Reconeixement de fulles

Projecte curt, Visió per Computador

Oscar Mañas
Jordi Piqué

1. Procés de desenvolupament

Davant d'un problema relativament complex i amb múltiples solucions vàlides, com és el fet de classificar imatges, és difícil encertar a la primera una solució satisfactòria. Normalment, en aquest tipus de situacions, es requereix tenir clara la base teòrica en la qual es basarà la solució i anar posant en pràctica tals coneixements per veure el resultat real que donen.

En aquest procés iteratiu d'anar provant diferents solucions és on es construeix el camí cap a la solució final bona. Davant la incertesa dels resultats d'aplicar un algorisme o un altre, cal provar-los tots dos per veure quin dels dos camins és el que s'acosta més cap allà on es vol anar.

En els següents apartats es mostra les diferents aproximacions que s'han anat intentant per solucionar el problema, per què algunes s'han descartat de cara a la solució final i per què d'altres han sobreviscut i han quedat seleccionades.

1.1. Vector de característiques

Pràcticament la major part del problema recau en trobar un vector de característiques que pugui descriure de forma clara, robusta i unívoca les diferents fulles. Dos vectors de característiques d'una mateixa espècie de fulles han de variar mínimament i entre fulles de diferents espècies ha de ser clarament diferent. També s'ha d'evitar trobar característiques redundants, doncs això podria disminuïr la precisió dels classificadors.

Amb aquests requisits a la ment, es va intentar definir matemàticament, per poder-ho implementar en Matlab, les característiques que feien que l'ull humà pogués diferenciar i reconèixer les diverses fulles.

1.1.1. Descriptors de Fourier

El contorn és potser la característica que més crida l'atenció a un humà per tal de classificar fulles. Un mètode que en principi hauria de funcionar bé per descriure la forma d'una fulla són els descriptors de Fourier.

Els descriptors de Fourier representaven un repte de programació, ja que s'havien d'implementar "a mà" (no hi ha funcions predefinides de Matlab que ho facin). Per

comprovar que l'algorisme fos correcte, es va crear una funció que recomponia el contorn de la imatge original a partir dels descriptors de Fourier.

A continuació es poden veure 3 imatges originals binaritzades i la reconstrucció d'aquestes, utilitzant els 50 primers descriptors de Fourier.

