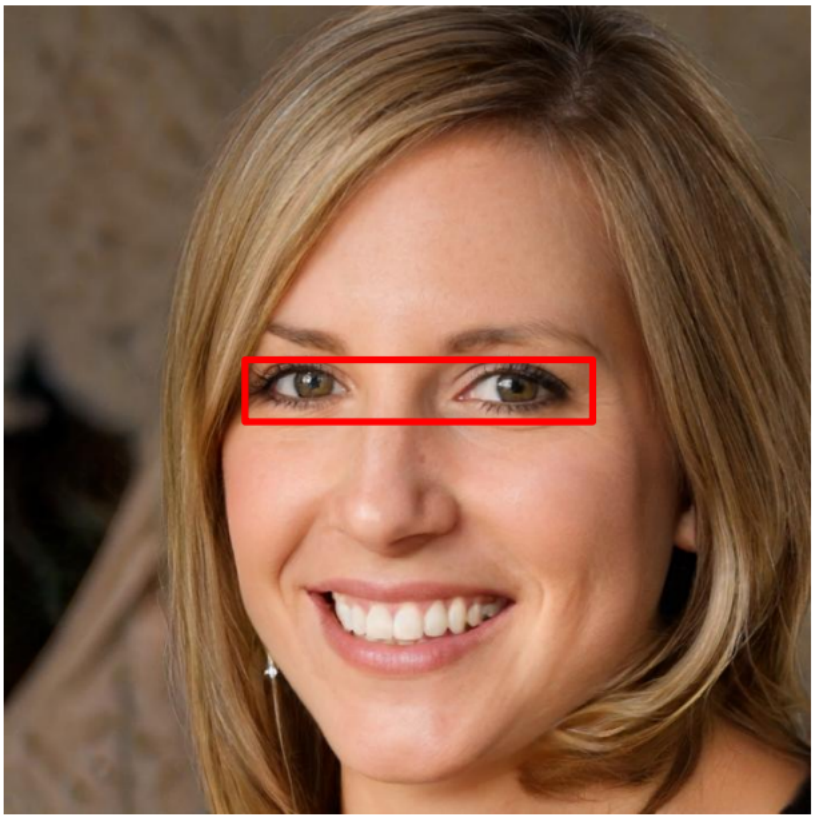
Short Project - *Checkpoint*
Visió per Computador

Leo Arriola Meikle
Zixuan Sun

18 de gener de 2022



Resum

En aquesta entrega del *Short Project*, preprocessarem les imatges proporcionades classificant-les de manera adequada i explorarem els diferents descriptors, principalment les `HOG Features`, amb la finalitat d'entrenar un classificador, en el nostre cas mitjançant `fitcsvm()` (*Support Vector Machines*), perquè classifiqui si hi ha ulls a la imatge.

Índex

1 Introducció

En aquest projecte implementarem un model de classificació de mirades mitjançant amb un sistema de visió per computador amb l'extracció de `HOG Features` i un model de classificació basat en una màquina de vector de suport (***S**upport-**V**ector **M**achines*).

Per aquesta primera entrega del projecte el sistema serà capaç de classificar i ens indicarà si una imatge té ulls o no. A l'entrega final, el sistema serà capaç de detectar la mirada i ho enquadrarà amb un rectangle en vermell. Finalment, també podrà mesurar l'angle horitzontal de la mirada basant-se en la posició de l'iris.

Per a aquest projecte, per fer la base de dades d'imatges hem extret imatges com se'ns ha indicat amb la pàgina `ThisPersonDoesNotExist.com`. Aquesta pàgina genera cares de persones amb un enquadrament de tipus bust, i ho fa gràcies amb una *Xarxa Generativa Antagònica* (o *GAN* de l'anglès). Un cop hem obtingut un conjunt prou gran d'imatges les hem dividit en imatges de no ulls i imatges d'ulls i hem procedit a crear els datasets adequats. Finalment, hem entrenat el nostre classificador i hem avaluat aquest amb el dataset de test per veure els resultats que produeix utilitzant algunes mètriques simples com la matriu de confusió, la proporció d'encerts global i similars.

Figura 1: Exemples d'imatges de la base de dades

2 Preprocessament de les imatges

Retallat d'imatges

Per a poder obtenir la base de dades d'imatges amb ulls i no ulls vam optar per a fer un script en *Python*. Vam dividir les imatges en requadres de mida similar a la *bounding box* dels ulls, d'aquesta manera vam evitar haver de redimensionar després.

La mida de la *bounding box* dels ulls l'hem establert una mica a ull sense ser massa gran, per estalviar temps de computació, ni massa petit per donar un marge a la posició dels ulls (ja que més o menys sempre estan a la mateixa posició dins de la imatge).

Amb aquesta mida de la *bounding box* (que concretament és de 450 d'amplada per 90 píxels d'altura) vam obtenir una proporció 1:21, és a dir, per a cada imatge amb ulls hi ha 21 sense ulls. En total, vam processar un total de 653 imatges d'ulls i un total de 12.502 de no-ulls.

El cos del codi del *script* en *Python* per retallar les imatges és el següent:

```
# Per a cada imatge de la BD
for img in imgList:
  x_i = y_i = 0
  x_f = 450; y_f = 90

  # Podem extreure els seguents 'trossos' de mida 450x90
  for i in range(1,22):
     orImg = cv2.imread('./images/' + img)
     if i % 2 == 0:
        x_i = 450; x_f = 900
        y_i += 90; y_f += 90
     else:
        x_i = 0; x_f = 450
     if i == 10:
        slc = orImg[440:530,287:737]
        cv2.imwrite(outImgPathE + '/Eyes' + img, slc)
     elif i == 11:
        pass
     else:
        slc = orImg[y_i:y_f,x_i:x_f]
        cv2.imwrite(outImgPathNE + '/NoEyes' + str(i) + img, slc)
```

Podem observar la presència de molts valors numèrics preestablerts, això és deu a què les imatges generades són invariants en la posició dels ulls com ja hem esmentat abans. De fet, ho són per a la majoria dels elements que conformen la cara. Per a poder extreure les imatges necessitades, és a dir, imatges de no-ulls, ens hem aprofitat d'aquest fet. Això ha facilitat en gran part la creació dels datasets.

