FIB Facultat d'Informàtica de Barcelona

Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial
UNIVERSITAT POLITÈCNICA DE CATALUNYA

VISIÓN POR COMPUTADOR

Short Project (Checkpoint)

Facultad de Informática de Barcelona

Adrian Cristian Crisan

Filip Gedung Dorm

Pablo Vega Gallego

Barcelona, diciembre de 2021

Índice

1. Objetivos del proyecto

El objetivo del proyecto es implementar un sistema automático para detectar la mirada mediante visión por computador. El sistema ha de ser capaz de detectar dónde están los ojos y posteriormente medir el ángulo horizontal de la mirada en base a la posición del iris. Las imágenes con las que se trabajarán serán con un encuadro tipo busto de un usuario sin gafas en una orientación vertical con rotaciones de la cabeza poco apreciables; tal como se muestra a continuación en la imagen de la figura siguiente:

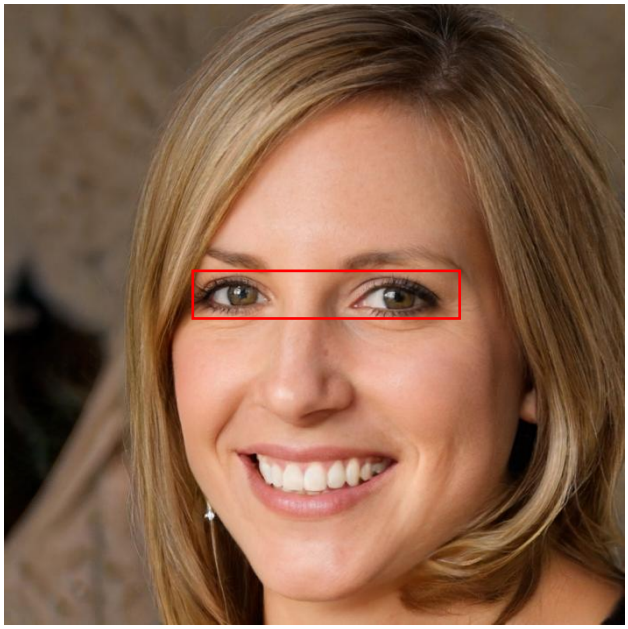

Figura 1 – Imagen de ejemplo de una detección de ojos (imagen extraída de thispersondoesnotexist.com)

2. Recopilación de la base de datos

Para que nuestro programa sepa si la imagen que estamos analizando contiene ojos o no, necesitamos que nuestra base de datos esté compuesta por imágenes con ojos e imágenes sin ojos.

2.1. Imágenes de ojos

Para este paso, hemos recogido las imágenes ofrecidas por la asignatura y hemos eliminado las personas que tenían gafas y añadido nuevas imágenes, teniendo en total de 220 caras.

Dado que todas las imágenes con ojos tendrán estos en aproximadamente la misma posición y tendrán el mismo tamaño siempre, podemos recortar todas las imágenes en la misma posición. Por lo que para recortar las imágenes con ojos, hemos ido ajustando el recorte para que abarque todos los ojos de las personas, dando cierto margen.

Figura 2.1 – Ejemplo de recorte para la base de datos para los ojos

El recorte se realiza en el punto (256, 426) y tiene unas dimensiones de 513 de ancho por 113 de alto, en el anexo podemos ver cómo se han realizado los cálculos y la ejecución de este apartado.

2.2. Imágenes de no ojos

En este caso, para obtener imágenes de no ojos, o de un solo ojo, lo que hemos hecho, han sido un programa que haga recortes de la misma dimensión que en el caso anterior, pero realizando recortes de puntos escogidos de manera aleatorio de las propias caras, por cada cara recogemos 20 recortes de no ojos. Lo hacemos de la siguiente manera:

1. Seleccionamos 2 puntos aleatorios de la imagen entre 1 y 1024.
2. Comprobamos que estos puntos no se encuentren en el área resaltada de color rojo en la figura 2.2.
    a. Esta área toma como punto central el punto (256, 426).
    b. Se extiende 512/2 por la izquierda y por la derecha formando un rectángulo de 512 de ancho.
    c. Para la altura de esta área prohibida hacemos lo mismo que en el caso anterior, pero obteniendo como resultado 112 píxeles de altura.
3. Hacemos el recorte de la imagen.

De esta manera nos aseguramos de que los recortes que hace el algoritmo no se encuentren los dos ojos enteros. A su vez tenemos también partes de ojos que no consideramos ojos enteros y que estarían mal detectarlos como ojos.

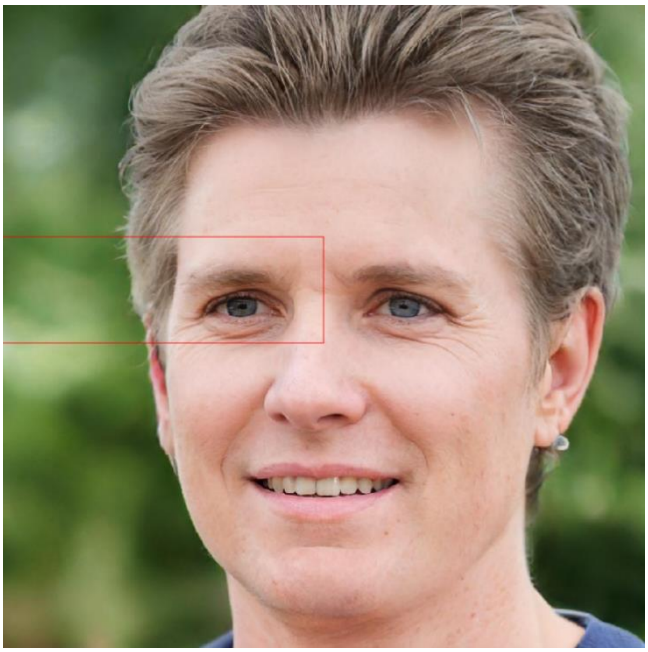

Figura 2.2 – Ejemplo de área prohibida para recorte de trozos de cara sin ojos

Para asegurarnos que el programa si ve un ojo lo detecte como no ojo, hemos creado un algoritmo que realiza el recorte por cada imagen de una persona su ojo izquierdo y derecho y lo hemos añadido a la base de datos de no ojos.