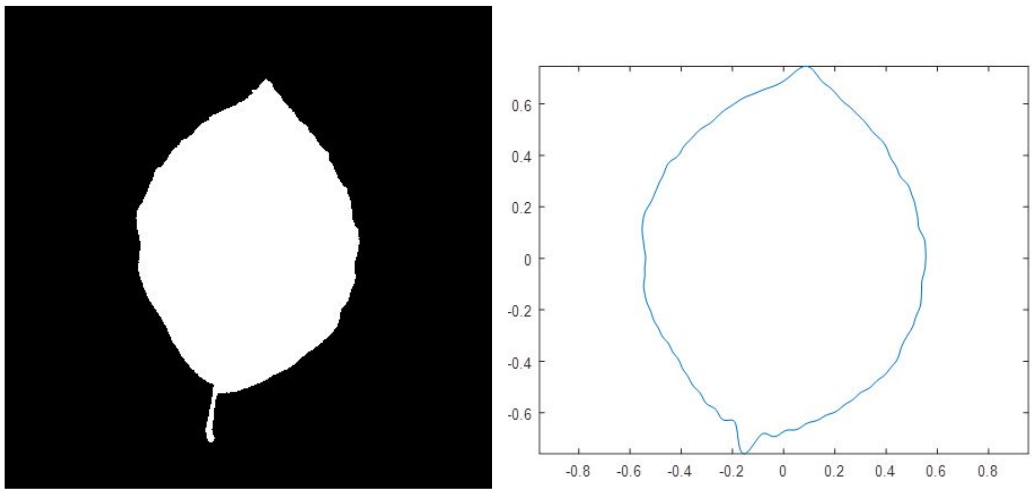

Fig1. I15nr043.tif

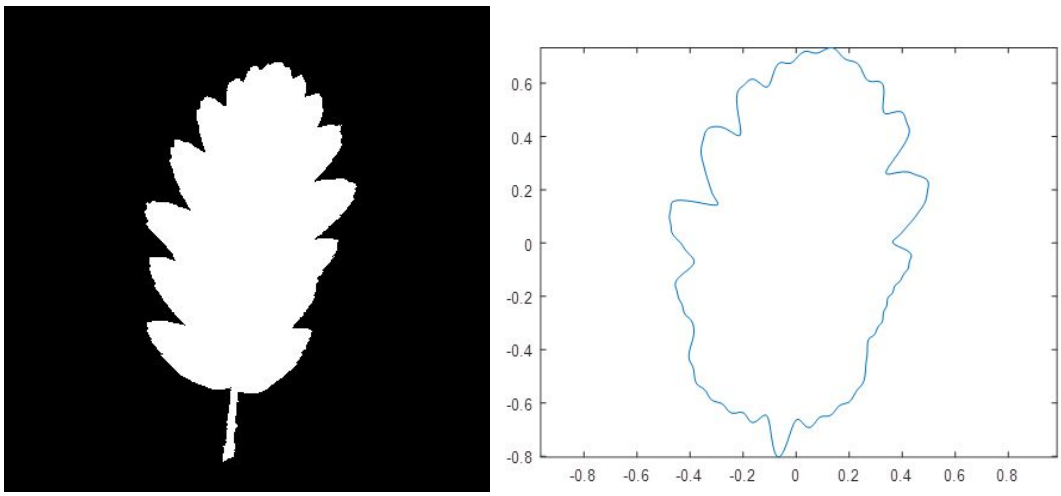

Fig 2. I14nr067.tif

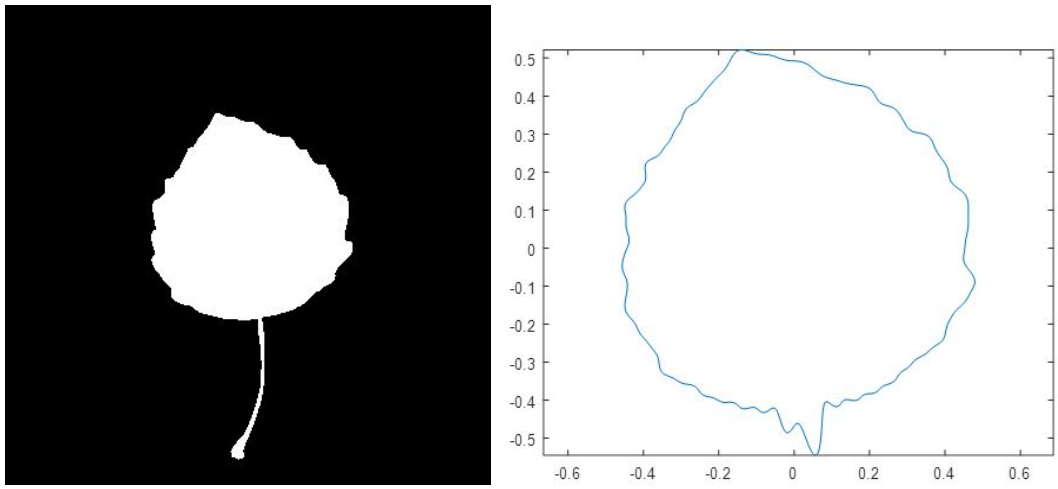

Fig 3. I8nr029.tif

Tot i que el contorn reconstruït s'aproxima força a l'original, es poden veure els problemes que immediatament van emergir.

Les fulles amb concavitats en el seu contorn no quedaven ben descrites amb Fourier. Es pot veure com hi ha zones de la fulla *I14nr067.tif* que no queden ben reconstruïdes del tot. També s'observa que en fulles com la *I8nr029.tif*, que tenen pecíol, aquest no queda representat en els descriptors.

Per conèixer el rendiment real dels descriptors de Fourier, es van extreure característiques d'un subconjunt de fulles i es va entrenar un classificador TreeBagger. El percentatge d'encert només rondava el 40%.

Per tal d'augmentar la precisió de la classificació mitjançant descriptors de Fourier, calia separar el pecíol de la fulla en si. Finalment, això no es va dur a terme ja que, tal com es veu al següent apartat, l'ús dels HOGs com a descriptors donava molta més potència a la descripció numèrica del contorn i eliminava els problemes que tenien els descriptors de Fourier.

1.1.2. HOGs sobre la imatge binaritzada

Mentre es provava l'eficàcia dels descriptors de Fourier per descriure el contorn de la fulla, també es va decidir provar els histogrames de gradients orientats o HOGs. Aquests tenen un clar avantatge sobre els descriptors de Fourier i és que codifiquen característiques locals i no globals, de manera que eviten haver de segmentar la imatge i, a més, són robustos al

soroll i les oclusions. Amb el conjunt d'imatges del problema plantejat, però, el guany no es tan gran ja que les fulles són molt fàcils de segmentar, i no hi ha oclusions ni soroll. Tot i així, es va decidir provar els HOGs com a descriptors ja que solucionaven alguns dels problemes que plantejaven els descriptors de Fourier.

En aquest cas, tot i que no representaven un repte d'implementació, es va optar per utilitzar la implementació de Matlab amb la funció *extractHOGFeatures*, doncs així es podia dedicar més temps a tasques més importants, com fixar els paràmetres de l'extracció dels HOGs o tunejar els classificadors. A més a més, això evita errors d'implementació i la consegüent pèrdua de temps en trobar-los i solucionar-los. L'entrada és la imatge (amb 1 o 3 canals), mentre que la sortida és un vector de característiques i, opcionalment, un objecte per visualitzar els HOGs. A més, es poden especificar una sèrie d'opcions mitjançant parells nom-valor passats com a argument a la funció. El nombre de característiques obtingudes depèn de la mida de cel·la, la mida de bloc, la quantitat d'overlapping entre blocs i el nombre de bins en cada histograma.

Inicialment, es va fixar un tamany de cel·la de 16x16 píxels, un tamany de bloc de 2x2 cel·les, un overlapping entre blocs del 50% i 9 bins per cada histograma.

A continuació es poden veure els HOGs de les 3 imatges anteriors:

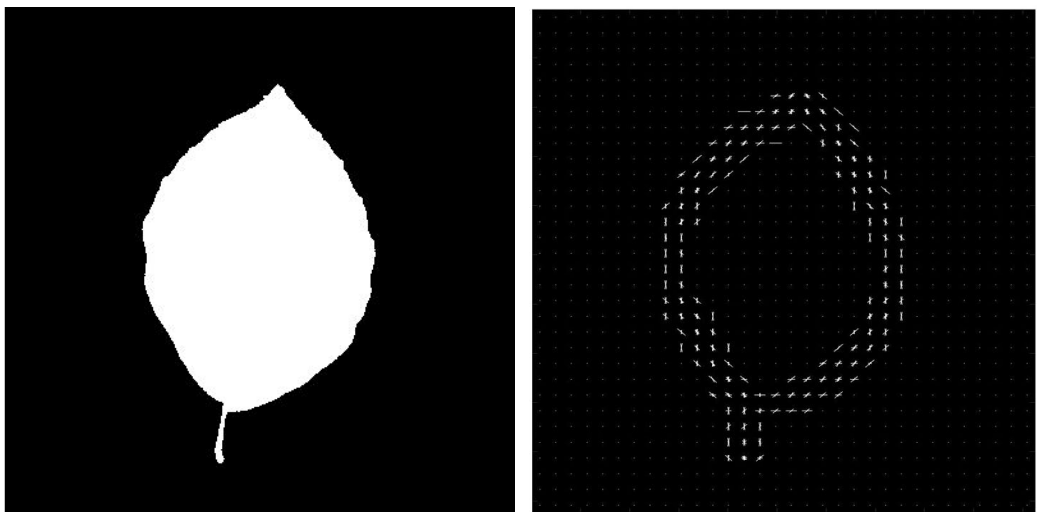

Fig 4. l15nr043.tif

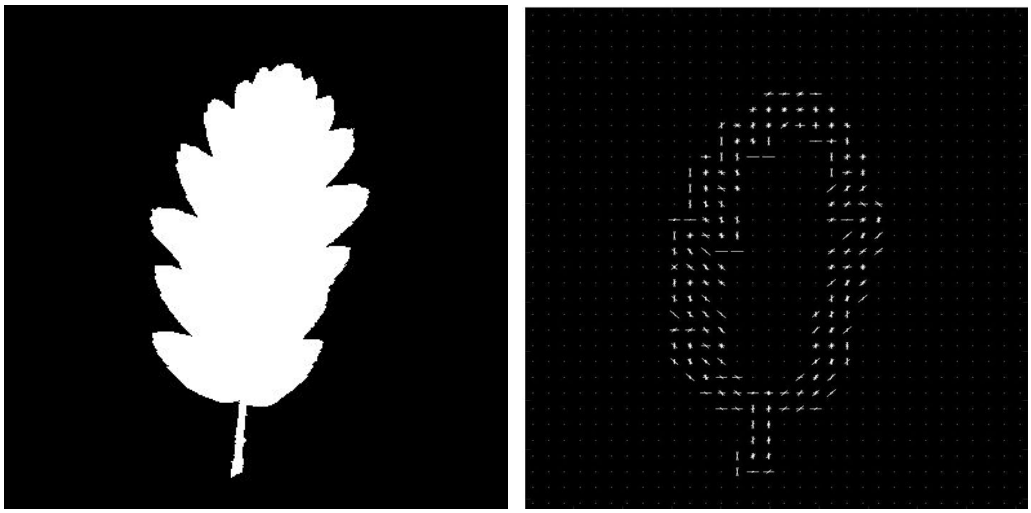

Fig 5. l14nr067.tif

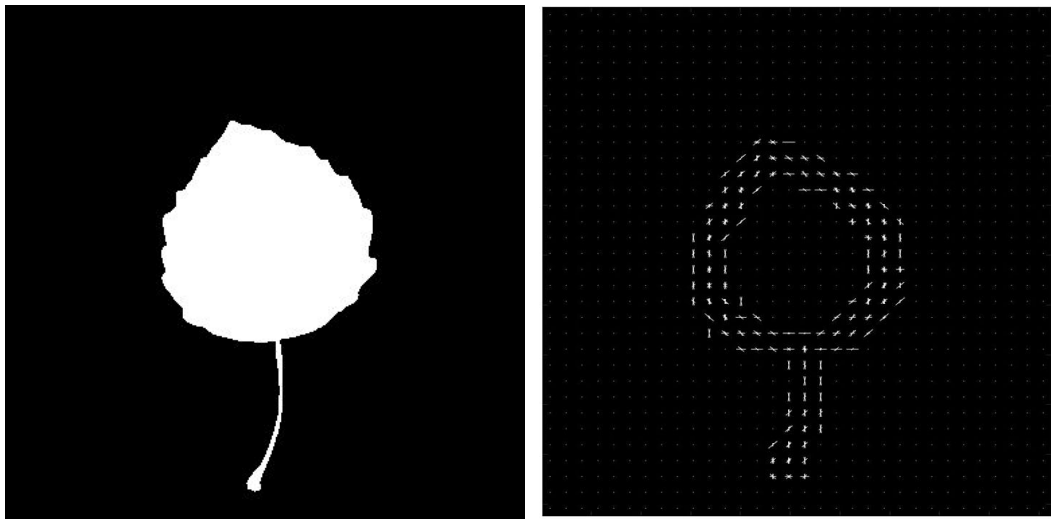

Fig 6. l8nr029.tif

Per conèixer el rendiment dels HOGs com a descriptors de les fulles, es va dur a terme una 10-fold cross-validation amb tot el dataset fent servir una SVM amb un kernel lineal com a classificador. El percentatge d'encert va augmentar fins al 87,6%. La matriu de confusió obtinguda es mostra a continuació.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	56	0	9	0	0	0	0	0	18	0	5	0	0	3	0
2	0	63	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	53	0	0	0	0	0	6	0	1	0	0	2	0
4	0	0	0	67	0	0	0	0	0	3	0	0	0	4	0
5	0	0	0	0	68	2	0	1	0	0	2	1	0	0	0
6	1	1	0	0	0	67	0	6	0	0	0	6	0	1	0
7	0	0	0	0	0	0	70	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	60	0	0	0	5	3	0	0
9	11	0	7	0	0	0	0	0	45	0	0	0	0	1	0
10	0	0	0	0	0	0	0	0	0	63	0	0	0	0	0
11	1	0	1	0	0	0	0	0	1	0	59	0	0	0	4
12	0	6	0	0	0	1	0	1	0	0	0	58	1	0	0
13	0	0	0	0	1	0	0	2	0	0	0	0	66	0	0
14	0	0	0	3	0	0	0	0	0	4	0	0	0	59	0
15	0	0	0	0	0	0	0	0	0	0	3	0	0	0	66

Fig 7 Matriu de confusió

Es pot observar que els valors més grans de classificacions errònies es produeixen entre les classes 1, 3 i 9. A continuació es detalla el perquè i com es va decidir atacar aquest problema.

1.1.3. Diferenciar les classes 1, 3 i 9

Tal com s'ha vist a l'apartat anterior, l'ús dels HOGs sobre la imatge binaritzada descriu el contorn de les fulles d'una manera molt precisa i robusta. No obstant això, en la classificació de fulles amb un contorn molt similar, com ho són les de les classes 1, 3 i 9, el rendiment dels HOGs baixava.

Per solucionar aquest problema es va intentar buscar característiques diferents del contorn que servissin per diferenciar aquestes classes més problemàtiques.

1.1.3.1. Quantificar la textura

Entre les classes 1 i 9 s'observa a simple vista que les fulles de la classe 9 tenen més nervis petits que les de la classe 1. És a dir, la textura de la classe 9 és més "rugosa" que la de la classe 1, que és més "llisa".

Per quantificar la textura de les fulles es va decidir calcular la magnitud del gradient de cada píxel de la imatge, exceptuant els píxels del contorn, sumar els valors de tots els gradients i dividir l'àrea de la fulla per aquest resultat.

Classe 1	0,0421	0,0431	0,0423	0,0433	0,0457	0,0461	0,0459	0,0420	0,0382	0,0439
Classe 9	0,0282	0,0372	0,0339	0,0327	0,0342	0,0327	0,0363	0,0329	0,0308	0,0394

