Proyecto final Visión por Computador
Detección de Logos

Eneko Sabaté
Tomás Calaf
Grupo 11K

7/01/2024

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Index

1

Detección de logos

El objetivo de esta prática es crear una aplicación que sepa identificar una cierta empresa dado un logo de esa misma empresa. Concretamente, haremos una aplicación funcional para las siguientes 8 empresas: Apple, Blackberry, Cisco Systems, Daewoo Electronics, HP, IBM, Intel y Motorola. Más adelante hablaremos de la implementación de la clase "Unknown", que servirá para clasificar todos los logos no mencionados anteriormente.

En este informe se explicarán los pasos seguidos hasta llegar a la app resultado, pasando por los descriptores, clasificadores y experimentación realizada.

1.1 Descriptores

En este apartado describiremos los descriptores utilizados para el funcionamiento de nuestra app, es decir, las características extraídas de los logos con el fin de clasificarlas después.
Para el enfoque de esta práctica, vimos desde un principio que el uso de segmentación era muy complicado. Debido a que los logos en la mayoría de casos están compuestos por diferentes formas y letras, la extracción de descriptores a través de segmentación no creímos que fuera a resultarnos útil. Es por este motivo que saltamos directamente a la extracción de características locales. Estas nos permiten describir la imagen sin tener que segmentar la forma/objeto que queremos describir.

1.1.1 Histogramas de orientación

Dentro de las características locales, las mas fáciles son los histogramas, estos dan una representación de el espectro de una propiedad para una imagen. Debido a que los logos tienden a cambiar de color, i las fotos tienen iluminaciones muy diferentes, los histogramas de color no dan un buen resultado.
Los histogramas de orientación describen una imagen según la dirección del gradiente, esto permite ahorrarse la localización de los contornos.

Al extraer el histograma, hay que eliminar los píxeles de gradiente pequeño, ya que distorsionan el histograma. Además, ya hemos visto que lo que nos importa de los histogramas son los picos, por lo que un treshold es necesario para mejorar el descriptor. Para ello empleamos un treshold sobre el modulo del gradiente, donde todo píxel con gradiente inferior es obviado. Empezando en 40 y después de algunas pruebas lo dejamos en 60.

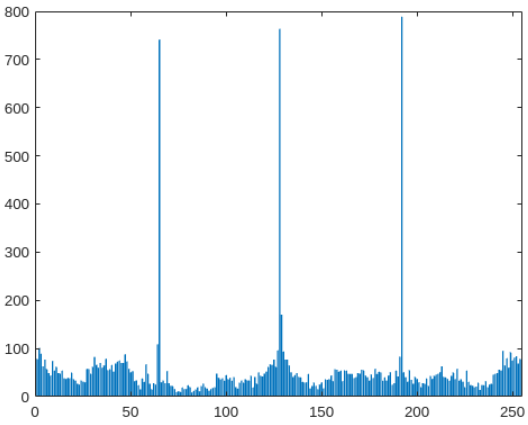
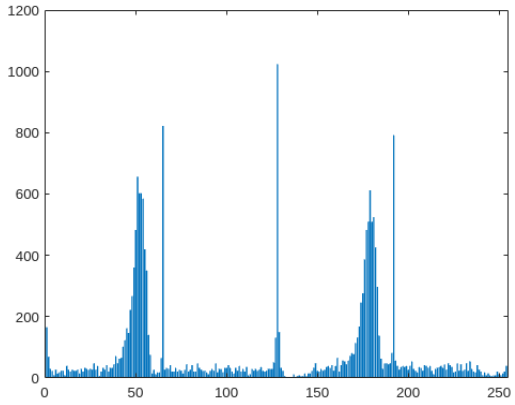


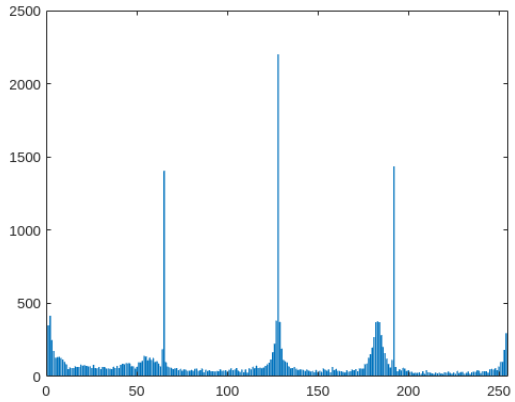
Figure 1.1: Histograma de orientación para apple

En la Figure 1.1 podemos ver el histograma de orientaciones de un logo de apple. Como se puede apreciar este histograma es bastante distintivo, con tres orientaciones bien marcadas.

La mayoría de logos concentran sus direcciones alrededor de los mismos picos, por lo que la distinción entre logos se basa mas en la forma del histograma, y en las variaciones que pueda haber, como se aprecia en el histograma de blackberry y hp(Figure 1.2).



(a) Blackberry.



(b) Hp

Figure 1.2: Histograma de orientacion

Este histograma sera el vector de características que usamos en el **classification learner**. En la sección 1.3 se explica su uso y resultados.

1.1.2 HOG

La mayor flaqueza de los histogramas de orientacion es que lo hacen para la imagen competa. Los Histogramas of Oriented Gradients dividen la imagen en celdas i se calcula los histogramas de orientacion de cada celda. Los histogramas se combinan para formar el vector de características. Este esta representado en la Figure 1.3 donde se puede apreciar el logo de apple.

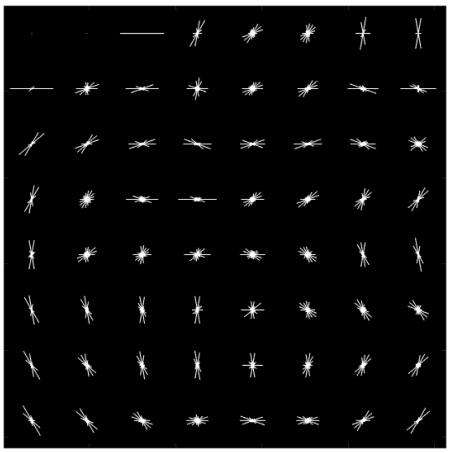


Figure 1.3: HOG para "data/Logos/Apple/1.jpg"

Para extraer HOG hay que determinar el tamaño de las celdas. sabemos que toas las imágenes miden 256 por 256. Cabe decir que cuanto mas pequeño sea el tamaño de la celda, mas grande sera el vector de características, por lo que peor rendimiento tendremos. La celda mínima con la que probamos es [16 16], los mejores resultados son con celdas [64 64]. Matlab proporciona una función *extractHOGfeatures* que devuelve el histograma, como argumentos le pasamos 'CellSize' = 32. Esta función también permite escoger el espectro de los histogramas, por defecto es de 0 a 180, pero también es posible de -180 a 180. En nuestro caso el mejor resultado es con la opción por defecto.