Figura 2.3 – Ejemplo recorte de solo un ojo (izquierdo y derecho)

Por último para tener referencias de espacios comunes que podrían salir en escenas de no ojos hemos seleccionado un conjunto de imágenes extraídas de la siguiente página web: https://search.bwh.harvard.edu/new/CBDatabase.html dónde se recopilan una serie de imágenes de oficinas e interiores.

Este conjunto de fondos contenía repeticiones de la misma imagen con pequeñas variaciones, por lo que creamos un algoritmo en Python que eliminara estas fotos que en nuestro caso hemos considerado repetidas.

Para este último paso, hemos cogido por cada imagen de fondo que teníamos, 18 recortes de esta misma de tamaño 113x513 (alto x ancho). Para evitar repeticiones los puntos de recorte el programa los escoge de manera aleatoria.

Finalmente, obtenemos un total de 11.032 imágenes de no ojos y 220 de ojos, lo que hace una proporción de $50.1\overline{45}$ imágenes de no ojos por cada una de ojos.

A modo de resumen, dejamos especificadas de donde sale cada imagen:

- Imágenes de ojos (220 imágenes)
  - 220 imágenes de caras
- Imágenes de no ojos (11.032 imágenes)
  - $220\ caras\ *\ \frac{20\ trozos}{cara} = 4400\ imágenes\ de\ trozos\ de\ cara$
  - $220\ caras * \frac{2\ ojos}{cara} = 440\ ojos$ (sólo un ojo)
  - $344\ fondos * 18\ recortes = 6192\ imágenes\ de\ fondos$

Dejamos un enlace al Google drive con un .zip para descargar los datasets.

2.3. Train/Test sets

Para seleccionar los conjuntos de imágenes de manera aleatoria hemos creado un script de Python que seleccione las imágenes, las introduzca en una lista y las mezcle de manera aleatoria, posteriormente copia las imágenes que corresponden al train y test de los ojos y no ojos en sus respectivas carpetas para usarlas a continuación.

Finalmente, para que los algoritmos que usaremos vayan más rápido, en concreto para evitar las lecturas por separado de cada imagen que usaremos, creamos una matriz de todas las imágenes para el *train de los ojos*, *test de los ojos*, *train de los no ojos* y *test de los no ojos*.

El porcentaje de fotos que usamos para entrenar los modelos es del 90%, el 10% restante nos lo quedamos para introducir nuevos inputs al probar los modelos.

3. HOGs

Para la primera entrega del Short Project hemos decidido utilizar HOGs, para ello, Matlab tiene una función *extractHOGFeatures* que nos permite extraer las características de una imagen y nos la deja el resultado en un vector de características. Una de las opciones que nos deja configurar esta función es el tamaño de las celdas que se usan para recorrer la imagen.

Usamos un algoritmo que extrae estas características de las imágenes que tenemos, usamos 2 configuraciones diferentes para observar los resultados del parámetro “CellSize”, primero con la opción por defecto [8 8] y posteriormente la opción [16 16].

Para la primera opción el vector de características tiene una dimensión de 29484 por cada imagen. En cambio, cuando aumentamos la celda de tamaño se reduce a 6696 por cada imagen.

4. Training

Para este paso, insertamos la tabla creada en el paso anterior en la app llamada "Classification Learner" y seleccionamos todos los modelos que hay disponible en la aplicación. Para que vaya más rápido, nos hemos descargado un Add-on de Matlab, llamada *Parallel Computing Toolbox*, que nos permite paralelizar el entrenamiento de los modelos e ir más rápido en este proceso.

Para todos los modelos usamos el método Holdout Validation, dado que con el Cross-Validation no llegaba a ejecutarse por completo ningún modelo sin un resultado satisfactorio.

Después de una larga espera, obtenemos fallos en casi todos los entrenamientos, excepto en los siguientes:

Con el *CellSize* de [8 8]:

- Fine Tree con accuracy de 98.8%.

Con el *CellSize* de [16 16]:

- Linear Discriminant con un accuracy de 98.9%.