10 primeres dades

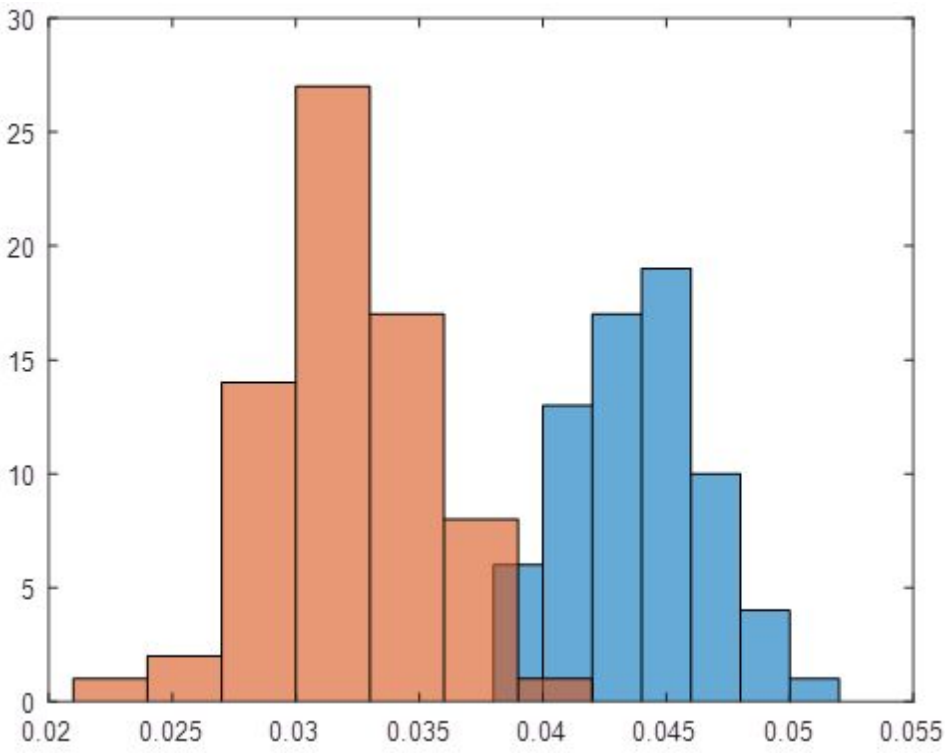

Fig 8. Histograma de la textura de les classes 1 i 9

Tal com es veu al gràfic anterior, aquesta mesura de la textura és un molt bon descriptor per diferencia les classes 1 i 9.

1.1.3.2. Comptar el nombre de nervis

Entre les classes 1 i 3, l'única característica que sembla que les pot diferencia és el nombre de nervis principals. Ara bé, realitzar un algorisme que dugui a terme un comptatge precís i robust del nombre de nervis no és una tasca fàcil.

Per tal de ressaltar més els nervis, es va aplicar primer un filtre Sobel. Tanmateix, aquest filtre ressalta també el soroll i altres textures de la fulla que no corresponen als nervis principals. Per això es va aplicar també un filtre Gaussià.

Després es va binaritzar la imatge. Per fer-ho, es va fer de forma local, ja que la intensitat mitjana de gris no és constant a tota la imatge. Per tal d'eliminar les petites estructures blanques, que no corresponen als nervis principals, es va aplicar un bwareaopen.

Finalment, per tal de comptar els nervis binaritzats, es va comptar el nombre de components connexes d'una regió vertical de la imatge.

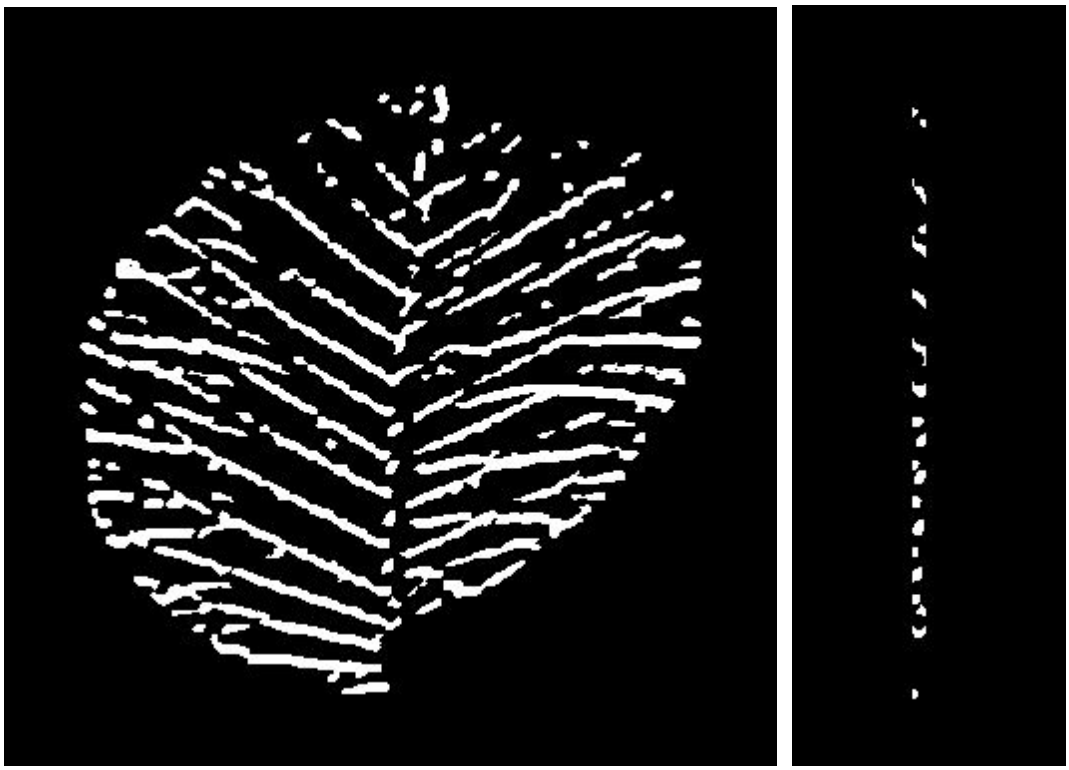

Fig 9. I1nr001.tif

El problema d'aquesta tècnica és que és poc robusta i, tot i que les mitjanes del nombre de nervis de les classes 1 i 9 són diferents, la variabilitat d'aquest descriptor de la forma en què està implementat és molt alta i, per tant, fa molt difícil poder diferenciar les classes correctament.

1.1.4. HOGs sobre la imatge en escala de grisos

L'avantatge d'aplicar els HOGs directament sobre la imatge en escala de grisos és que són capaços d'extreure de forma implícita, a part de la informació sobre el contorn, altres dades com la textura, el nombre de nervis, etc.

D'aquesta manera es pot obtenir molta més informació d'una fulla i no fa falta extreure de forma "manual" les característiques que manquen per tal de classificar les classes que semblen més ambigües.

A continuació es poden veure els HOGs de la fulla *I8nr029.tif*, binaritzada a l'esquerra i en escala de grisos a la dreta.