1.1.3 SIFT

SIFT, o Scale-Invariant Feature Transform, es un algoritmo que extrae descriptores basados en keypoints. Es un algoritmo muy robusto ya que es invariante tanto al escalado como a la rotación de las imágenes. Este descriptor resulta ideal para la detección de logos, ya que los logos siempre van a ser (relativamente) iguales, el problema es saber detectarlos, por lo que la invarianza a la escala y a la rotación es idónea.

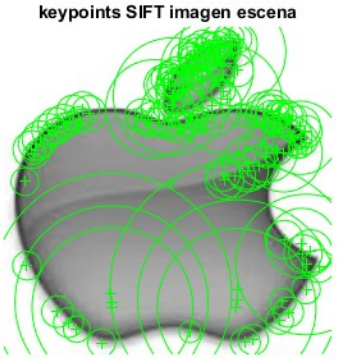


Figure 1.4: Características SIFT para la imagen "data/Logos/Apple/6.jpg"

Para extraer las características SIFT de una sola imagen hemos desarrollado la función SIFTExtractor. En esta función nos valemos de la función ya implementada en el paquete de CV de MatLab, detectSiftTFeatures(), para detectar los keypoints de una imagen. La hemos utilizado con las opciones EdgeThreshold=15 y ContrastThreshold=0.01 con el fin de extraer más keypoints. También usamos la funcion extractFeatures del paquete de CV de MatLab para extraer los histogramas alrededor de cada keypoint detectado anteriormente. Por último, hemos implementado la aplicación extractLogoSIFT, que

se encarga de extraer los descriptores SIFT del dataset de imágenes de training y testing, guardándolos en los archivos "testSIFTDescriptors.mat" y "trainingSIFTDescriptors.mat" en la carpeta "data". La aplicación extractLogoSIFT hace uso de extractSIFTSetImages, que hace lo mismo que SIFTExtractor pero, en vez de para una imagen, para un set de imágenes. Todo el código implementado por nosotros mencionado se puede encontrar en el 2.1 del Anexo A.

1.1.4 SURF

SURF, o Speeded-Up Robust Features, es un algoritmo basado en SIFT, por lo que también utilizará keypoints para la detección de características. Este algoritmo es igual de robusto que el SIFT, pero su funcionamiento a la hora de seleccionar características es distinto, por lo que vimos viable usar descriptores SURF para usar una mezcla de los dos en los clasificadores.

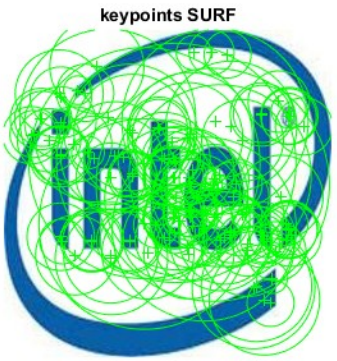


Figure 1.5: Características SURF para la imagen "data/Logos/Intel/4.jpg"

Las funciones y aplicaciones implementadas para el SURF son simétricas a las del SIFT: SURFExtractor, extractSURFSetImages y extractLogoSURF. El código de estos archivos es idéntico execptuando algunas llamadas a funciones, como es en el caso de SURFExtractor, que se llama a la función ya implementada detectSurfFeatures(). detectSurfFeatures es llamada con la opción numScaleLevels=5 con el fin de detectar más keypoints.

1.2 Clasificadores

1.2.1 Distancia Euclidiana

Utilizamos la distancia euclidiana entre dos histogramas como clasificador de las métricas SIFT y SURF. El funcionamiento es el siguiente: Una vez calculada la distancia entre dos histogramas, si la distancia entre las dos es menor que un cierto threshold entonces consideraremos que ha habido un match. La categoría que mas matches haga con la imagen de test será nuestra predicción.

Existe un problema con lo mencionado en el anterior parágrafo. Si existe una categoría en la que se hayan detectado más keypoints que otra, tendrá más histogramas, por lo que es más fácil que se den más falsos positivos y por ende que la predicción sea incorrecta. Para solucionar esto dividimos el número de matchings entre el número de histogramas total de la categoría de logo. Por otro lado, a la hora de clasificar las imagenes mezclaremos los resultados de la clasificación del SIFT y de SURF en un ratio de 7:3. La razón de esto se explicará en la experimentación.

Hemos implementado la función computeMatchingEuclidean con el objetivo de que calcule el número de matchings entre una imagen test y un set de imagenes training. Esta función usa a su vez la función ya implementada matchFeatures, la cual devuelve los matchings de dos imagenes según la distancia

Figure 1.6: Matching de keypoints usando la distancia euclidiana sobre "data/Logos/Intel/2.jpg" y "data/Logos/Intel/4.jpg"

euclidiana. Las opciones utilizadas en matchfeatures son Method=Exhaustive, MatchThreshold=2 y Unique=True, para conseguir unos matches lo más precisos posibles. También se han implementado las funciones computeNumHistograms y matchLogos, para contar el numero de histogramas en un set de features y para realizar lo mismo que computeMatchingEuclidean pero para muchas imagenes de test respectivamente.

1.3 Experimentación y Resultados

1.3.1 Test Set

El test set es el set de las imagenes que no se han utilizado para entrenar al modelo y que por lo tanto solo se utilizan para el testing. Este set se ha decidido en la aplicación storeImages, donde se define un set de imagenes para training y otro para testing, en un ratio de 85:15.

Una vez extraidos los histogramas de orientacion y los HOG para todas las imagenes, usamos el **classification learner** para entrenar dos modelos.

En la Figure 1.7 podemos observar los resultados para los modelos basados en histogramas de orientacion. Entre los modelos probados, SVM quadratica y cubica, Fine KNN y Bagged Trees, el mejor modelo es la SVM quadratica con una precision en validacion del 74.91% y del 83.33% en test. En la Figure 1.11 y Figure 1.12 se pueden observar los resultados del mejor modelo entrenado.