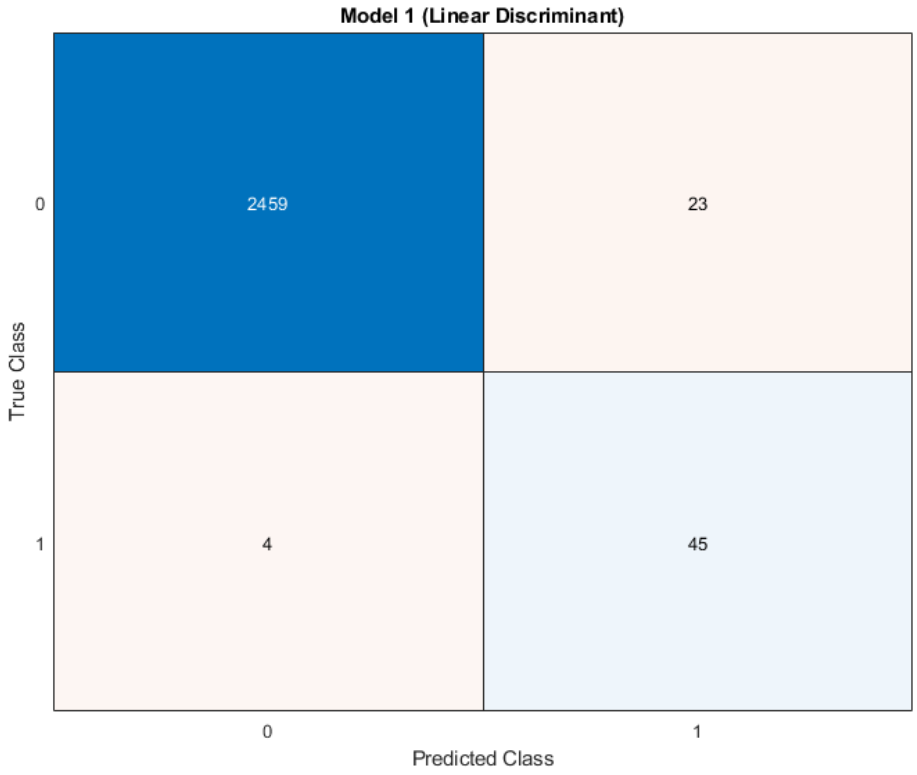

Figura 4.1 – *Confusion Matrix* del modelo *Linear Discriminant* con el *kernel* 16

- Cosine KNN con un accuracy de 100%.

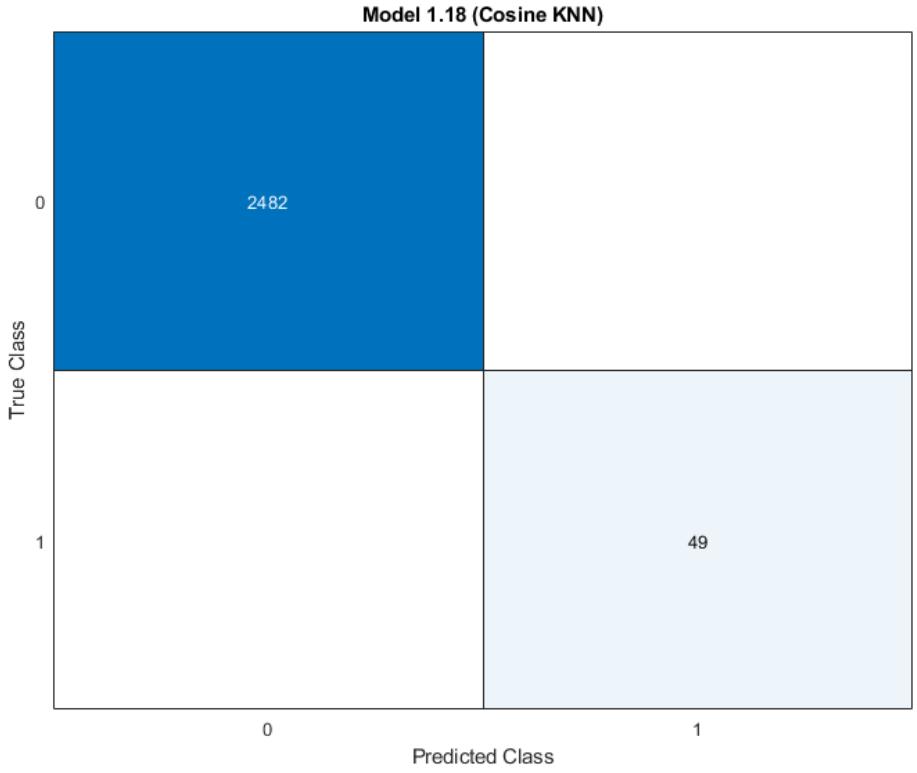

Figura 4.2 – *Confusion Matrix* del modelo *Cosine KNN* con el *kernel* 16

- Weighted KNN con un accuracy de 99.9%.