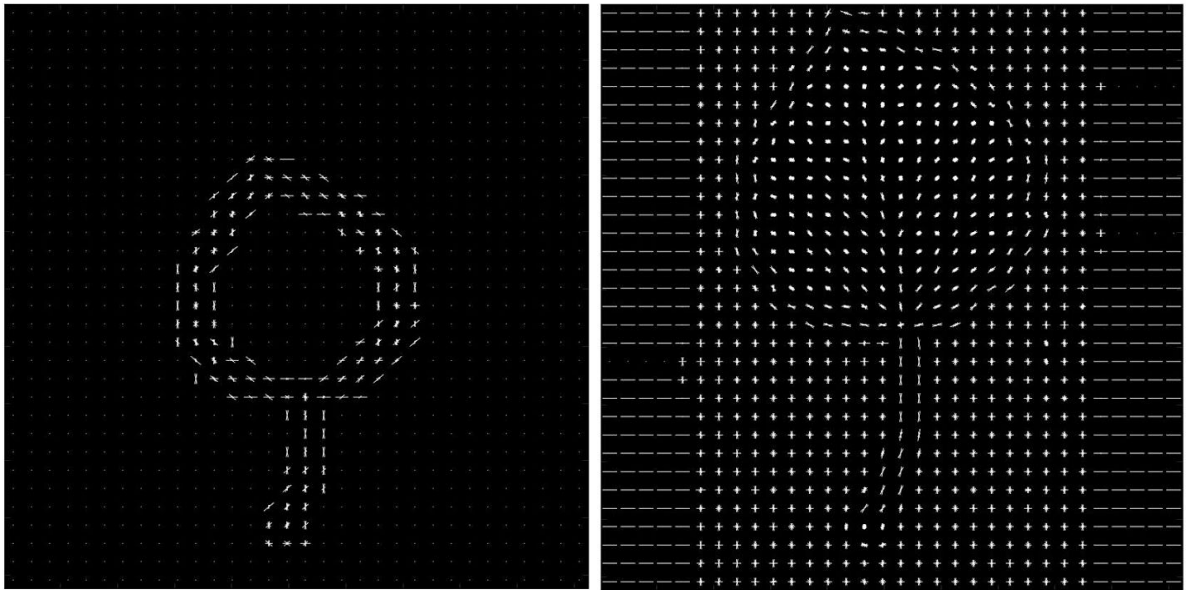

Fig 10. I8nr029.tif

1.2. Algorismes classificadors

Durant el procés de selecció i extracció de característiques, es va fer servir el mateix classificador amb els mateixos paràmetres per tal de poder comparar el rendiment dels diferents descriptors: una SVM amb un kernel lineal. Després d'obtenir un vector de característiques, que es va considerar suficientment bo, es va mantenir constant per tal d'escollir el classificador que funcionés millor en combinació amb aquest.

Per dur a terme una primera exploració, es va fer servir l'aplicació de Matlab *Classification Learner*, que entrena diversos models a la vegada i proporciona el rendiment de cadascun amb diferents mesures: matriu de confusió, corba ROC, etc.

A continuació es mostra una captura de pantalla de l'aplicació.

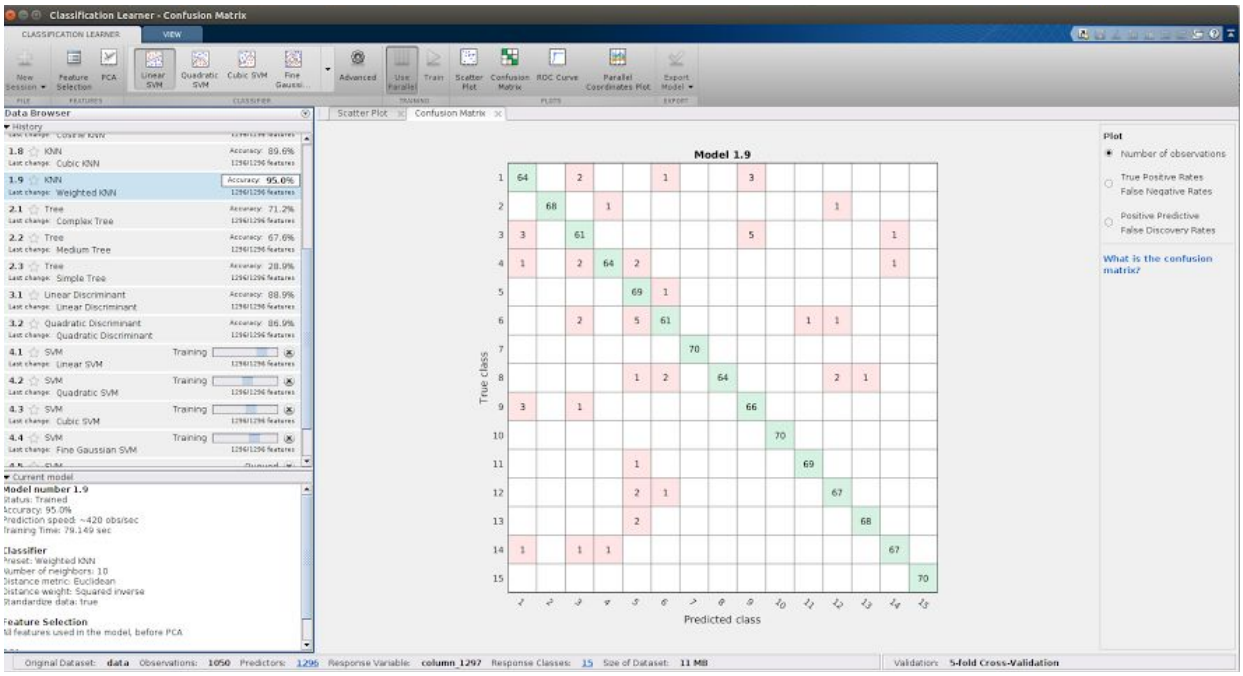

Fig 11. Captura de pantalla de Matlab Classification Learner

Això va permetre descartar de manera ràpida molts classificadors que no tenien un bon rendiment amb el conjunt de dades amb el qual es treballava. Es van provar diferents variants de KNN, arbres de decisió, SVMs, discriminants i meta-classificadors (Bagging i AdaBoost) i, amb cadascun, es va dur a terme una 5-fold cross-validation amb tot el dataset. La precisió màxima va ser d'un 97% per diversos classificadors, i finalment es va decidir seleccionar el classificador més simple que proporcionava aquesta precisió: una SVM amb un kernel lineal. La selecció del classificador més simple va estar motivada per dues raons; per una banda, el temps de còmput a l'hora d'entrenar el model era menor i, per l'altra, hi havia menys risc de tenir sobre-entrenament (overfitting).