Exemples de la classe *Eyes*

Figura 2: Exemple d'imatge de la classe *Eyes*

Figura 3: Exemple d'imatge de la classe *Eyes*

Exemples de la classe *noEyes*

Figura 4: Exemple d'imatge de la classe *noEyes*

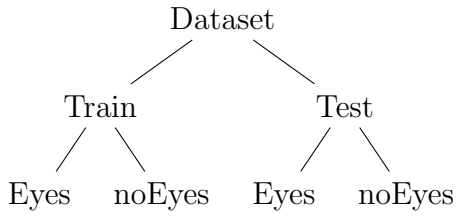
Figura 5: Exemple d'imatge de la classe *noEyes*

Figura 6: Exemple d'imatge de la classe *noEyes*

Creació dels *Datasets*

Un cop obtingudes les imatges de les classes ulls i no-ulls, hem separat les imatges en dos conjunts de dades: de *test* i d'entrenament (*training*). La proporció que hem utilitzat ha sigut de 1:9, és a dir, un 10% d'imatges destinades al dataset de *test* i la resta (el 90% de les imatges) al dataset de *training*. Amb aquesta proporció esperem trobar bons resultats en la classificació, ja que disposem d'un gran dataset.

A continuació, per a posteriorment tractar, extreure les *HOG features* i entrenar el classificador, hem jerarquitzat les dades de la següent manera:



En utilitzar aquesta jerarquia poden fer ús dels mètodes `fullfile()` i `imageDataStore()` que permeten obtenir mitjançant els paràmetres d'invocació, els *labels* i la classe de les imatges de manera "automàtica".

De manera generalitzada, respecte a la cardinalitat dels datasets, treballarem amb el següent:

Conjunt	**Classe**	**Cardinalitat**
Test	`Eyes`	65
	`noEyes`	1250
Training	`Eyes`	588
	`noEyes`	11252
TOTAL		13155

3 Entrenament del classificador

En aquesta secció tractarem el procés d'entrenament del classificador (codi inspirat en aquesta guia de MATLAB: *Digit Classification Using HOG Features*). Per començar, per a poder carregar el dataset de *training* amb els *labels* corresponents cridem a les funcions següents:

```
% Setting the directory for the training dataset
trainingDir = fullfile('Dataset', 'Train');
trainingSet =
    imageDatastore(trainingDir,'IncludeSubfolders',true,'LabelSource','foldernames');
```

A continuació, hem de crear els descriptors de les imatges. Després d'una recerca hem optat per utilitzar els histogrames de gradients orientats (de l'anglès *Histogram of Oriented Gradients* (o *HOG*)), que si recordem de teoria, funcionen molt bé per detectar objectes o figures com persones, però que en el nostre cas seran un parell d'ulls.

Aquesta tècnica de HOG té en compte les aparicions de l'orientació del gradient en porcions localitzades d'una imatge. Aquest mètode és similar a altres descriptors visuals, com ara *Edge Orientation Histograms* (*EOH*), però es diferencia d'aquests en el fet que es calcula sobre una malla densa de cel·les uniformement espaiades i utilitza la superposició del contrast local normalitzat per millorar la precisió.

Per la mida de la cel·la, hem anat provant diferents configuracions. La visualització mostra que una mida de cel·la de `[10 10]` no codifica molta informació de la silueta/forma dels ulls, mentre que una mida de cel·la de `[2 2]` codifica molta informació de forma, però augmenta la dimensionalitat del vector de característiques HOG de forma significativa. Un bon compromís és una mida de cel·la `[4 4]`. Aquesta mida codifica prou informació espacial per identificar visualment la forma d'un ull i limita el nombre de dimensions del vector de característiques HOG, cosa que ajuda a accelerar l'entrenament.

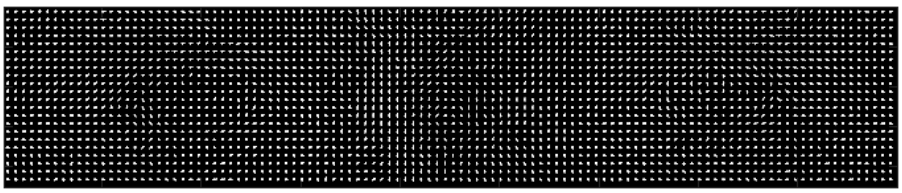


Figura 7: Visualització del HOG amb cel·la de mida `[4 4]`

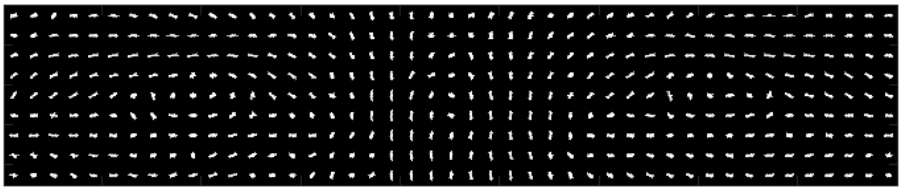


Figura 8: Visualització del HOG amb cel·la de mida `[10 10]`

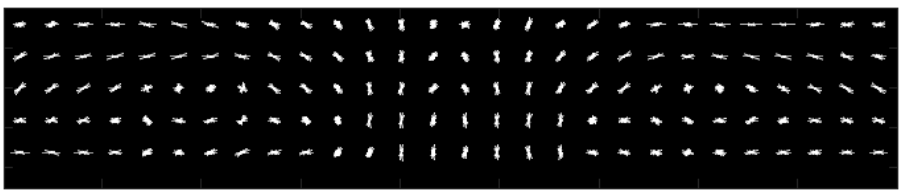


Figura 9: Visualització del HOG amb cel·la de mida `[16 16]`

Dit això hem definit la mida de la cel·la, també la mida de les *HOG features*, el nombre d'imatges, etc.; per posteriorment utilitzar el mètode ja implementat de MATLAB per extreure el HOG de les imatges.

```
% Declaring sizes and variables
img = readimage(trainingSet, 1);
[hog_4x4, ~] = extractHOGFeatures(img,'CellSize',[4 4]);

cellSize = [4 4];
hogFeatureSize = length(hog_4x4);

sizeTrainingSet = numel(trainingSet.Files);
trainingFeatures = zeros(sizeTrainingSet,hogFeatureSize,'single');
```