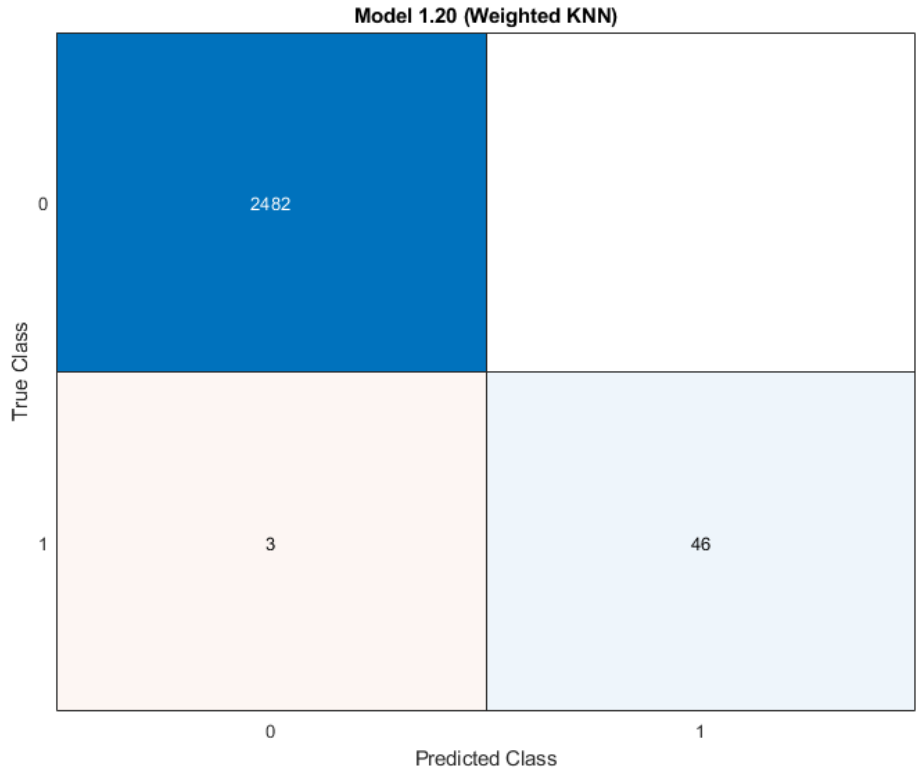

Figura 4.3 – *Confusion Matrix* del modelo *Weighted KNN* con el *kernel* 16

5. Resultados

Cuando usamos las imágenes que hemos reservado nosotros para testear los modelos obtenemos los siguientes resultados:

Con el *CellSize* de [8 8]:

- Fine Tree con accuracy de 98.8%.

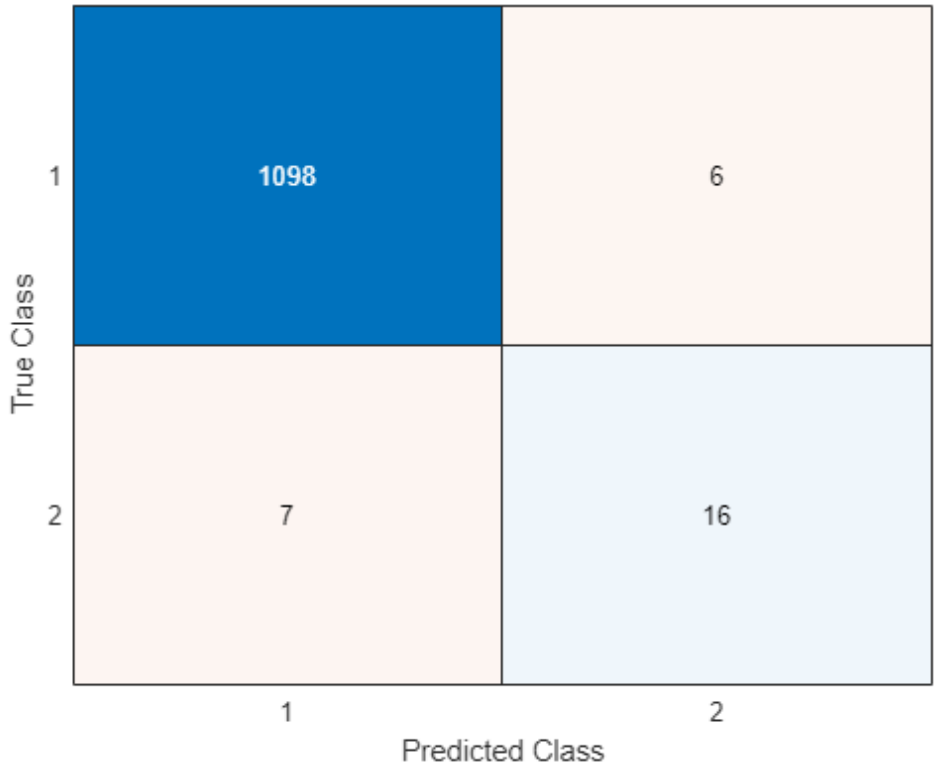

Figura 5.1 – *Confusion Matrix* del Test Data del modelo *Fine Tree* con el *kernel* 8

Con el *CellSize* de [16 16]:

- Linear Discriminant con un accuracy de 99.56%.

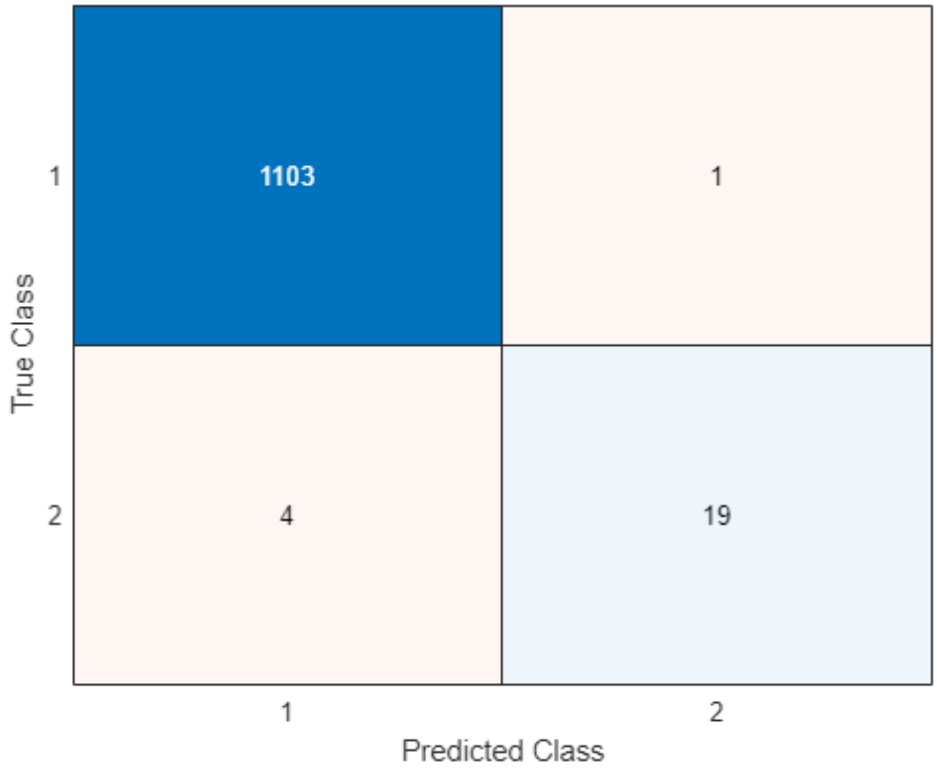

Figura 5.2 – *Confusion Matrix* del Test Data del modelo *Linear Discriminant* con el *kernel* 16