Més tard, es va decidir provar una xarxa neuronal ja que avui en dia són l'estat de l'art en classificadors, donant un rendiment molt alt en molts conjunts de dades.

2. Algorisme final

2.1. Selecció de característiques

Com s'ha comentat en apartats anteriors, la gran potència que demostra tenir l'ús del HOGs per descriure el contorn de la fulla, així com la seva textura interior, ha estat, finalment, l'algorisme per extreure característiques.

Per evitar introduir altres característiques que haguessin acabat essent redundants, per la completitud del vector de característiques extret amb HOGs, s'ha decidit no seleccionar més característiques.

2.2. Extracció de característiques

Tal i com s'ha esmentat, per extreure els HOGs de les imatges es fa servir la funció de Matlab *extractHOGFeatures*. Aquesta funció es pot configurar amb diversos paràmetres, que afectaran al nombre de característiques extretes i, per tant, al cost computacional del càlcul del vector de característiques.

Per trobar els paràmetres òptims d'extracció dels HOGs, es va anar canviant cada paràmetre individualment i, després de cada canvi, es va dur a terme una 10-fold cross-validation per veure si les característiques extretes ajudaven a discriminar millor entre classes. Val a dir que el cost computacional del càlcul dels HOGs augmenta proporcionalment amb el valor d'aquests paràmetres, però la precisió del classificador té un límit. Per tant, es va anar augmentant el valor d'aquests paràmetres fins que la precisió del classificador començava a disminuir, i es seleccionava el valor anterior.

Finalment, els paràmetres òptims en el nostre cas van ser els següents. El cost computacional del càlcul del vector de característiques varia linealment respecte el nombre de característiques extretes. Per tant, s'especifica com varia la mida del vector de característiques *N* quan es varia cada paràmetre.

$$N = BlocksPerImage \cdot BlockSize \cdot NumBins$$

$$BlocksPerImage = floor(\frac{\frac{size(I)}{CellSize} - BlockSize}{BlockSize - BlockOverlap} + 1)$$

- *CellSize* = 64. Si disminueix la mida de la cel·la, això vol dir que hi haurà més cel·les, més histogrames i, per tant, el vector de característiques serà més gran. Ara bé, també cal tenir en compte que per calcular l'histograma d'orientació de gradients d'una cel·la, l'algorisme ha de mirar el gradient a cadascun dels píxels de la cel·la. Per tant, com més petita sigui la cel·la, menys costós serà el càlcul de cada histograma.

- *BlockSize* = 2. El valor del gradient dels píxels es normalitza respecte els valors dels píxels de les cel·les del mateix bloc. Com que és complicat descriure la dependència entre la mida del vector de caracterísiques i BlockSize, el que s'ha fet és fer un plot de la funció en Matlab, on a l'eix d'abscisses hi ha BlockSize i a l'eix d'ordenades, N.

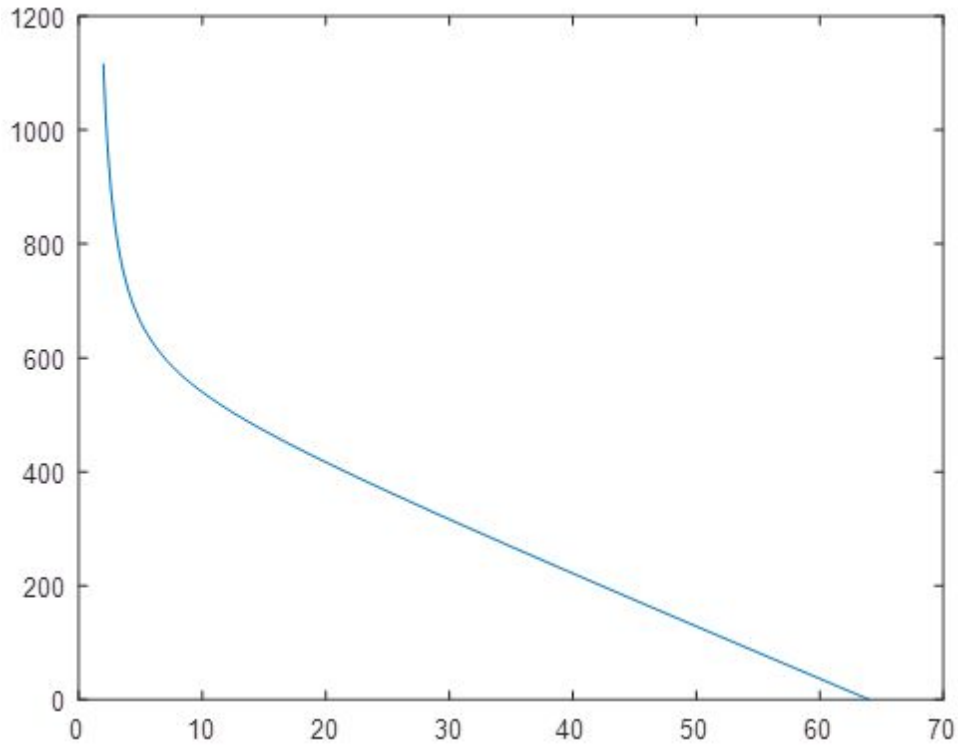

Fig 12. Mida del vector de característiques en funció de BlockSize

- *BlockOverlap* = ceil(*BlockSize*/2) = 1. Un valor gran d'overlapping entre blocs pot capturar més informació, però també produeix un vector de característiques més gran. Concretament, la mida del vector de característiques és quadràtica respecte aquest paràmetre.

- *NumBins* = 9. Cada histograma aporta un nombre de característiques igual al nombre de bins. Per tant, la mida del vector de característiques és lineal respecte aquest paràmetre.

A més, cal tenir en compte la mida de la imatge d'entrada. Si augmenta el nombre de píxels de la imatge, això vol dir que hi haurà més cel·les, més histogrames i, per tant, el vector de característiques serà més gran.

2.3. Algorisme classificador

Després de veure el rendiment de diversos classificadors "clàssics", es va decidir provar un dels classificadors més usats en els últims anys i que millors resultats dóna en problemes de reconeixement d'objectes en imatges: una xarxa neuronal artificial.

En lloc de muntar un sistema de classificadors que resolgués el problema per parts, es va optar per deixar que la capacitat classificadora d'una xarxa neuronal resolgués tot el problema de cop. Observant l'alta precisió obtinguda, una xarxa neuronal basada en perceptrons sembla més que suficient suficient per discriminar entre fulles de diferents classes.