Favorite	Model Number	Model Type	Status	Accuracy (Validation)	Total Cost (Validati...	Accuracy (Test)	Total Cost (Te...
☐	2	SVM	Tested	74.91 %	137	83.33 %	16
☐	3	SVM	Tested	74.91 %	137	81.25 %	18
☐	4	KNN	Tested	74.91 %	137	80.21 %	19
☐	5	Ensemble	Tested	70.70 %	160	68.75 %	30

Figure 1.7: Resultados classification learner Histogramas de orientación

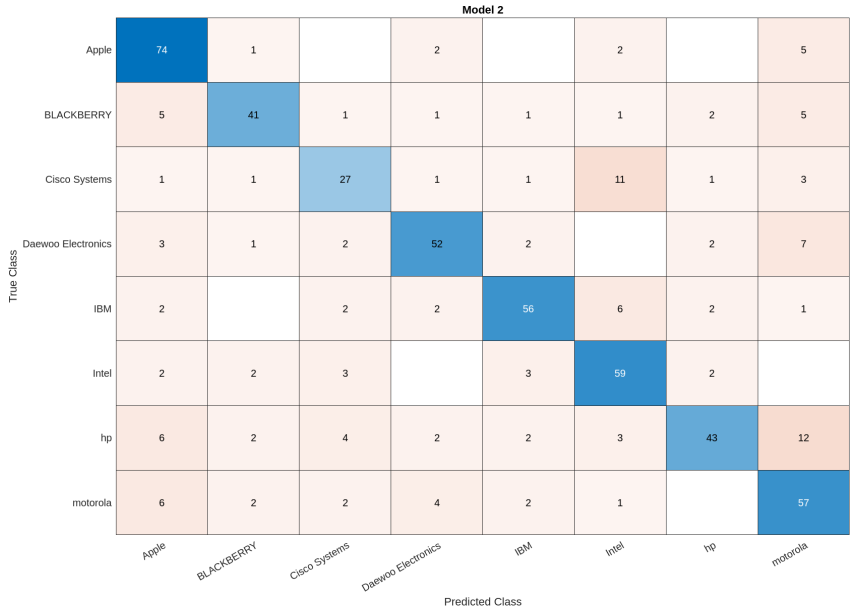


Figure 1.8: CM Histogramas de orientación Validation

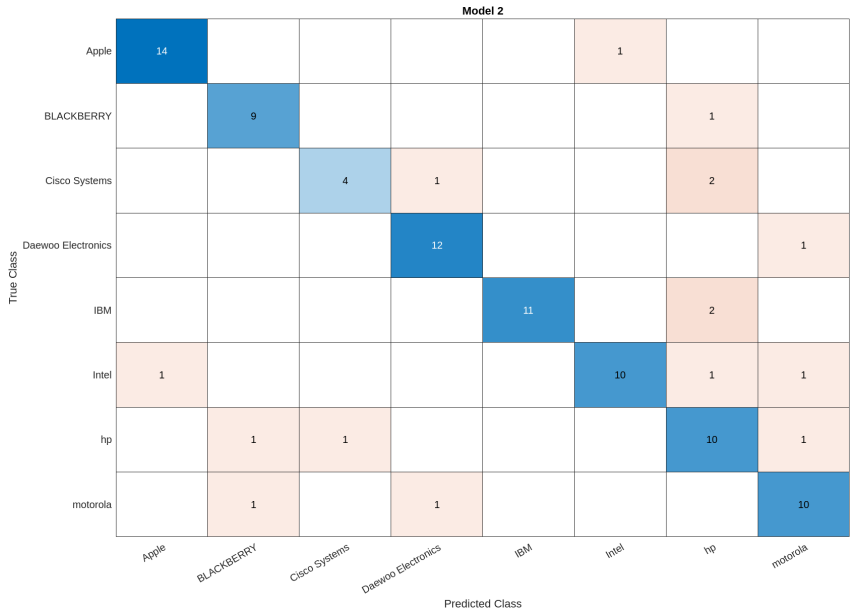


Figure 1.9: CM Histogramas de orientación Test

En cuanto al modelo de HOG, en la Figure 1.16 podemos observar los resultados. Entre la SVM cubica, SVM gaussiana, Bagged Trees y Fine KNN el mejor modelo vuelve a ser la SVM cubica, con una precisión en validación del 80.59% i del 84.4% en test. Este resultado es mejor que para los histogramas de orientacion, cosa razonable ya que HOG es una mejora de estos. También se puede apreciar en las matrices de confusión como el resultado mejora levemente, aunque siguen habiendo errores con todos los logos.

Favorite	Model Number	Model Type	Status	Accuracy (Validation)	Total Cost (Validati...	Accuracy (Test)	Total Cost (Te...
☐	2.3	SVM	Tested	80.59 %	106	84.38 %	15
☐	2.5	SVM	Tested	80.59 %	106	81.25 %	18
☐	3	Ensemble	Tested	75.27 %	135	72.92 %	26
☐	4	KNN	Tested	77.47 %	123	80.21 %	19