- Cosine KNN con un accuracy de 99.82%.

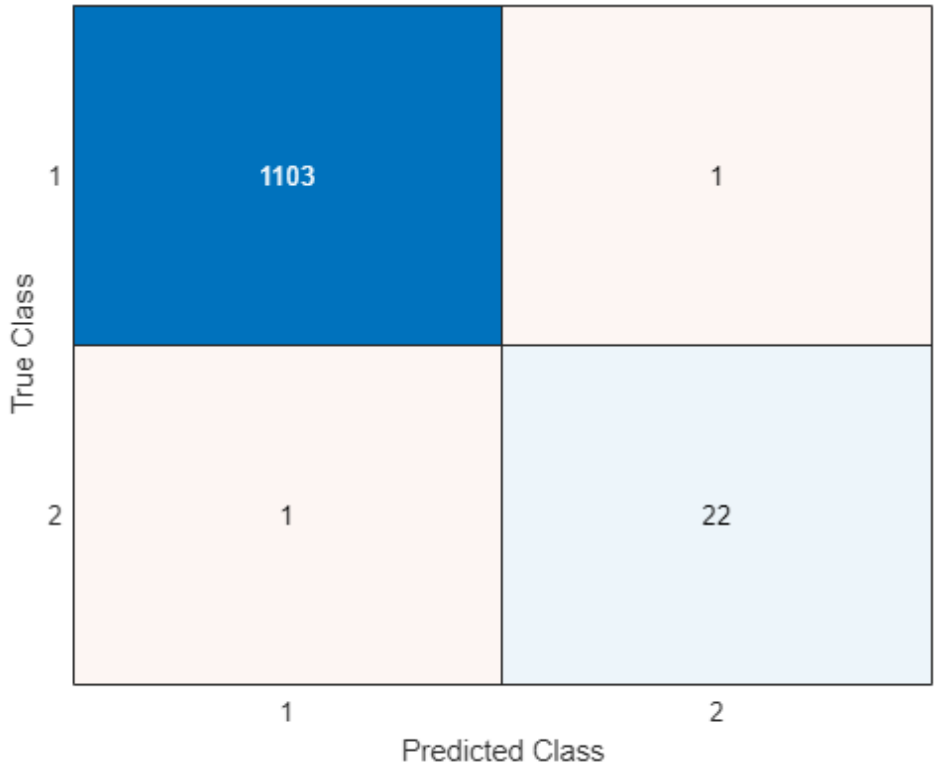

Figura 5.3 – *Confusion Matrix* del Test Data del modelo *Cosine KNN* con el *kernel* 16

- Weighted KNN con un accuracy de 99.73%.

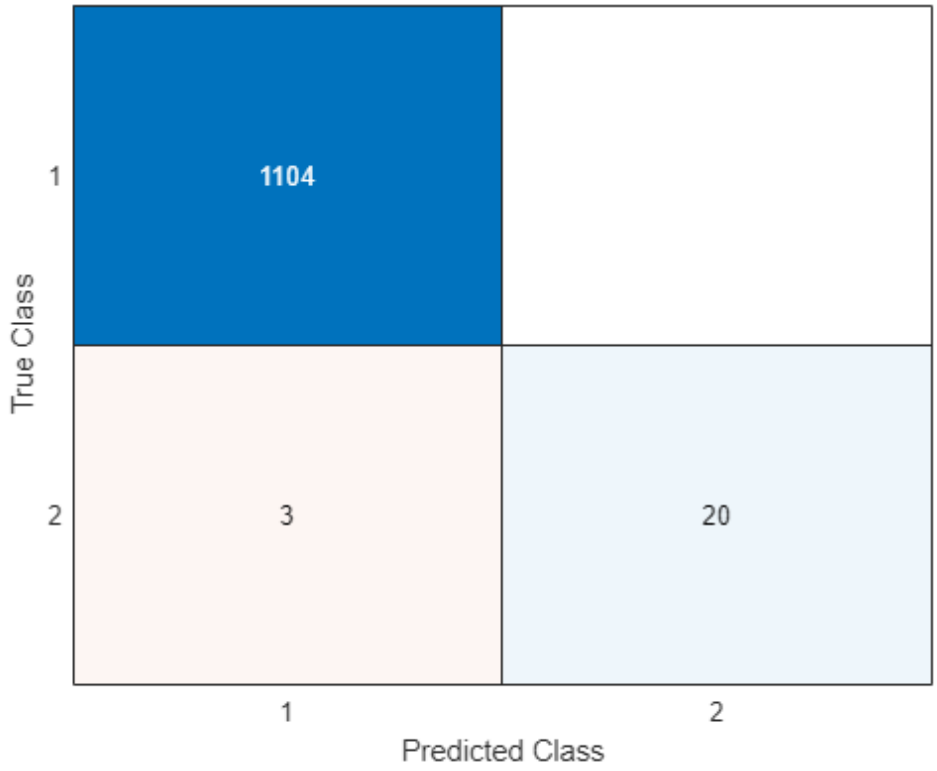

Figura 5.4 – *Confusion Matrix* del Test Data del modelo *Weighted KNN* con el *kernel* 16

Para comprobar los modelos usamos un algoritmo que crea una tabla que la pasa como parámetro a una función llamada *predictFcn()* que nos la ofrece el modelo.