Existeixen molts tipus de xarxes neuronals, però les que implementa Matlab són força senzilles. Concretament, Matlab utilitza un tipus de xarxa neuronal *feedforward* anomenat perceptró multicapa. Així doncs, una xarxa neuronal consta de 3 capes: una capa d'*input*, una capa d'*output* i una *hidden layer* o capa intermèdia. Una capa consta d'una sèrie de neurones, que són les que guarden la informació. Cada capa està connectada completament a la següent, és a dir, totes les neurones d'entrada estan connectades amb totes les neurones de sortida. Cadascuna d'aquestes connexions entre neurones, que es podrien assimilar a les sinapsis del nostre cervell, té un pes associat. Quan es diu que una xarxa neuronal "aprèn", el que aprèn realment és el valor correcte d'aquests pesos.

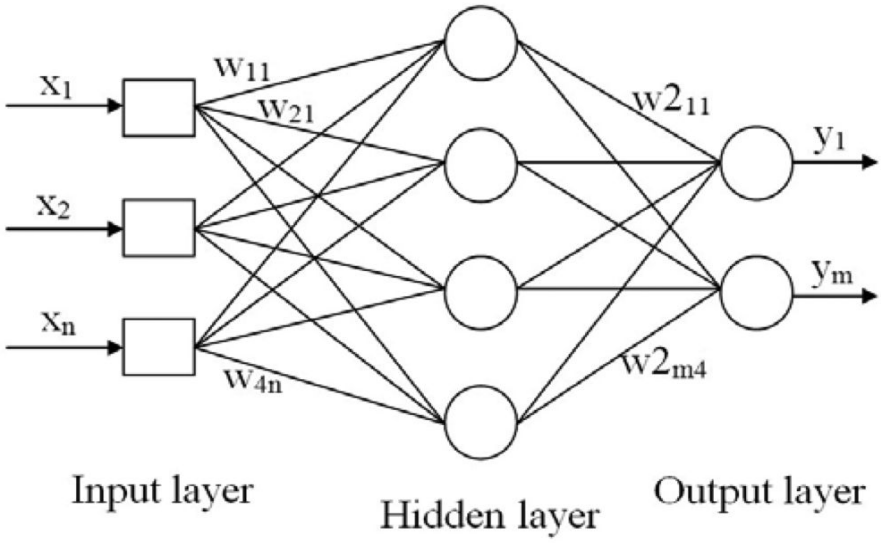

FIg 13. Esquema de l'estructura d'una xarxa neuronal

La capa d'entrada està formada pel vector de característiques d'una fulla. Per tant, aquesta capa tindrà tantes neurones com descriptors (1764 en el nostre cas). La capa intermèdia té una mida variable, que es pot especificar com a paràmetre a l'hora de construir la xarxa (20 en el nostre cas). La mida d'aquesta capa influirà en la precisió final de la xarxa com a classificador. Finalment, la capa de sortida té tantes neurones com classes (15 en el nostre cas). El valor de sortida de cada neurona és la confiança que té la xarxa de que la fulla pertanyi a la classe associada a aquella neurona.

L'algorisme utilitzat per entrenar la xarxa és *backpropagation* fent servir l'*scaled conjugate gradient* com a mètode d'optimització. Aquest algorisme consta de dues fases: propagació (*forward pass*) i actualització dels pesos (*backward pass*). A la primera fase, el vector d'entrada es propaga cap endavant per tota la xarxa, capa per capa, fins que arriba a la capa de sortida. Aleshores, la sortida de la xarxa es compara amb la sortida desitjada, fent servir una funció de cost o *loss function* que permet calcular un valor d'error per cadascuna de les neurones de la capa de sortida. Aleshores, els valors d'error es propaguen cap enrere, començant per la capa de sortida, fins que cada neurona de la xarxa té associat un valor d'error que vindria a representar la seva contribució a la sortida original.

A continuació, l'algorisme fa servir aquests valors d'error per calcular el gradient de la funció de cost respecte els pesos de la xarxa. A la segona fase, aquest gradient es passa com a entrada al mètode d'optimització, en aquest cas l'*scaled conjugate gradient*, que fa servir els

gradients per actualitzar els pesos de la xarxa amb l'objectiu de minimitzar la funció de cost. Aquest procés es repeteix fins que el valor de la funció de cost és suficientment petit.

La importància d'aquest procés és que, un cop la xarxa està entrenada, les neurones de les capes intermèdies s'organitzen de tal manera que diferents neurones aprenen a reconèixer diferents característiques individuals de l'espai total de característiques. Quan entra un patró amb soroll o incomplet, les neurones de la capa oculta respondran amb una sortida activa si l'entrada conté un patró que s'assembla a una característica que les neurones individuals han après a reconèixer. Es creu que aquest és el motiu pel qual aquest classificador té un rendiment superior a la resta de classificadors provats.

2.4. Cost computacional

Primerament, es discutirà el cost general d'executar tot l'algorisme, des de l'extracció de característiques fins a l'entrenament.

Com que el cost d'extreure les característiques ja s'ha explicat a nivell teòric en l'apartat 2.2, aquí simplement es posaran els resultats empírics obtinguts.

Etapa	Temps (s)
Extracció de característiques	338.97
Entrenament	13.67

Pel que fa a la xarxa neuronal, es distingeixen dos tipus de cost computacional pel que fa a la classificació. El cost d'entrenament de la xarxa i el cost de classificar una o més imatges.

La fase de classificació consisteix en un *forward pass*, és a dir, propagar el vector d'entrada per tota la xarxa, capa a capa, fins que arriba a la capa de sortida. A l'hora d'implementar, propagar els valors d'una capa a una altra equival a calcular un producte de matrius. Per un perceptró multicapa, el temps d'execució està dominat pels productes de matrius. Assumint el producte de matrius estàndard, i sent *d* el nombre d'imatges d'entrada, en una sola capa amb dimensió d'entrada *n* i dimensió de sortida *m*, el cost del producte de matrius és $O(nmd)$. Sumant el cost del producte de matrius per cada capa obtenim el cost d'un *forward pass*.

En general, segons la literatura[1], la derivació automàtica inversa que es duu a terme en el *backward pass* és com a molt un factor constant més lenta que el *forward pass*. Tal i com s'ha explicat abans, una iteració de l'algorisme de *backpropagation* consta d'un *forward pass* i d'un *backward pass*. Per tant, el cost d'una iteració de l'algorisme d'entrenament és del mateix ordre que el del *forward pass* calculat anteriorment, $O(nmd)$. Ara bé, el cost total de l'entrenament d'una xarxa neuronal és difícil de calcular, ja que no hi ha cap garantia formal del nombre d'iteracions requerides.