Figure 1.10: Resultados classification learner HOG

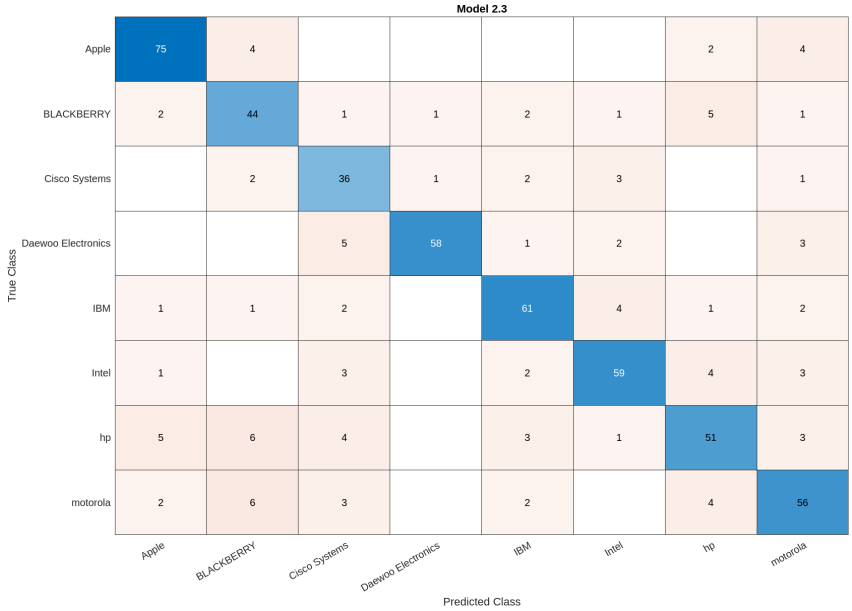


Figure 1.11: CM HOG Validation

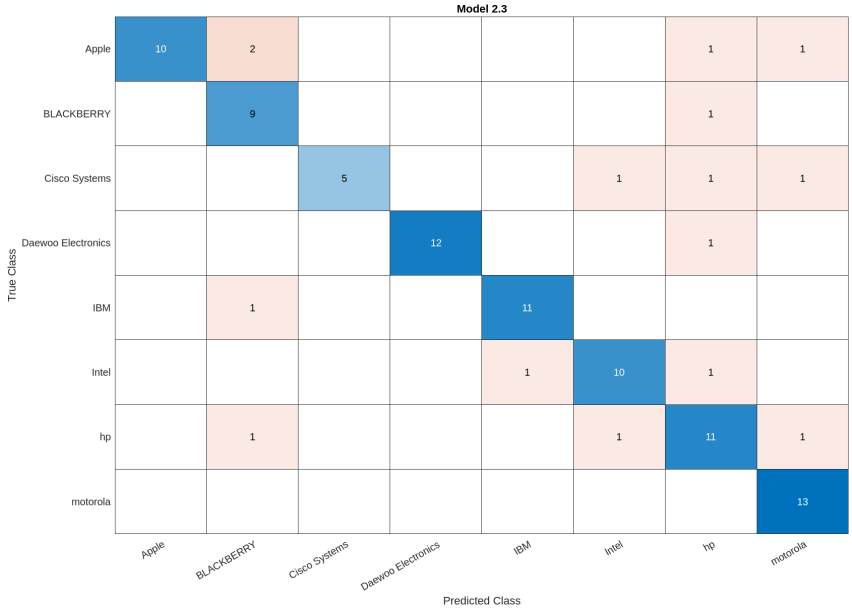


Figure 1.12: CM HOG Test

Vistos estos dos resultados, el mejor modelo es el que esta basado en HOG. Pero también había la posibilidad de utilizar los dos en el mismo modelo. Para ello, simplemente concatenamos el histograma

de orientacion de la imagen con su HOG, y formamos la tabla de imágenes. Seguimos el mismo procedimiento que anteriormente, y obtenemos los siguientes resultados.

Model Number	Model Type	Status	Accuracy (Validation)	Total Cost (Validati...	Accuracy (Test)	Total Cost (Te...
1	Tree	Tested	55.49 %	243	60.42 %	38
2	SVM	Tested	81.87 %	99	81.25 %	18
3	SVM	Tested	82.60 %	95	81.25 %	18
5	KNN	Tested	79.12 %	114	80.21 %	19
6	Ensemble	Tested	76.74 %	127	69.79 %	29

Figure 1.13: Resultados classification learner HOG + Histogramas de orientacion

Como se puede ver la precisión en validación aumenta levemente con respecto a HOG hasta el 82.60% con la SVM cubica, mientras que en el test baja al 81.25%. Esto seguramente es debido a que nuestra base de datos es limitada, por lo que las imágenes de test son pocas i aleatorias. Entonces podemos concluir que la diferencia entre HOG y este es muy poca, y no podemos determinar cual es mejor y peor. Es por eso que por motivos de rendimiento usaremos de momento solo HOG. En la Figure 1.14 i la Figure 1.15 podemos ver las matrices de confusión, donde tampoco hay gran variación respecto a HOG.

Ahora podemos exportar el modelo entrenado en el classification learner al workspace, donde podemos hacer perdiciones con la función *predictFcn(hog)*