Una vegada definits els paràmetres, iterem per cada imatge del dataset per extreure les característiques de HOG d'aquestes i ho guardem a la matriu de `trainingFeatures`. Finalment, només en queda cridar al mètode `fitcsvm()` per ajustar les dades amb les etiquetes (és a dir si són ulls o no) a un model de màquina de vector de suport (*SVM*). El classificador retornat ho guardem com a arxiu de MATLAB.

```
% Extracting the HOG features for each image
for i = 1:sizeTrainingSet
    img = readimage(trainingSet,i);
    img = im2gray(img);
    trainingFeatures(i, :) = extractHOGFeatures(img,'CellSize',cellSize);
end

% Fits a SVM model
trainingLabels = trainingSet.Labels;
Classifier = fitcsvm(trainingFeatures, trainingLabels);
% Saving the classifier
save('Dataset/eyeClass.mat', 'Classifier', '-v7.3');
```

El temps emprat en entrenar aquest model amb una cel·la de mida `[4 4]` per el HOG és de 16 minuts (recordem que tenim 11.252 imatges de no-ulls i 588 d'ulls).

3.1 Entrenament amb `Classification Learner App`

De cara a l'entrega final, hem repetit l'entrenament inicial amb les característiques HOG però usant l'aplicació de `Classification Learner App` del mateix MATLAB. També hem realitzat l'estudi amb el descriptor LBP per comparar-ho amb HOG. Hem optat per utilitzar aquesta app, ja que incorpora eines que ens facilitava l'anàlisi estadístic juntament amb la versatilitat per entrenar amb diferents models d'aprenentatge ràpidament.

Així doncs, un requeriment per utilitzar aquesta aplicació de MATLAB és tenir les dades amb el format '`.csv`'. És a dir, una taula amb files, on cada fila es una mostra del dataset; i columnes, on cada columna significa un atribut o característica de les mostres. Per tant, per obtenir el format '`.csv`' vam modificar el codi de la següent manera:

```
function [] = saveHOGdescriptor()
    % Setting the directory for the training dataset
    dir = fullfile('Dataset');
    set =
        imageDatastore(dir,'IncludeSubfolders',true,'LabelSource','foldernames');

    % Declaring sizes and variables
    cellSize = [16 16];

    % Declaring sizes and variables
    img = readimage(set, 1);
    [hog_, ~] = extractHOGFeatures(img,'CellSize',cellSize);

    hogFeatureSize = length(hog_);

    sizeSet = numel(set.Files);
    trainingFeaturesHOG = zeros(sizeSet,hogFeatureSize,'single');

    % Extracting the HOG and LBP features for each image
    for i = 1:sizeSet
        img = readimage(set,i);
        img = im2gray(img);
        trainingFeaturesHOG(i, :) = extractHOGFeatures(img,'CellSize',cellSize);
    end

    % Get labels for each image
    labels = grp2idx(set.Labels);

    % Join train features and labels
    labeledFeaturesHOG = [trainingFeaturesHOG labels];

    % Saves the matrix as a csv file
    csvwrite('Dataset/labeledFeaturesHOG.csv',labeledFeaturesHOG);
end
```

Amb aquest codi, guardem la matriu de característiques HOG de cada mostra/imatge que tenim a la base de dades en format '`.csv`'. Per a poder agilitzar el procés d'entrenament, hem optat finalment per a utilitzar una mida de cel·la de `[32 32]`, hem optat per aquesta mida perquè les mètriques de rendiment dels models no variaven molt amb valors de `[4 4]` o `[16 16]` mentre que el temps d'entrenament augmentava considerablement. Cal dir que per l'estudi amb les característiques LBP hem seguit aquest mateix esquema per obtenir el format '`.csv`'.

Una vegada tenim les dades en format '`.csv`', només cal importar aquestes des de l'aplicació de `Classification Learner App`. Aquí haurem d'especificar la variable de "*response*" o *target* que és la variable que volem predir, és a dir, una etiqueta (concretament l'etiqueta '1' o '2') que ens dirà si és un ull o no donat un vector de característiques HOG o LBP de la imatge.

Per al descriptor HOG (*Histogram of Oriented Gradients*), aquesta vegada hem definit la mida de la cel·la de `[32 32]` ja que és el balanç perfecte entre el temps de computació i bona precisió. Vam provar amb cel·les de menor mida, però no valien la pena, ja que tardaven més a computar i donaven pitjors resultats.

Respecte al descriptor *Local Binary Patterns*, observem que transforma la imatge d'entrada en un *array* d'*integers* que descriuen a petita escala les diferencies de textura de la imatge. Amb la mida de la cel·la, la *CellSize*, es pot modificar l'escala a la qual es detalla la textura.

De manera gràfica els *features* extrets es mostren de la següent manera:

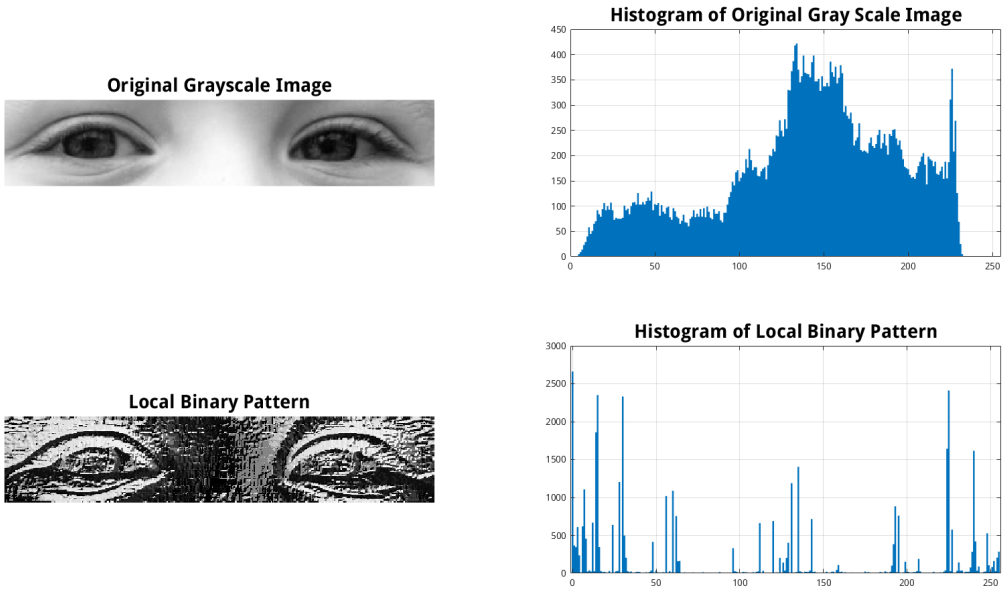


Figura 10: Visualització dels LBP Features

La mida dels vectors de característiques pels dos casos son:

HOG:	**LBP:**
468	1652

4 Experimentació i resultats provisionals

En aquesta secció tractarem i mostrarem els resultats obtinguts amb les mètriques que hem calculat, aquestes seran principalment la *confusionchart* similar a la *comfusionmatrix* i el prencentatge d'encerts.

Per a poder experimentar amb el dataset de test, hem seguit la mateixa estructura que a l'entrenament. Definim els directoris, els paràmetres com la mida de les cel·les, el nombre d'imatges, etc.; i de la mateixa manera iterem per cada imatge del dataset de test. Processem cada imatge, extraiem les seves característiques de HOG i les guardem dins de la matriu de característiques.

Per últim, per no tornar a computar-ho de nou per les següents execucions (i per estalviar temps a l'hora de *debuggar*), hem guardat aquesta matriu de característiques del dataset de test com un arxiu de MATLAB.

```
function [] = saveTestHOGfeatures()
    % Setting the directory for the test dataset
    testDir = fullfile('Dataset', 'Test');
    testSet =
        imageDatastore(testDir,'IncludeSubfolders',true,'LabelSource','foldernames');

     % Declaring sizes and variables
    img = readimage(testSet, 1);
    [hog_4x4, ~] = extractHOGFeatures(img,'CellSize',[4 4]);

    cellSize = [4 4];
    hogFeatureSize = length(hog_4x4);

    numImages = numel(testSet.Files);
    features = zeros(numImages,hogFeatureSize,'single');

    % Extracting the HOG features for each test image
    for j = 1:numImages
        img = readimage(testSet,j);
        img = im2gray(img);
        features(j, :) = extractHOGFeatures(img,'CellSize',cellSize);
    end

    % Saving the HOG features matrix of the test images
    save('Dataset/featuresTest.mat', 'features', '-v7.3');
end
```

Confusion matrix i *chart*

Un cop calculats els HOG *features* per al conjunt de test i tenim el classificador, ja podem calcular la matriu de confusió i la `confusionchart()` implementem el codi següent:

```
% Make class predictions using the test features
predictedLabels = predict(eyeClassifier.Classifier, testFeatures.features);

% Tabulate the results using a confusion matrix
confMat = confusionmat(setLabels, predictedLabels);

% Prints the confusion matrix graphically
confChart = confusionchart(setLabels, predictedLabels)
```

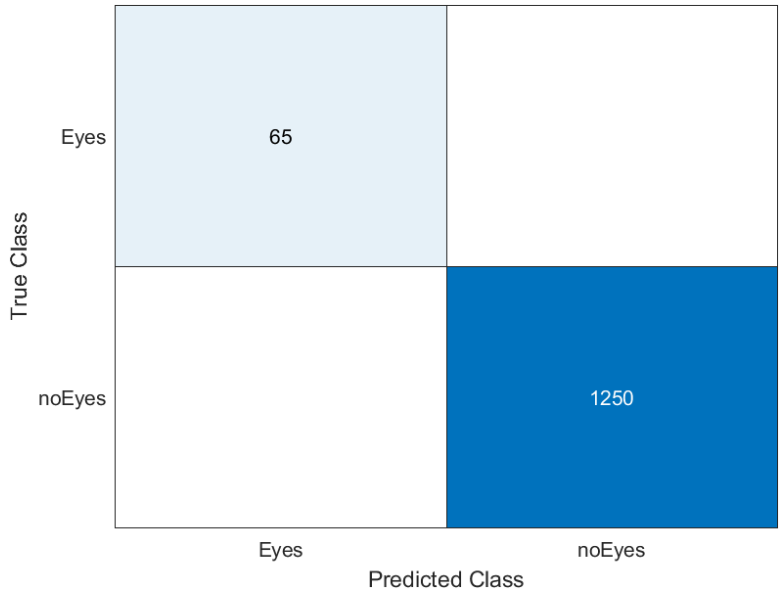


Figura 11: Predicció

Observem que totes les imatges de test pertanyen als *true positives* o *true negatives*, trobem que el model prediu correctament totes les imatges del conjunt test. Aquests resultats semblen una mica estranys donat que estem treballant amb un conjunt no exageradament extens i, per tant, podria ser el resultat d'*overfitting*.

Accuracy

Un cop calculada la matriu de confusió procedim a calcular l'*accuracy* de la predicció, l'*accuracy* mesura la proporció d'encerts global de la predicció duta a terme. Aquesta la podem calcular de manera simple de la forma següent:

```
% Accuracy calculation
accuracy = sum(predictedLabels == setLabels)/numel(setLabels)
```

Com podem esperar observant la matriu de confusió, l'*accuracy* que obtenim és d'1.

4.1 Estudi estadístic final

Utilitzant el `Classification Learner App` hem explorat una gran varietat de diferents models. Per a trobar els més adequats, hem fet diverses execucions per als dos conjunts de descriptors, el basat en els `HOG Features` i el basat en `LBP Features`. El protocol de '*resampling*' que hem fet servir és *cross-validation* amb 5 particions del dataset.