Podemos ver que los resultados obtenidos, aunque no sean perfectos, son bastante satisfactorios.

Para el algoritmo final de esta entrega, hemos hecho un programa que te haga una foto en tiempo real con la webcam, usando el add-on MATLAB Support Package for USB Webcams, posteriormente, se nos abre un pop-up para seleccionar el trozo de imagen que queremos que nos detecte y hacemos un resize para que la imagen sea 113 de alto y 513 de ancho, finalmente selecciona un modelo y predice si es un ojo o no ojo. Dejamos aquí una muestra de cómo sería el proceso.

Obtenemos una foto con la webcam:

Figura 5.5 – Imagen de prueba tomada por la webcam

Se abre un pop-up y seleccionamos el trozo de imagen que queremos que nos detecte:

Figura 5.6 – Imagen con la selección del trozo que queremos

Se hace el resize de la imagen y se pasa por parámetro para que el modelo nos detecte si son ojos o no.

Figura 5.7 – Imagen a detectar

```
ejecutarPredictor(imagenRecortada, CosineKNNModelKernel16)
    ans = single
    1
```

Llamamos a la función *ejecutarPredictor* y nos devuelve 1, dado que detecta ojos.

Dejamos otro ejemplo de no ojos con la misma imagen que la figura 5.5.

Seleccionamos el trozo de imagen que queremos:

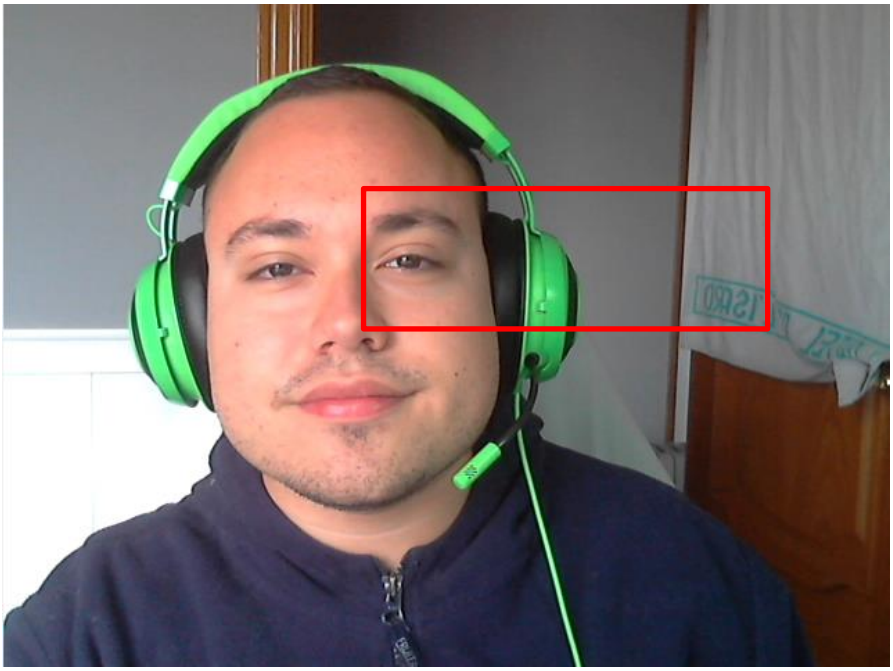

Figura 5.8 – Selección de imagen de no ojo

Se hace el resize de la imagen y se pasa por parámetro para que el modelo nos detecte si son ojos o no.

Figura 5.9 – Recorte y resize de la imagen de no ojo

```
ejecutarPredictor(imagenRecortada, CosineKNNModelKernel16)
    ans = single
    0
```

Llamamos a la función *ejecutarPredictor* y nos devuelve 0, dado que no detecta ojos.

Hemos detectado un problema con nuestra base de datos de ojos, la cual, es muy estricta a la hora de seleccionar que es un ojo, ya que como se puede ver en la figura 5.10 no tenemos en cuenta el ruido que podría haber al costado de los ojos, cosa que provoca que cuando seleccionamos un área muy amplia de ojos y no ojos con mucha probabilidad detectará no ojos. Para la entrega final mejoraremos en este apartado.

Figura 5.10 – Recorte de ojos para la base de datos

La velocidad de cálculo que nos sale al ejecutar nuestro programa es de aproximadamente 3'5 segundos. Para calcular esto hemos usado la función tic toc, que calcula el tiempo que pasa entre el tic y el toc.

```
tic
ejecutarPredictor(imagenRecortada, CosineKNNModelKernel16)
   ans = single
   0
toc
   Elapsed time is 3.573467 seconds.
```

6. Funciones usadas

Funciones que no son nuestras:

- dir()
- length()
- height()
- width()
- size()
- floor()
- rand()
- imwrite()
- imread()
- imcrop()
- rgb2gray()
- extractHOGFeatures()
- zeros()
- ones()
- strings()
- int2str()
- transpose()
- array2table()

- model.predictFcn()
- confusionchart()
- confusionmat()
- webcam()
- getrect()
- snapshot()
- imshow()
- imresize()
- single()
- tic
- toc

El resto de las funciones y código es completamente nuestro.