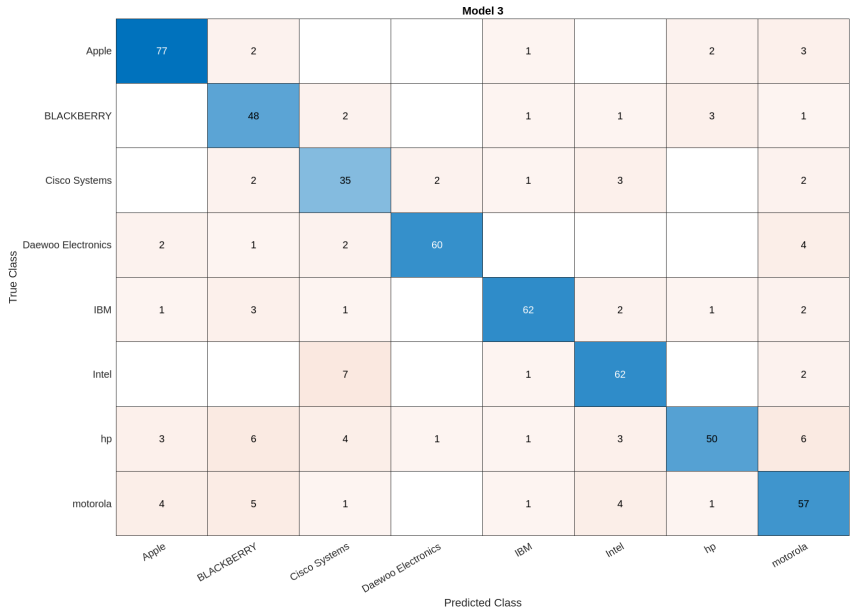


Figure 1.14: CM HOG + Histogramas de orientacion Validation

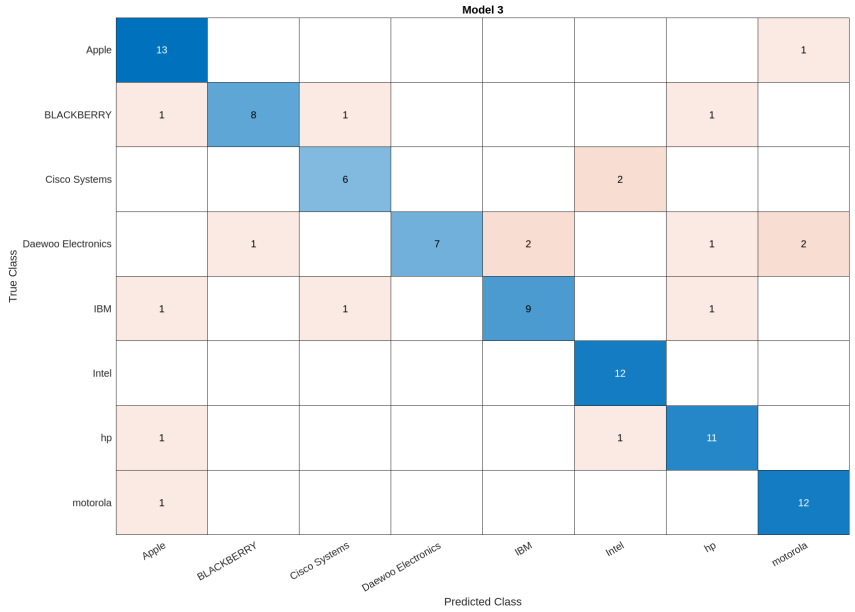
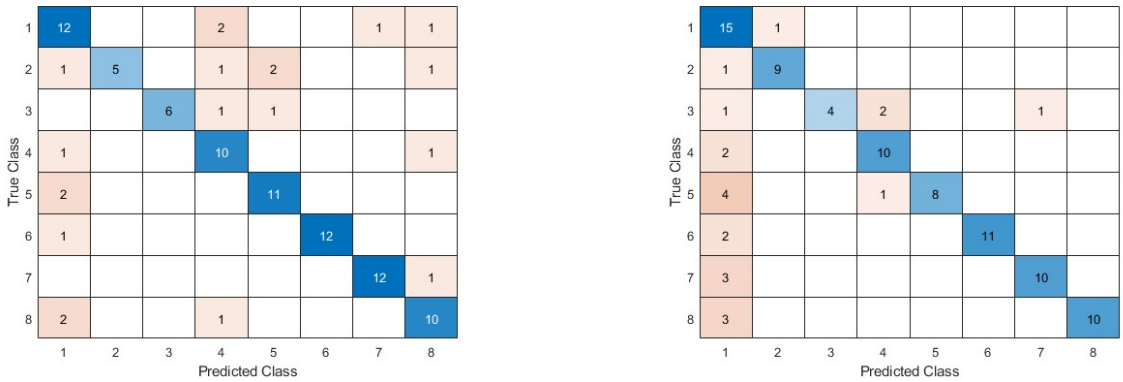


Figure 1.15: CM HOG + Histogramas de orientacion Test

En la Figure 1.16 podemos encotrarnos las matrices de confusión de las clasificaciones SIFT y SURF. Las clasificaciones son relativamente buenas, superando el SIFT al SURF por poco, teniendo una precisión de 79.59% y 78.57% respectivamente.



(a) CM SIFT.

(b) CM SURF.

Figure 1.16: Test separate.

Una versión mejorada a las anteriores dos es si juntamos los dos clasificadores para hacer uno nuevo, uno que utilice SIFT y SURF. Después de varios experimentos, concluimos que la mejor solución es la que utiliza 7 partes del resultado de la clasificación de las características SIFT con 3 partes de SURF. La matriz de confusión de este nuevo modelo es visible en la figura 1.5, con una precisión de 84.69%.

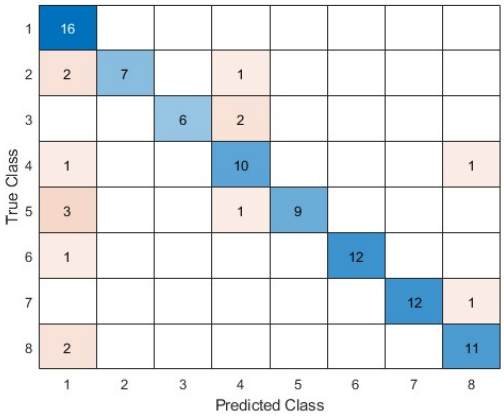


Figure 1.17: CM del test conjunto.

Con estos resultados, hemos decidido que usaremos SIFT y SURF para hacer la predicción, ya que tiene mejor precisión. Cabe decir que el margen es pequeño, y que los histogramas de orientacion y HOG han funcionado mejor de lo que esperábamos.

1.3.2 Complex Set

Llamamos complex set al conjunto de imágenes con fondo analíticamente complicado a la hora de extraer el logo de ella. Estas imágenes se pueden encontrar en "data/Logos/complex_background". Como podemos ver en las siguientes figuras los resultados no son buenos, solamente con un 50% de precisión en SIFT y SURF y un 37.5% con HOG. Aún así, se trata de imágenes muy complicadas de analizar por la cantidad de detalles que se pueden encontrar en el fondo. Además, no es posible realizar muchas deducciones dado el pequeño dataset.

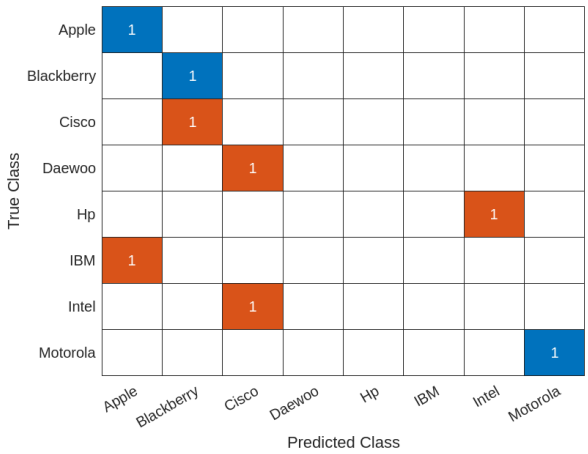


Figure 1.18: CM del test complex HOG

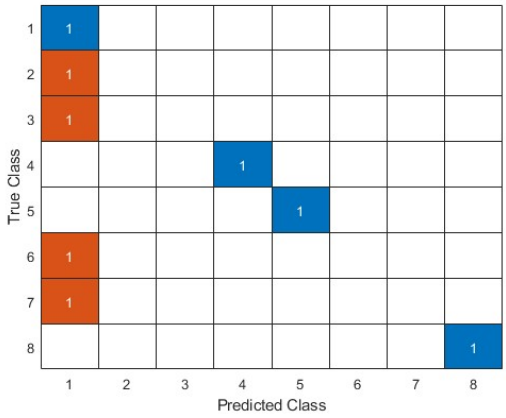


Figure 1.19: CM del test complex SIFT y SURF

1.3.3 Unknown

En las siguientes experimentaciones hemos añadido la clase 0 "Unknown", que como el propio nombre indica representa todo logo no contemplado.