Experimentació amb els *HOG Features*

Començarem per visualitzar el '*scatter plot*' per veure l'aspecte general de les nostres dades.

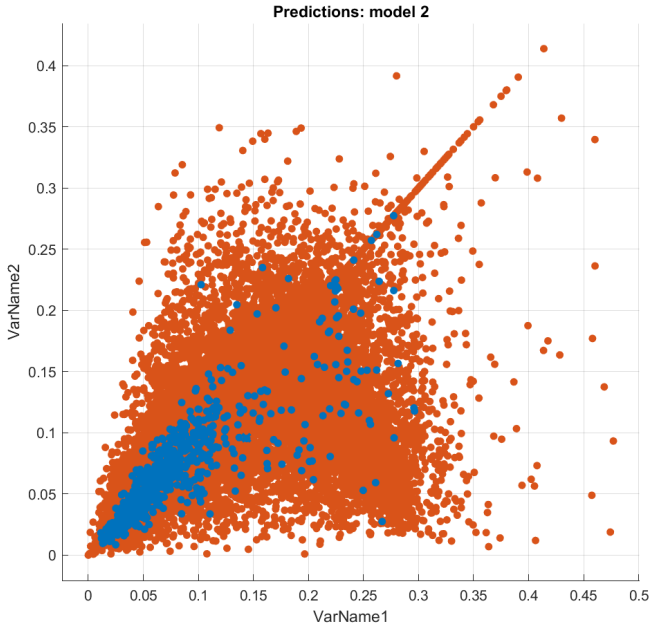


Figura 12: *Scatter plot* de les dades (en blau 'ulls')

El primer model de predicció que analitzarem és el '*Coarse Tree*'. Aquest és un model de predicció basat en un arbre de decisió. Aquest model presenta molts bons resultats, concretament observem una precisió del **99.0%**. Com veiem a la figura d'avall, aquest model encerta la majoria de positius i negatius vertaders.

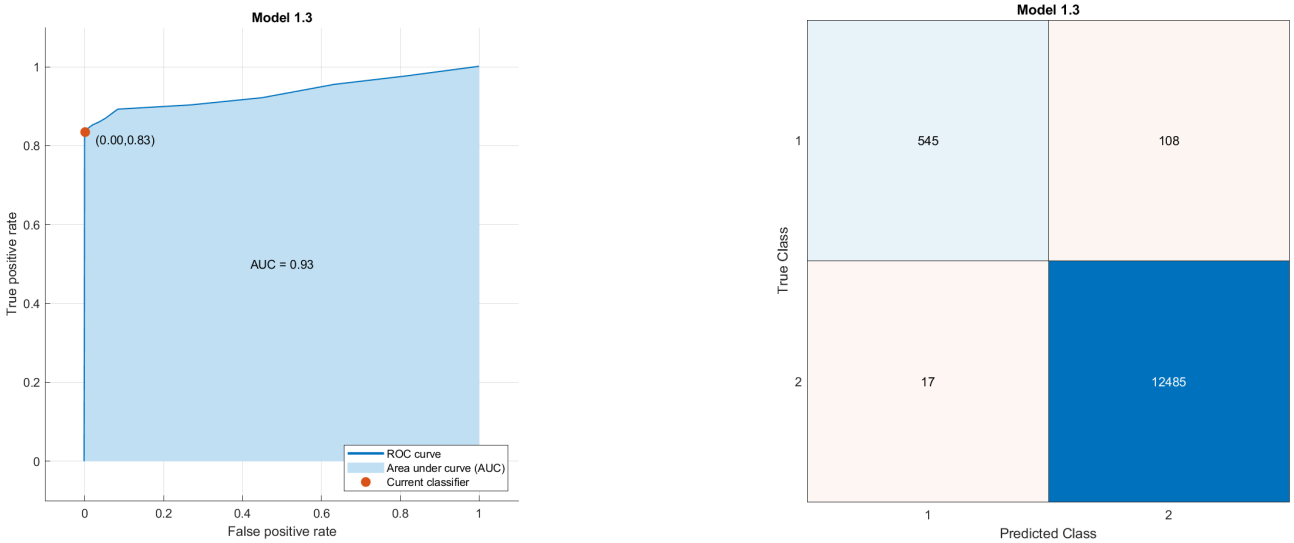


Figura 13: Rendiment del model: corba de ROC i matriu de confusió

A continuació, tenim el model de 'Cosine KNN'. Aquest model està basat en el KNN original, l'única diferència és que utilitza com a mètrica de distància la similitud cosinus. Com veiem a la figura d'avall, la presició és encara més bona comparat amb el model anterior (el model '*Coarse Tree*'), concretament té una precisió del **99.9%**.