7. Anexo

Programa de recorte de ojos.

```
imagefiles = dir('VC/Final Project/DataFaces/images_faces/*.jpg');
for i = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    alto = height(image);
    ancho = width(image);
    x = floor(ancho/4);
    y = floor(alto/2.4);
    distance_h = floor(ancho/4 + ancho/2) - x;
    distance_w = floor(alto/1.9) - y;

    imwrite(imcrop(image, [x, y, distance_h, distance_w]), "eye_" + i +
".jpg");
end
```

Programa para recortar diferentes trozos de la cara.

```
% Recogemos los datos de recorte de una imagen de prueba para no hacer los
cálculos en todas las iteraciones.

image = imread("VC/Final
Project/DataFaces/images_faces/000c3c3ebe3e1c1e.jpg");

alto = height(image);
ancho = width(image);
x = floor(ancho/4);
y = floor(alto/2.4);
distance_x = floor(ancho/4 + ancho/2) - x;
distance_y = floor(alto/1.9) - y;

% Recorremos y recortamos las imagenes
count  = 1;
imagefiles = dir('VC/Final Project/DataFaces/images_faces/*.jpg');
for i = 1:length(imagefiles)
```

```
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    prohibited_x_left = 1;
    prohibited_x_right = x + distance_x / 2;
    prohibited_y_top = y - distance_y / 2;
    prohibited_y_bottom = y + distance_y;

    for j = 1:20
        T = false;
        while (~T)
            random_x = rand() * (ancho - 1) + 1;
            random_y = rand() * (alto - 1) + 1;

            cond1 = prohibited_y_top > random_y || random_y >
prohibited_y_bottom;
            cond2 = prohibited_x_left > random_x || random_x >
prohibited_x_right;
            if (cond1 || cond2)
                h = floor(random_x + distance_x);
                w = floor(random_y + distance_y);
                if (h < alto && w < ancho)
                    T = true;
                end
            end
        end

        imwrite(imcrop(image, [random_x, random_y, distance_x, distance_y]),
"VC/Final Project/DataFaces/not_eyes/not_eye_" + count + ".jpg");
        im = imread("VC/Final Project/DataFaces/not_eyes/not_eye_" + count +
".jpg");
        count = count + 1;
    end
end
```

Algoritmo de recorte de los ojos por separado.

```
count = 1;
imagefiles = dir('VC/Final Project/DataFaces/images_faces/*.jpg');
for i = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    alto = height(image);
    ancho = width(image);
    x = floor(ancho/4);
    y = floor(alto/2.4);
    distance_x = floor(ancho/4 + ancho/2) - x;
    distance_y = floor(alto/1.9) - y;

    alto = height(image);
    ancho = width(image);
    x = floor(ancho/4);
    y = floor(alto/2.4);
```

```
    distance_x = floor(ancho/4 + ancho/2) - x;
    distance_y = floor(alto/1.9) - y;

    imwrite(imcrop(image, [1, y, distance_x, distance_y]), "VC/Final
Project/DataFaces/not_eyes/half_eyes_" + count + ".jpg");
    count = count + 1;
    imwrite(imcrop(image, [distance_x, y, distance_x, distance_y]),
"VC/Final Project/DataFaces/not_eyes/half_eyes_" + count + ".jpg");
    count = count + 1;
end
```

Programa que elimina imágenes repetidas de las imágenes que usamos como fondo.

```
# import required module
import os
import random
import shutil
# assign directory
# iterate over files in
# that directory
print("hola")
n = int(input("Introduce cada cuantas imagenes quieres borrar\n"))
directory = input("Introduce nombre de la carpeta\n")
donde = input("Donde\n")

lista = os.listdir(directory)
longitud = len(lista)

lists = list(range(0, n))

print(lista)

print(directory)
print(donde)

for i in range(0, longitud, n):
    adder = random.choice(lists)
    indice = i + adder

    original = directory + "\\" + lista[indice]
    target = donde + "\\" + lista[indice]

    print("original: " + original)
    print("copia: " + target)

    shutil.copyfile(original, target)
```

Programa que recorta las imágenes de fondo.

```
image = imread("VC/Final
Project/DataFaces/images_faces/000c3c3ebe3e1c1e.jpg");
```

```
alto = height(image);
ancho = width(image);
x = floor(ancho/4);
y = floor(alto/2.4);
distance_x = floor(ancho/4 + ancho/2) - x;
distance_y = floor(alto/1.9) - y;

count = 1;
imagefiles = dir('VC/Final Project/DataFaces/backgrounds/*.jpg');
for i = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);
    alto = height(image);
    ancho = width(image);

    for j = 1:18
        T = false;
        while (~T)
            random_x = rand() * (ancho - 1) + 1;
            random_y = rand() * (alto - 1) + 1;
            w = random_x + distance_x;
            h = random_y + distance_y;
            if (w < ancho && h < alto)
                T = true;
            end
        end

        imwrite(imcrop(image, [random_x, random_y, distance_x, distance_y]),
"VC/Final Project/DataFaces/not_eyes/background_" + count + ".jpg");
        count = count + 1;
    end
end
```

Programa que comprueba que todas las imágenes son de tamaño 113x513

```
imagefiles = dir('VC/Final Project/DataFaces/not_eyes/*.jpg');
for i = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    if (height(image) ~= 113 || width(image) ~= 513)
        currentimage
    end
end
```