En el modelo HOG, simplemente es añadir una nueva clase al dataset, y volver a entrenar el modelo. En este caso el mejor modelo (SVM cubica) da una precisión del 76.1%, lo cual no esta mal ya que sin unknown era de 84.38%. Como se puede apreciar en la confusion matrix la cantidad de predicciones unknown no es demasiado alta, que era el mayor problema que esperábamos.

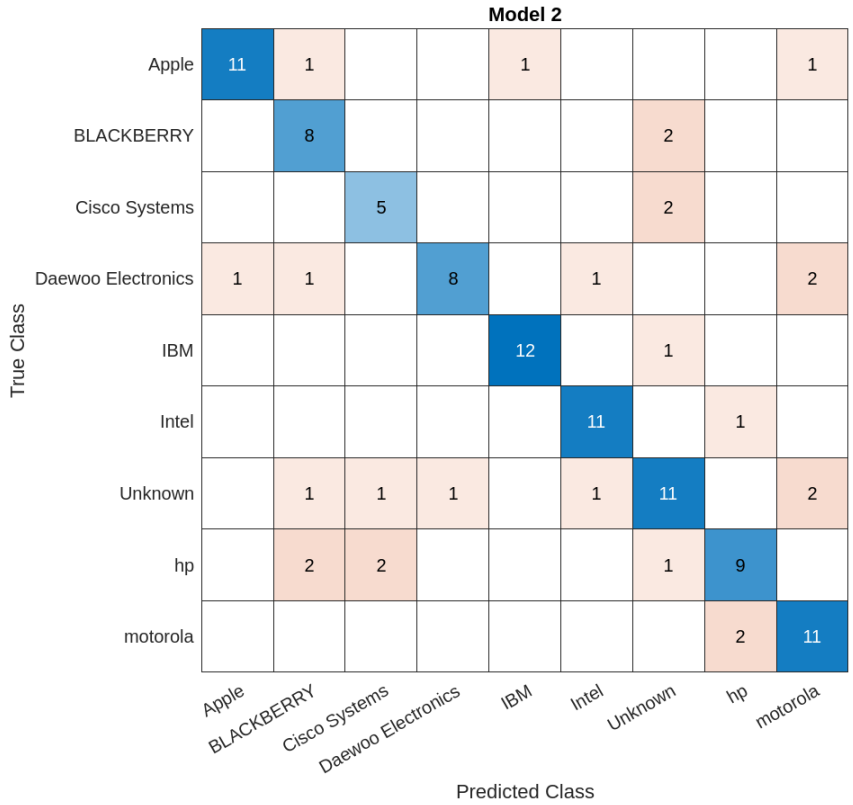


Figure 1.20: CM del test set + Unknown HOG

En SIFT y SURF para hacer esta clase posible hemos implementado un threshold que, cuando el

número de matchings máximo obtenido es demasiado bajo, se clasifica en la clase 0. Después de realizar muchos experimentos, hemos decidido el valor del threshold: 0.08. Aún así, como se puede ver el la matriz de confusión, el resultado no es muy bueno, con solo un 56% de aciertos. Esto es debido a que un montón de imágenes no Unknown tienen un número muy bajo de matchings, por lo que son clasificadas en la categoría 0.