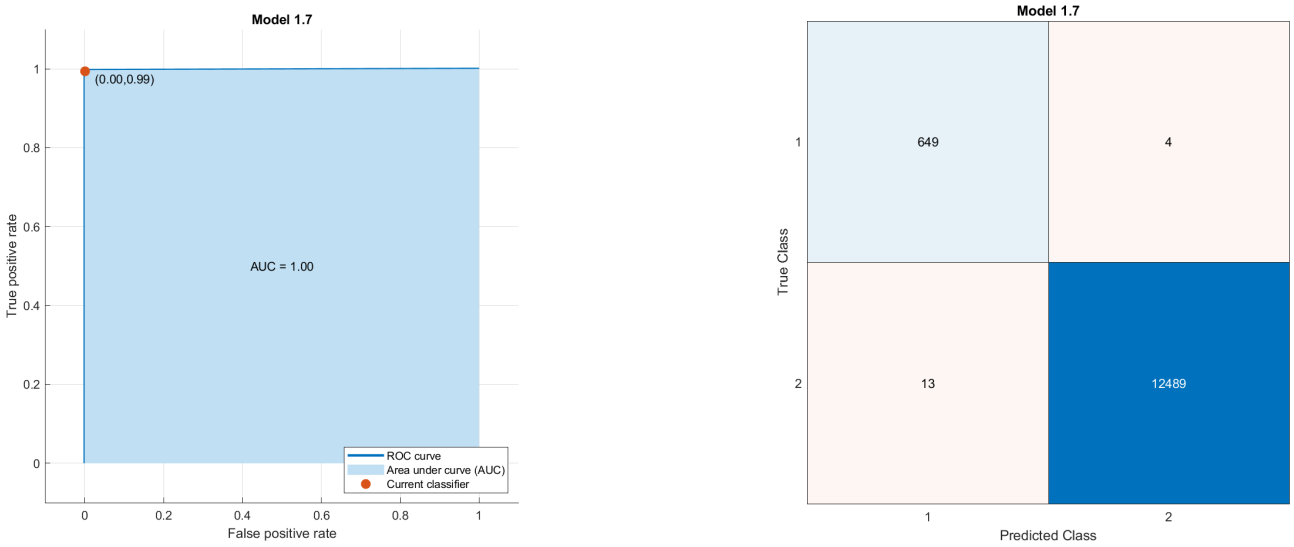


Figura 14: Rendiment del model: corba de ROC i matriu de confusió

Finalment, com a últim model més que veurem utilitzant les característiques HOG és '*Linear SVM*'. Aquest model es basa en el model de màquina de vector suport d'aprenentatge supervisat. Com veiem a la figura posterior, aquest molt també dona bons resultats, obté una precisió del **99.9%** també. L'única diferència que hi ha amb el model anterior, si ens fixem en ambdues matrius de confusió, veiem que un detecta més falsos positius, i l'altre, més falsos negatius.

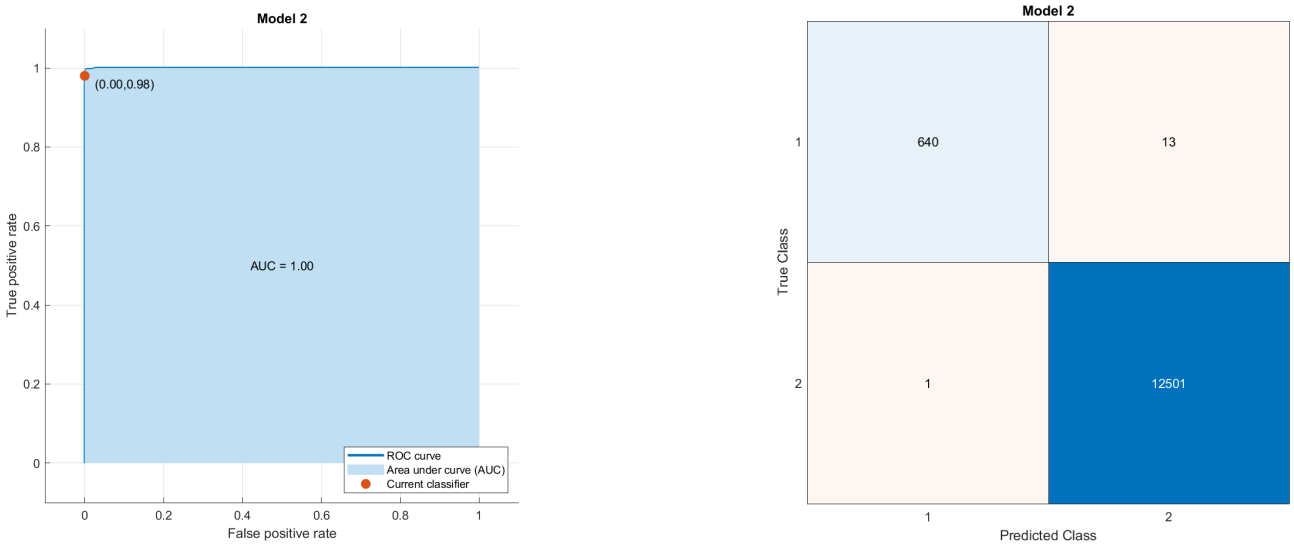


Figura 15: Rendiment del model: corba de ROC i matriu de confusió

Experimentació amb els LBP Features

Per a començar l'estudi, comencem visualitzant els *Scatter plots* resultants per al conjunt d'imatges. Observem diferències difícilment identificables a simple ull, per tant, prenem com a referència el *scatter plot* del model KNN.

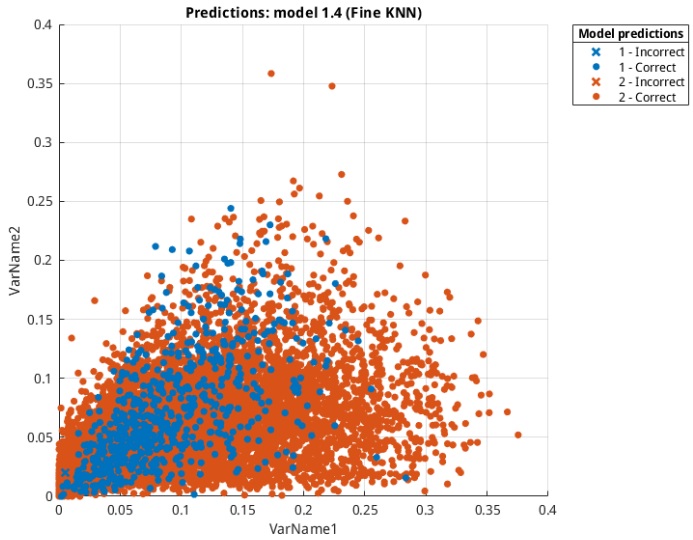


Figura 16: *Scatter plot* per el model KNN

De manera inicial hem decidit experimentar amb el model de '*Fine Tree*', aquest model, basat en arbres de classificació, mostra uns resultats amb precisió elevada, d'aproximadament **99%**. A continuació observem amb més detall els resultats del model.