Programa que mezcla las imágenes para el train y el test de ojos y no ojos aleatoriamente.

```
import os
import random
import shutil

directory = input("Introduce nombre de la carpeta\n")
```

```
donde1 = input("Donde el train\n")
donde2 = input("Donde el test\n")
porcentaje = input("Porcentaje del train (valores de 0 a 1)\n")

lista = os.listdir(directory)
longitud = len(lista)

random.shuffle(lista)
print(lista)
n = longitud * float(porcentaje);

for i in range(0, int(n)):
    original = directory + "\\" + lista[i]
    target = donde1 + "\\" + lista[i]

    print("original: " + original)
    print("copia: " + target)

    shutil.copyfile(original, target)

for i in range(int(n), longitud):
    original = directory + "\\" + lista[i]
    target = donde2 + "\\" + lista[i]

    print("original: " + original)
    print("copia: " + target)

    shutil.copyfile(original, target)
```

Algoritmo que pasa las imágenes a una matriz.

```
imagefiles = dir('eyes_test/*.jpg');
for i = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    EyesTestData(:, :, :, i) = image;
end
```

Algoritmo que usamos para extraer el vector de características.

```
load EyesTrainData.mat
load NotEyesTrainData.mat

image = EyesTrainData(:, :, :, 1);
imageGray = rgb2gray(image);
cell = [16 16];
hog = extractHOGFeatures(imageGray, 'CellSize', cell);

mida_eyes = size(EyesTrainData);
train_data_eyes = zeros(mida_eyes(4), width(hog));
for i=1:mida_eyes(4)
    image = EyesTrainData(:, :, :, i);
```

```
    imageGray = rgb2gray(image);
    train_data_eyes(i, :) = extractHOGFeatures(imageGray, 'CellSize', cell);
end

mida_not_eyes = size(NotEyesTrainData);
train_data_not_eyes = zeros(mida_not_eyes(4), width(hog));
for i=1:mida_not_eyes(4)
    if (mod(i, 500) == 0)
        i
    end
    image = NotEyesTrainData(:, :, :, i);
    imageGray = rgb2gray(image);
    train_data_not_eyes(i, :) = extractHOGFeatures(imageGray, 'CellSize',
cell);
end

train_data = [train_data_eyes, ones(mida_eyes(4), 1); train_data_not_eyes,
zeros(mida_not_eyes(4), 1)];

titulos = strings(width(hog), 1);
for i = 1:width(hog)
    titulos(i) = "Atr " + int2str(i);
end

titulos = [titulos; "Clase"];
titulos = transpose(titulos);


T = array2table(train_data, 'VariableNames', titulos);
```

Algoritmo para comprobar los modelos.

```
load EyesTestData.mat
load NotEyesTestData.mat

load CosineKNNModelKernel16.mat

cell = [16 16]

image = EyesTestData(:, :, :, 1);
imageGray = rgb2gray(image);
hog = extractHOGFeatures(imageGray, 'CellSize', cell);

mida_eyes = size(EyesTestData);
test_data_eyes = zeros(mida_eyes(4), width(hog));
for i=1:mida_eyes(4)
    image = EyesTestData(:, :, :, i);
    imageGray = rgb2gray(image);
    test_data_eyes(i, :) = extractHOGFeatures(imageGray, 'CellSize', cell);
end


mida_not_eyes = size(NotEyesTestData);
test_data_not_eyes = zeros(mida_not_eyes(4), width(hog));
for i=1:mida_not_eyes(4)
    if (mod(i, 500) == 0)
        i
```

```
    end
    image = NotEyesTestData(:, :, :, i);
    imageGray = rgb2gray(image);
    test_data_not_eyes(i, :) = extractHOGFeatures(imageGray, 'CellSize',
cell);
end

test_data = [test_data_eyes, ones(mida_eyes(4), 1); test_data_not_eyes,
zeros(mida_not_eyes(4), 1)];
T = array2table(test_data);

titulos = strings(width(hog), 1);
for i = 1: width(hog)
    titulos(i) = "Atr " + int2str(i);
end

titulos = [titulos; "Clase"];


T.Properties.VariableNames = titulos;

should = T.Clase;
yfit = CosineKNNModelKernel16.predictFcn(T);

confusionchart(confusionmat(should, yfit))
```

Programa de detección de ojos.

```
load CosineKNNModelKernel16.mat

webcamlist;
cam = webcam(1);


imagenOriginal = snapshot(cam);

imshow(imagenOriginal)
rect = getrect()

imagenSinRecortar = imagenOriginal(rect(2):rect(2)+rect(4),
rect(1):rect(1)+rect(3), :);

imshow(imagenSinRecortar)

imagenRecortada = imresize(imagenOriginal(rect(2):rect(2)+rect(4),
rect(1):rect(1)+rect(3), :), [113 513]);
imshow(imagenRecortada)

ejecutarPredictor(imagenRecortada, CosineKNNModelKernel16)
```

Función ejecutarPredictor()

```
function[prediction] = ejecutarPredictor(imagen, clasificador)

    imagenGris = rgb2gray(imagen);

    imshow(imagenGris)
```

```
    HOGFeatures = extractHOGFeatures(imagenGris, 'CellSize', [16 16]);
    HOGFeatures = [HOGFeatures 1];

    titulos = strings(width(HOGFeatures), 1);
    for i = 1: width(HOGFeatures)
        titulos(i) = "Atr " + int2str(i);
    end

    titulos(width(HOGFeatures)) = "Clase";

    tabla = array2table(HOGFeatures);
    tabla.Properties.VariableNames = titulos;

    prediction = single(clasificador.predictFcn(tabla));
end
```