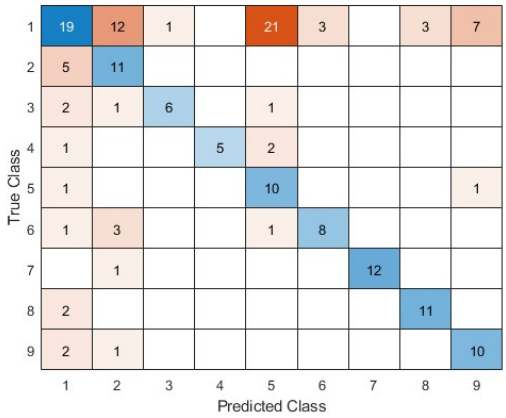


Figure 1.21: CM del test set + Unknown SIFT y SURF

2

Anexo A: Código utilizado

2.1 Código descriptores

```
%% HOGExtractor.m
%% Funcion para cargar todas las caracteristicas HOG de las imagenes en una tabla para e
function [ T ] = getDataSet()
    folder = 'logos2/Apple/';
    apple = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/hp/';
    hp = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Intel/';
    intel = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/BLACKBERRY/';
    blackberry = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Cisco Systems/';
    cisco = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/IBM/';
    ibm = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Daewoo Electronics/';
    daewoo = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/motorola/';
    motorola = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Unknown/';
    unknown = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    dataset = [];
    class = [];
    [train_data, clase] = getTrainData(apple);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(hp);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(intel);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(blackberry);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(cisco);
    dataset = [dataset;train_data];
    class = [class; clase];
```

```
    [train_data, clase] = getTrainData(ibm);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(daewoo);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(motorola);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(unknown);
    dataset = [dataset;train_data];
    class = [class; clase];
    T = array2table(dataset);

    [f,c] = size(dataset);
    titulos = strings(c, 1);
    for i = 1:c
        titulos(i) = "Atr hog " + int2str(i);
    end

    titulos = transpose(titulos);
    allVars = 1:width(T);
    names = append("dataset",string(allVars));
    T = renamevars(T,names, titulos);
    T = addvars(T,class);
end

function [ train_data, clase ] = getTrainData(images)
    cell = [64 64];
    numImages = numel(images.Files);
    train_data = [];
    clase = strings(numImages, 1);
    names = split(images.Folders,"/");
    name = names(length(names));
    for i=1:numImages
        im = readimage(images, i);
        im = processImage(im);
        hog = extractHOGFeatures(im, 'CellSize', cell);
        train_data = [train_data; hog];
        clase(i) = name;
    end
end

function [ img ] = processImage(im)
    img = rgb2gray(im);
    img = imresize(img, [256 256]);
end



%% OrientationExtractor.m
%% Funcion para cargar todos los histogramas de orientacion de las imagenes en una tabla
function [ T ] = getDataSet()
    folder = 'logos2/Apple/';
    apple = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/hp/';
```

```
    hp = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Intel/';
    intel = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/BLACKBERRY/';
    blackberry = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Cisco Systems/';
    cisco = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/IBM/';
    ibm = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Daewoo Electronics/';
    daewoo = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/motorola/';
    motorola = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    folder = 'logos2/Unknown/';
    unknown = imageDatastore(fullfile(folder, '*.jpg'), 'LabelSource','foldernames');
    dataset = [];
    class = [];
    [train_data, clase] = getTrainData(apple);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(hp);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(intel);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(blackberry);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(cisco);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(ibm);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(daewoo);
    dataset = [dataset;train_data];
    class = [class; clase];
    [train_data, clase] = getTrainData(motorola);
    dataset = [dataset;train_data];
    class = [class; clase];
    T = array2table(dataset);

    [f,c] = size(dataset);
    titulos = strings(c, 1);
    for i = 1:c
        titulos(i) = "Atr hist " + int2str(i);
    end

    titulos = transpose(titulos);
    allVars = 1:width(T);
    names = append("dataset",string(allVars));
    T = renamevars(T,names, titulos);
    T = addvars(T,class);
end
```

```
function [ train_data, clase ] = getTrainData(images)
    treshold = 60;
    numImages = numel(images.Files);
    train_data = [];
    clase = strings(numImages, 1);
    names = split(images.Folders,"/");
    name = names(length(names));
    for i=1:numImages
        im = readimage(images, i);
        im = processImage(im);
        features = HistOrientations(im, treshold);
        train_data = [train_data; features];
        clase(i) = name;
    end
end

function [ h ] = HistOrientations(im, treshold)
    im = double(im);
    sob = fspecial('sobel')/4;
    gx = imfilter(im, sob); % Gradient horitzontal
    gy = imfilter(im, sob'); % Gradient vertical
    alfa = atan2(gy,gx);
    dir = uint8(254*(alfa+pi)/2/pi);
    mod = sqrt(gx.^2+gy.^2);
    mask = (mod < treshold);
    dir(mask) = 255;
    h = imhist(dir);
    h = h';
end

function [ img ] = processImage(im)
    img = imresize(im, [256 256]);
    img = rgb2gray(img);
    img = double(img);
end



%% SIFTExtractor.m
%% Funci n para cargar todas las imagenes con una extension
%%determinada de una carpeta
function [features, featureMetrics, location] = SIFTExtractor(img)
    gray = rgb2gray(img);

    kp_img = detectSIFTFeatures(gray, "EdgeThreshold", 15, "ContrastThreshold", 0.01);
    location = kp_img.Location;
    featureMetrics = kp_img.Metric;

    features = extractFeatures(gray, kp_img, "Method", "SIFT");
end



%% extractSIFTSetImages.m
%% Funci n para cargar todas las imagenes con una extension determinada
```

```
%% de una carpeta
function featuresCell = extractSIFTSetImages(imds)
    numImages = numel(imds.Files);
    featuresCell = cell(numImages, 1);

    for i = 1:numImages
        im = readimage(imds, i);
        featuresCell{i} = SIFTExtractor(im);
    end
end
```

```
%% extractLogoSIFT.m
%% Extrae los descriptores SIFT de "testDataset.mat" y "trainingDataset.mat"

%% Carga de Im genes
load("./data/trainingDataset.mat");
load("./data/testDataset.mat");

trainingSIFTDescriptors = cell(numel(trainingDataset), 1);
testSIFTDescriptors = cell(numel(testDataset), 1);

for i = 1:numel(testDataset)
    trainingSIFTDescriptors{i} = extractSIFTSetImages(trainingDataset{i});
    testSIFTDescriptors{i} = extractSIFTSetImages(testDataset{i});
end

save(".\data\testSIFTDescriptors.mat", "testSIFTDescriptors");
save(".\data\trainingSIFTDescriptors.mat", "trainingSIFTDescriptors");
```

```
%% SURFExtractor.m
%% Funci n para extraer las caracteristicas SURF de una imagen
function [features, featureMetrics, location] = SURFExtractor(img)
    gray = rgb2gray(img);

    kp_img = detectSURFFeatures(gray, "NumScaleLevels", 5);
    location = kp_img.Location;
    featureMetrics = kp_img.Metric;

    features = extractFeatures(gray, kp_img, "Method", "SURF");
end
```

```
%% extractSURFSetImages.m
%% Funci n para extraer las caracter sticas de una
%% estructura imageDatastore
function featuresCell = extractSURFSetImages(imds)
    numImages = numel(imds.Files);
    featuresCell = cell(numImages, 1);

    for i = 1:numImages
```

```
        im = readimage(imds, i);
        featuresCell{i} = SURFExtractor(im);
    end
end
```

```
%% extractLogoSURF.m
%% Extrae los descriptores SURF de "testDataset.mat" y "trainingDataset.mat"

%% Carga de Im genes
load("./data/trainingDataset.mat");
load("./data/testDataset.mat");

testSURFDescriptors = cell(numel(testDataset), 1);
trainingSURFDescriptors = cell(numel(trainingDataset), 1);

for i = 1:numel(testDataset)
    testSURFDescriptors{i} = extractSURFSetImages(testDataset{i});
    trainingSURFDescriptors{i} = extractSURFSetImages(trainingDataset{i});
end

save(".\data\testSURFDescriptors.mat", "testSURFDescriptors");
save(".\data\trainingSURFDescriptors.mat", "trainingSURFDescriptors");
```

2.2 Código clasificadores

```
%% computeMatchingsEuclidean.m
%% Calcula el numero de matchings de histogramas entre
%% dos sets de imagenes
function numMatchings = computeMatchingsEuclidean(testImg, trainingImgs)
    numMatchings = 0;

    for i = 1:numel(trainingImgs)
        trainImg = trainingImgs{i};
        pairs = matchFeatures(testImg, trainImg, "Method","Exhaustive" ,"MatchThreshold"

        [nrows, ~] = size(pairs);
        numMatchings = numMatchings + nrows;
    end
end
```

```
%% computeNumHistograms.m
%% Cuenta el n mero de histogramas de un set de descriptores SIFT o SURF
function numHistograms = computeNumHistograms(trainingImgs)
    numHistograms = 0;

    for i = 1:numel(trainingImgs)
        trainImg = trainingImgs{i};
```

```
        [nrows, ~] = size(trainImg);
        numHistograms = numHistograms + nrows;
    end
end
```

```
%% matchLogos.m
%% Funcion que devuelve el numero de matchings/numero de historgramas
%% por cada imagen de test
function [group, matchings] = matchLogos(trainingDescriptors, testDescriptors)
    logo_types = ["Apple", "BLACKBERRY", "Cisco Systems", "Daewoo Electronics", "hp", "I
    numLogos = numel(logo_types);
    numImgsTest = sum(cellfun(@numel, testDescriptors));
    group = zeros(numImgsTest, 1);
    matchings = cell(numImgsTest, 1);

    counter = 1;
    for i = 1:numLogos
        c = testDescriptors{i};
        for j = 1:numel(c)
            histImg = c{j};
            group(counter) = i;

            numMatchingsLogo = zeros(numLogos, 1);
            for k = 1:numel(logo_types)
                numMatchingsLogo(k) = computeMatchingsEuclidean(histImg, trainingDescrip
                numMatchingsLogo(k) = numMatchingsLogo(k) / computeNumHistograms(trainin
            end

            matchings{counter} = numMatchingsLogo;
            %[m, index] = max(numMatchingsLogo);
            %grouphat((i - 1)*numImgsTest + j) = index;

            counter = counter + 1;
        end
    end
end
```