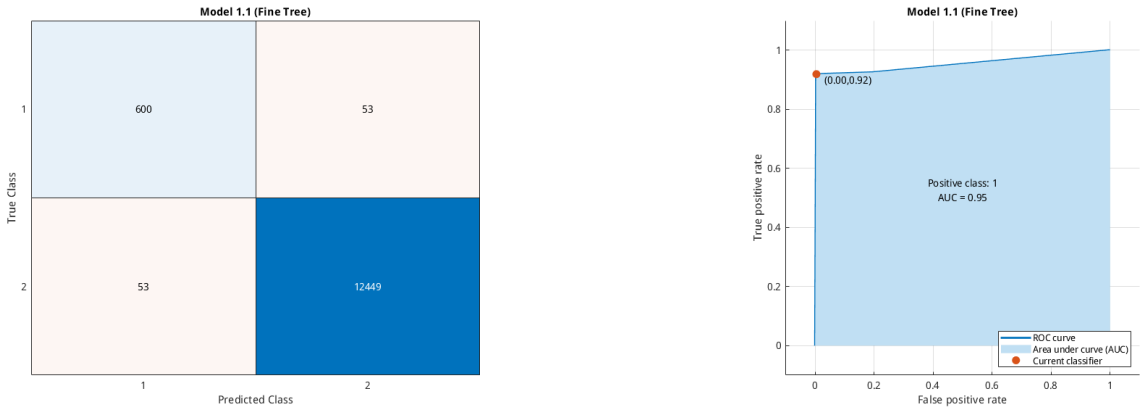


Figura 17: Confusion matrix and ROC Curve for FineTree Model

El segon model amb el qual hem experimentat per a les LBP Features ha sigut el model de KNN, aquest mostra uns resultats bons. Ho podem observar a continuació en la matriu de confusió, trobem que la predicció del model encerta aproximadament el **100%** de les imatges. Això ho podem veure amb més detall en la ROC Curve.

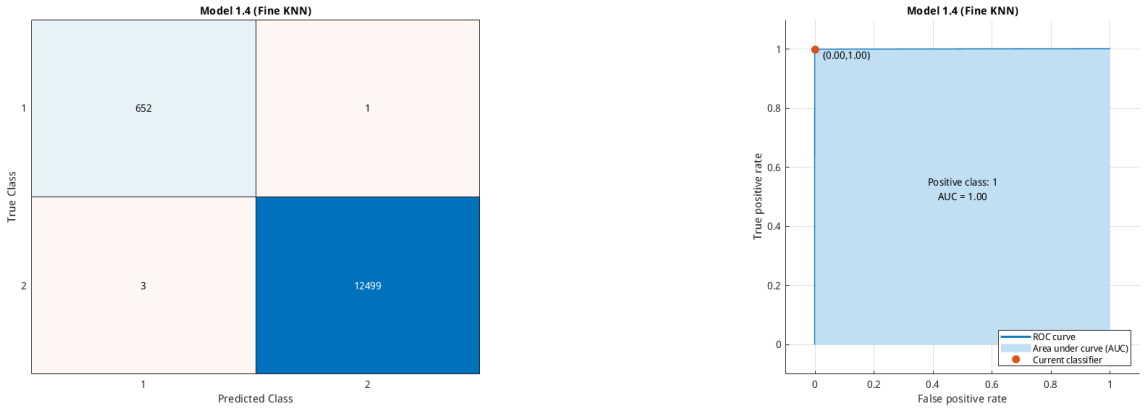


Figura 18: Confusion matrix and ROC Curve for KNN Model

Finalment, l'últim model amb el qual provem és el '`Narrow Neural Network`' basat en xarxes neuronals, presenta uns resultats molt bons, molt similars als obtinguts amb el model `KNN`.

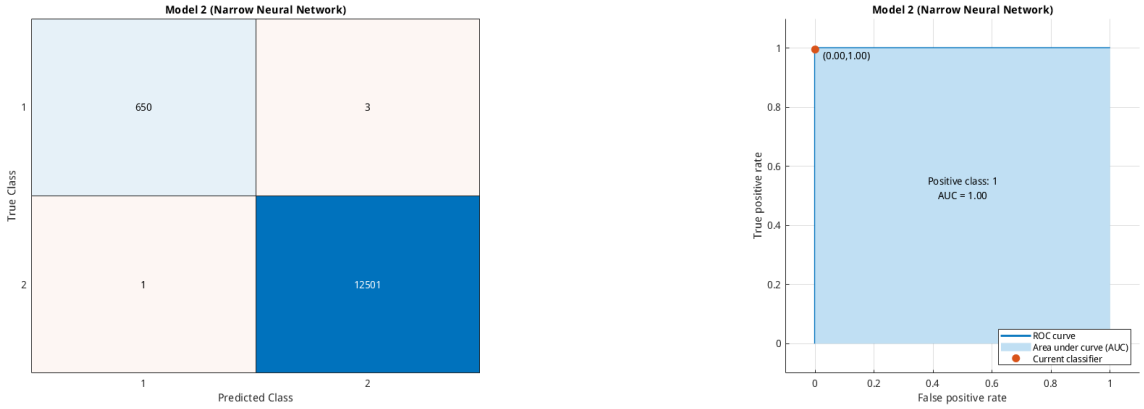


Figura 19: Confusion matrix and ROC Curve for KNN Model

Trobem una `accuracy` del **100%**, ho podem veure representat en la matriu de confusió, on la classificació cau majoritàriament en True Positives i False Negatives, amb la ROC Curve tornem a veure reflectida aquesta `accuracy` elevada.

Observem que no hi ha una combinació de descriptor i model preferent, els dos descriptors mostren mètriques molt bones en funció del model amb el qual es combini. Cal destacar que les mètriques són lleugerament millors per al cas del descriptor basat en `LBP`, això si, el *trade-off* es dona en els temps d'execució. Observant les mides dels vectors dels descriptors entenem aquesta lleugera diferència, en tenim bastants més per al cas del `LBP`.

5 Annex

Les funcions que hem implementat són: `main()`, `trainingClassifier()`, `saveHOGdescriptor()`, `saveLBPdescriptor()`, `saveTestHOGfeatures()` i `printResults()`. La resta de funcions són del mateix MATLAB.

