullname = append(eyes(1).folder, '\', eyes(i).name);
        imwrite(I, fullname);
    end
end
```

Codi 4 ExtractEyes

Funció per extreure les característiques d'una imatge

```
function [characteristics] = ExtractCharacteristics(grayImage)
    [~, ~, channels] = size(grayImage);
    if(channels == 3) % RGB image
        grayImage = rgb2gray(grayImage);
    end
    I = imresize(grayImage, [48, 32]);

    % graylevel histogram
    hist = imhist(I)';
    % sum of rows
    sumRow = sum(I,2)';
    % sum of rows of gradient magnitude and direction
    [Gmag, Gdir] = imgradient(I,'sobel');
    sumRowGmag = sum(Gmag, 2)';
    sumRowGdir = sum(Gdir, 2)';

    characteristics = table(hist, sumRow, sumRowGmag, sumRowGdir);
end
```

Codi 5 ExtractCharacteristics

Script per analitzar el temps de càlcul de les característiques

```
eyesImages = GetAllImagesInDatabaseFolder('Eyes');
n = numel(eyesImages);
index = round(randi(n));
image = eyesImages{index};
image = rgb2gray(image);
tic
chars = ExtractCharacteristics(image);
toc
```

Codi 6 ComputeCharExtractionSpeed

Funció per obtenir totes les imatges d'un directori

```
function [images] = GetAllImagesInDatabaseFolder(folderName)
    fullPath = what(folderName).path;
    dirFiltered = append(fullPath,'\*.jpg');
    imageFiles = dir(dirFiltered);
    L = length(imageFiles);
    images = cell(1, L);
    for i = 1:L
        I = imread(imageFiles(i).name);
        images{i} = I;
    end
end
```

Codi 7 GetAllImagesInDatabaseFolder

Script per crear el model

```
PopulateDatabase;
if exist('eyesImages', 'var') == 0
    eyesImages = GetAllImagesInDatabaseFolder('Eyes');
end
if exist('noEyesImages', 'var') == 0
    noEyesImages = GetAllImagesInDatabaseFolder('NoEyes');
end
eyesChars = ExtractCharacteristicsFromImages(eyesImages);
noEyesChars = ExtractCharacteristicsFromImages(noEyesImages);
chars = [eyesChars; noEyesChars];
lables = table('Size', [700, 1],'VariableTypes',"string",'VariableNames',"Expected");
lables(1:300,1) = {'Eye'};
lables(301:end,1) = {'NoEye'};
chars = [chars, lables];

databasePath = what('VC-EyeDetector\Database').path;
oldDir = cd(databasePath);
writematrix(table2array(chars), 'DescriptorTable.csv');
cd(oldDir);

classificationLearner(chars, "Expected")
```

Codi 8 CreateModel

Funció per extreure les característiques d'un conjunt d'imatges

```
function [charsTable] = ExtractCharacteristicsFromImages(images)
    charsTable = cellfun(@ExtractCharacteristics, images, 'UniformOutput', false);
    charsTable = cell2table(charsTable');
    charsTable = splitvars(charsTable);
end
```

Codi 9 ExtractCharacteristicsFromImages

Script per calcular el mèrit del model

```
prediction = trainedModel.predictFcn(chars);
%   from:   https://es.mathworks.com/matlabcentral/answers/388575-convert-string-array-into-cell-array#answer_310224
expected = arrayfun(@(x)char(chars.Expected(x)),1:numel(chars.Expected),'uni',false)';
accuracy = sum(strcmpi(expected,prediction),'all')/numel(expected);

charsSize = width(chars) - 1; % we don't consider the label
imSize = 48 * 32;
merit = sqrt(pow2(1 - charsSize/imSize) + pow2(accuracy));
```

Codi 10 ComputeMerit

Funció per processar un vídeo i marcar-hi els ulls

```
function [] = Demo(videoName, trainedModel)
    % Create a cascade detector object.
    FaceDetector = vision.CascadeObjectDetector('FrontalFaceLBP');
    % Read a video frame and run the face detector.
    videoReader = VideoReader(videoName);
    videoNameSplit = split(videoName, '.');
    resultName = strcat(videoNameSplit(1), 'Result');
    resultName = resultName{1};
    videosPath = what('Videos').path;
    oldDir = cd(videosPath);
    result = VideoWriter(resultName);
    open(result);
    while hasFrame(videoReader)
        % get the next frame
        videoFrame = readFrame(videoReader);
        [r, c, ~] = size(videoFrame);
        bboxes = step(FaceDetector, videoFrame);

        faces = ExtractFaces(videoFrame, bboxes);

        eyesFound = cellfun(@(face) FindEyes(face, trainedModel), faces, 'UniformOutput', false);

        for i = 1:numel(faces)
            [rf, cf, ~] = size(faces{i});
            eyes = imresize(eyesFound{i}, [rf, cf]);
            bb = bboxes(i,:);
            eyeMask = false(r, c);
            eyeMask(bb(2):bb(2)+bb(4), bb(1):bb(1)+bb(3)) = eyes;
            largestRegion = bwareafilt(eyeMask,1);
            props = regionprops('table', largestRegion, 'BoundingBox');
            videoFrame = insertShape(videoFrame, 'Rectangle', props.BoundingBox);
        end
        writeVideo(result, videoFrame);
    end
    close(result);
    cd(oldDir);
end
```

Codi 11 Demo

Funció per retallar les cares

```
function [faces] = ExtractFaces(videoFrame,bboxes)
    [nBoxes, ~] = size(bboxes);
    faces = cell(1, nBoxes);
    for i = 1:nBoxes
        bb = bboxes(i,:);
        I = imcrop(videoFrame, bb);
        faces{i} = I;
    end
end
```

Codi 12 ExtractFaces

Funció per processar una imatge per blocs i indicar si pertany o no a un ull

```
function [res] = FindEyes(face, trainedModel)
    grayFace = rgb2gray(face);
    step = 10;
    % rectangle found empirically
    rect = [283.112, 446.532, 185.909, 82.459];
    h = round((rect(4) - step)/2);
    w = round((rect(3) - step)/2);
    blk = [h, w];
    res = blkproc(grayFace, [step, step], blk, @(block) IsEye(block, trainedModel));
    res = logical(res);
end
```

Codi 13 FindEyes

Funció que determina si una imatge pertany o no a un ull

```
function [eye] = IsEye(image, trainedModel)
    chars = ExtractCharacteristics(image);
    prediction = trainedModel.predictFcn(chars);
    eye = strcmp(prediction{1},'Eye');
end
```

Codi 14 IsEye