2.3 Código experimentación

```
%% matchLogos.m
%% Funcion que hace la preddicion para una imagen con el modelo entrenado de HOG
function [ pred, scores ] = predictHOG(im)
    hog = extractHOGFeatures(im, 'CellSize', [64 64]);
    titulos = strings(width(hog), 1);
    for i = 1:width(hog)
        titulos(i) = "Atr hog " + int2str(i);
    end

    titulos = transpose(titulos);
```

```
    T = array2table(hog, 'VariableNames', titulos);
    load('modelHOG.mat');
    [pred, scores] = trainedModelHOG.predictFcn(T);
end

function [ img ] = process(im)
    img = rgb2gray(im);
    img = imresize(img, [256 256]);
end



%% testing.m
% Testing sobre el dataset "logos"

%% Carga descriptores e imagen input
logo_types = ["Apple", "BLACKBERRY", "Cisco Systems", "Daewoo Electronics", "hp", "IBM",
load("./data/testSIFTDescriptors.mat");
load("./data/trainingSIFTDescriptors.mat");
load("./data/testComplexSIFTDescriptors.mat");
load("./data/testUnknownSIFTDescriptors.mat");

load("./data/testSURFDescriptors.mat");
load("./data/trainingSURFDescriptors.mat");
load("./data/testComplexSURFDescriptors.mat");
load("./data/testUnknownSURFDescriptors.mat");


%% Test testSet

%[groupSIFT, matchingSIFT] = matchLogos(trainingSIFTDescriptors, testSIFTDescriptors);
%[groupSURF, matchingSURF] = matchLogos(trainingSURFDescriptors, testSURFDescriptors);

%% Test complexSet

%[groupSIFT, matchingSIFT] = matchLogos(trainingSIFTDescriptors, testComplexSIFTDescrip
%[groupSURF, matchingSURF] = matchLogos(trainingSURFDescriptors, testComplexSURFDescrip

%% Test testSet + Unknown

[groupSIFT, matchingSIFT] = matchLogos(trainingSIFTDescriptors, testSIFTDescriptors);
[groupSURF, matchingSURF] = matchLogos(trainingSURFDescriptors, testSURFDescriptors);

matchingSIFTUnknown = cell(numel(testUnknownSIFTDescriptors), 1);
matchingSURFUnknown = cell(numel(testUnknownSURFDescriptors), 1);
groupSIFTUnknown = zeros(numel(testUnknownSURFDescriptors), 1);
for j = 1:numel(testUnknownSURFDescriptors)
    inputFeaturesSIFT = testUnknownSIFTDescriptors{j};
    inputFeaturesSURF = testUnknownSURFDescriptors{j};
    numMatchingsLogoSIFT = zeros(numel(logo_types), 1);
    numMatchingsLogoSURF = zeros(numel(logo_types), 1);
    for i = 1:numel(logo_types)
        numMatchingsLogoSIFT(i) = computeMatchingsEuclidean(inputFeaturesSIFT, trainingS
        numMatchingsLogoSIFT(i) = numMatchingsLogoSIFT(i) / computeNumHistograms(trainin
```

```
        numMatchingsLogoSURF(i) = computeMatchingsEuclidean(inputFeaturesSURF, trainingS
        numMatchingsLogoSURF(i) = numMatchingsLogoSURF(i) / computeNumHistograms(trainin
    end
    matchingSIFTUnknown{j} = numMatchingsLogoSIFT;
    matchingSURFUnknown{j} = numMatchingsLogoSURF;
end

groupSIFT = [groupSIFT; groupSIFTUnknown];
matchingSIFT = [matchingSIFT; matchingSIFTUnknown];
matchingSURF = [matchingSURF; matchingSURFUnknown];

numTestImgs = numel(matchingSIFT);
grouphat = zeros(numTestImgs, 1);
for i = 1:numTestImgs
    testMatchSIFT = matchingSIFT{i};
    testMatchSURF = matchingSURF{i};

    matching = testMatchSIFT*0.7 + testMatchSURF*0.3;
    [m, index] = max(matching)

    if m < 0.08
        grouphat(i) = 0;
    else
        grouphat(i) = index;
    end
end

C = confusionmat(groupSIFT,grouphat);
figure, confusionchart(C);

accuracy =  sum(diag(C)) / sum(C(:))
```

2.4 Código app

```
%% app.m
%% Practica final VC: Detecci n de Logos

%% Carga descriptores e imagen input
logo_types = ["Apple", "BLACKBERRY", "Cisco Systems", "Daewoo Electronics", "hp", "IBM",
load("./data/trainingSIFTDescriptors.mat");
load("./data/trainingSURFDescriptors.mat");

im = imread("data\logos\Unknown\5.jpg");
figure, imshow(im);

%% Extract descriptors from input image
[inputFeaturesSIFT, ~, ~] = SIFTExtractor(im);
[inputFeaturesSURF, ~, ~] = SURFExtractor(im);

%%
matchSIFT = zeros(8, 1);
matchSURF = zeros(8, 1);
```

```
for i = 1:numel(logo_types)
    matchSIFT(i) = computeMatchingsEuclidean(inputFeaturesSIFT, trainingSIFTDescriptors{
    matchSIFT(i) = matchSIFT(i) / computeNumHistograms(trainingSIFTDescriptors{i});

    matchSURF(i) = computeMatchingsEuclidean(inputFeaturesSURF, trainingSURFDescriptors{
    matchSURF(i) = matchSURF(i) / computeNumHistograms(trainingSURFDescriptors{i});
end

matchings = matchSIFT*0.7 + matchSURF*0.3;

[m, i] = max(matchings);

%if m < X

disp(logo_types(i));
```