Script d'obtenció de característiques HOG en format `.csv` (codi nou)

```
function [] = saveHOGdescriptor()
    % Setting the directory for the training dataset
    dir = fullfile('Dataset');
    set =
        imageDatastore(dir,'IncludeSubfolders',true,'LabelSource','foldernames');

    % Declaring sizes and variables
    cellSize = [16 16];

    % Declaring sizes and variables
    img = readimage(set, 1);
    [hog_, ~] = extractHOGFeatures(img,'CellSize',cellSize);

    hogFeatureSize = length(hog_);

    sizeSet = numel(set.Files);
    trainingFeaturesHOG = zeros(sizeSet,hogFeatureSize,'single');

    % Extracting the HOG and LBP features for each image
    for i = 1:sizeSet
        img = readimage(set,i);
        img = im2gray(img);
        trainingFeaturesHOG(i, :) = extractHOGFeatures(img,'CellSize',cellSize);
    end

    % Get labels for each image
    labels = grp2idx(set.Labels);

    % Join train features and labels
    labeledFeaturesHOG = [trainingFeaturesHOG labels];

    % Saves the matrix as a csv file
    csvwrite('Dataset/labeledFeaturesHOG.csv',labeledFeaturesHOG);
end
```

Script d'obtenció de característiques LBP en format `.csv` (codi nou)

```
function [] = saveLBPdescriptor()
    % Setting the directory for the training dataset
    dir = fullfile('Dataset');
    set =
        imageDatastore(dir,'IncludeSubfolders',true,'LabelSource','foldernames');

    % Declaring sizes and variables
    cellSize = [16 16];

    % Declaring sizes and variables
    img = readimage(set, 1);
    lbpFeatures = extractLBPFeatures(img,'CellSize',[16 16]);
    lbpFeatureSize = length(lbpFeatures);

    sizeSet = numel(trainingSet.Files);

    featuresLBP = zeros(sizeSet,lbpFeatureSize,'single');

    % Extracting the HOG and LBP features for each image
    for i = 1:sizeTrainingSet
        img = readimage(set,i);
        img = im2gray(img);
        featuresLBP(i, :) = extractLBPFeatures(img,'CellSize',cellSize);
    end

    % Get labels for each image
    labels = grp2idx(trainingSet.Labels);

    % Join train features and labels
    labeledFeaturesLBP = [trainingFeaturesLBP labels];

    % Saves the matrix as a csv file
    csvwrite('Dataset/labeledFeaturesLBP.csv',labeledFeaturesLBP);
end
```

Script del `main()`

```
% Netejem les variables i les comandes anteriors
clc;
clear all;
close all;

trainingClassifier();
saveTestHOGfeatures();
printResults();
```

Script de l'entrenament del classificador

Idea extreta de: MATLAB: *Digit Classification Using HOG Features*

```
function [] = trainingClassifier()
    % Setting the directory for the training dataset
    trainingDir = fullfile('Dataset', 'Train');
    trainingSet =
        imageDatastore(trainingDir,'IncludeSubfolders',true,'LabelSource','foldernames');

    % Declaring sizes and variables
    img = readimage(trainingSet, 1);
    [hog_4x4, ~] = extractHOGFeatures(img,'CellSize',[4 4]);

    cellSize = [4 4];
    hogFeatureSize = length(hog_4x4);
    sizeTrainingSet = numel(trainingSet.Files);
    trainingFeatures = zeros(sizeTrainingSet,hogFeatureSize,'single');

    % Extracting the HOG features for each image
    for i = 1:sizeTrainingSet
        img = readimage(trainingSet,i);
        img = im2gray(img);
        trainingFeatures(i, :) = extractHOGFeatures(img,'CellSize',cellSize);
    end

    % Get labels for each image
    trainingLabels = trainingSet.Labels;

    % Fits a SVM model
    Classifier = fitcsvm(trainingFeatures, trainingLabels);

    % Saving the classifier
    save('Dataset/eyeClass.mat', 'Classifier', '-v7.3');
end
```

Script d'extracció de features per a test

Idea extreta de: MATLAB: *Digit Classification Using HOG Features*

```
function [] = saveTestHOGfeatures()

    % Setting the directory for the test dataset
    testDir = fullfile('Dataset', 'Test');
    testSet =
        imageDatastore(testDir,'IncludeSubfolders',true,'LabelSource','foldernames');

     % Declaring sizes and variables
    img = readimage(testSet, 1);
    [hog_4x4, ~] = extractHOGFeatures(img,'CellSize',[4 4]);

    cellSize = [4 4];
    hogFeatureSize = length(hog_4x4);

    numImages = numel(testSet.Files);
    features = zeros(numImages,hogFeatureSize,'single');

    % Extracting the HOG features for each test image
    for j = 1:numImages
        img = readimage(testSet,j);
        img = im2gray(img);
        features(j, :) = extractHOGFeatures(img,'CellSize',cellSize);
    end

    % Saving the HOG features matrix of the test images
    save('Dataset/featuresTest.mat', 'features', '-v7.3');

end
```

Script càlcul de metriques simples

Càlcul métriques: MATLAB: *How to calculate confusion matrix accuracy and precision*

```
function [] = printResults()

    % Setting the test directory
    testDir = fullfile('Dataset', 'Test');
    testSet =
        imageDatastore(testDir,'IncludeSubfolders',true,'LabelSource','foldernames');

    % Getting the labels
    setLabels = testSet.Labels;

    % Loading the classifier and the test features matrix
    testFeatures = matfile('Dataset/featuresTest.mat');
    eyeClassifier = matfile('Dataset/eyeClass.mat');

    % Make class predictions using the test features
    predictedLabels = predict(eyeClassifier.Classifier, testFeatures.features);

    % Tabulate the results using a confusion matrix
    confMat = confusionmat(setLabels, predictedLabels);

    % Prints the confusion matrix graphically
    confChart = confusionchart(setLabels, predictedLabels);

    % Accuracy calculation
    accuracy = sum(predictedLabels == setLabels)/numel(setLabels)

end
```