2.5. Resultat de l'aprenentatge i del reconeixement

Un cop entrenat el model de la xarxa neuronal, es comprova el seu rendiment amb el dataset de validació, un subconjunt de les imatges proporcionades (5 de cada classe) que el classificador no ha vist durant el seu entrenament. A continuació es mostra la bondat del classificador mitjançant diferents mesures.

Matriu de confusió

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	4	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

Fig 14. Matriu de confusió

Precisió: 0.973333

[1] https://justindomke.wordpress.com/2009/03/24/a-simple-explanation-of-reverse-mode-automatic-differentiation/

Corba ROC

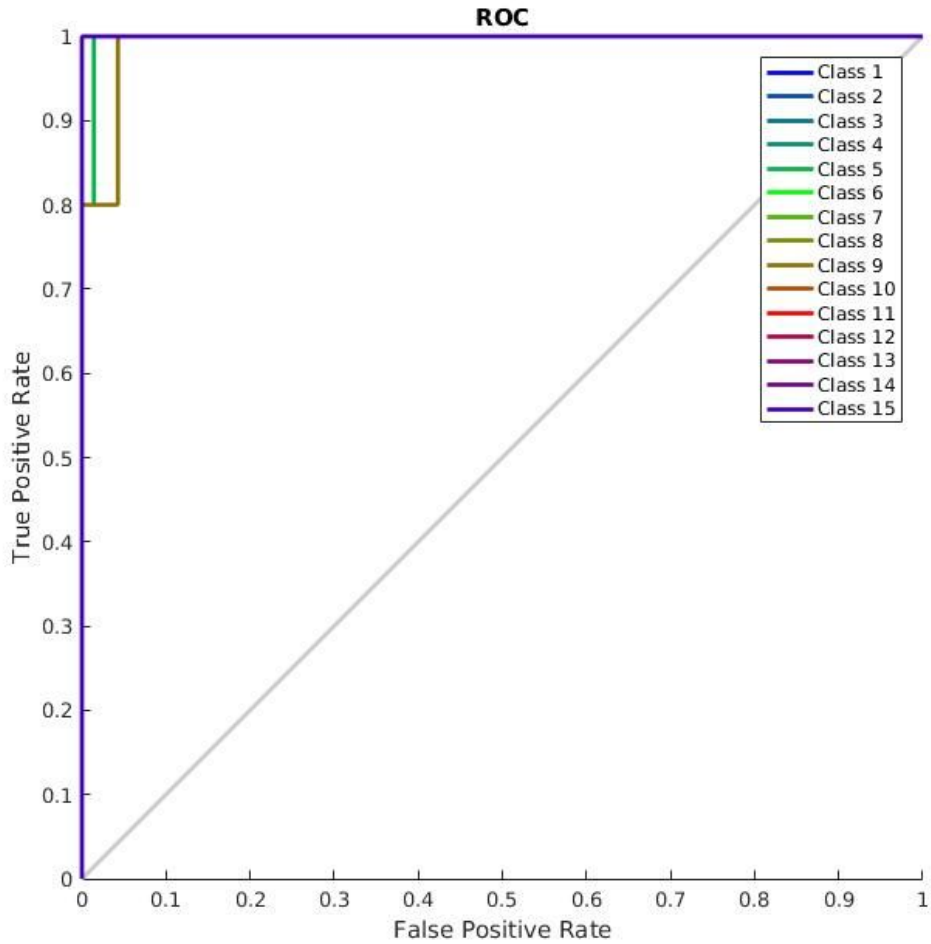

Fig 15. Corba ROC

Precisió i sensitivitat

Classe	Precisió	Sensitivitat
1	1.0	1.0
2	1.0	1.0
3	0.83	1.0
4	1.0	1.0
5	1.0	0.8
6	0.83	1.0
7	1.0	1.0
8	1.0	1.0

9	1.0	0.8
10	1.0	1.0
11	1.0	1.0
12	1.0	1.0
13	1.0	1.0
14	1.0	1.0
15	1.0	1.0

3. Bibliografia

[1] Salve, Manza, Sardesai, Yannawar (2016) Identification of the Plants Based on Leaf Shape Descriptors [en línea] [Consultat el 15 de desembre 2016] https://www.researchgate.net/profile/Pravin_Yannawar/publication/284887430_Identification_of_the_Plants_Based_on_Leaf_Shape_Descriptors/links/56ed04a008aed17d09f69ffb.pdf

[2] Quora (2016) What is the time complexity of backpropagation algorithm for training artificial neural networks? [en línea] [Consultat el 2 de gener de 2017] https://www.quora.com/What-is-the-time-complexity-of-backpropagation-algorithm-for-training-artificial-neural-networks

[3] Wikipedia (2016) Histogram of oriented gradients [en línea] [Consultat el 20 de desembre de 2016] https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

4. Annex

4.1. Implementació dels descriptors de Fourier

Primerament, aquesta implementació dels descriptors de Fourier discretitza la imatge en 360 parts, de forma radial. Cada element discretitzat té el valor del radi corresponent a l'angle amb valor equivalent a la posició que ocupa tal radi en el vector.

Per calcular el valor de cada radi, s'agafa la intersecció de la imatge amb un triangle amb un angle d'obertura de 1º i orientat en l'angle corresponent. Llavors es pondera cada píxel resultant de la intersecció per la inversa de la seva distància respecte el centre de la imatge. Finalment, es fa la suma de tots els píxels ponderats.

Per calcular la component de sin i cos de cada freqüència es fa la multiplicació escalar entre el vector de radis calculat anteriorment i el vector sin o cos, segons sigui el cas, de la freqüència que toca.

```
% Funcio que retorna l'amplitud dels n primers cos (FDc) i sin (FDs) de %
l'aproximació de Fourier del contorn de la imatge bw
function [FDc, FDs] = gfd(bw,n)

% get object informations
state = regionprops(bw,'Centroid','Extrema');

% get the maximal radius maxR
maxR = sqrt((state.Extrema(:,1)-state.Centroid(1)).^2 +..
    (state.Extrema(:,2)-state.Centroid(2)).^2);
maxR = max(maxR);


% meshgrid with origin centered to the image center
N = sz(1);
x = linspace(-N/2,N/2,N);
y = x;
[X,Y] = meshgrid(x,y);

% matrix containing radius of each cell to image center
radius = sqrt(X.^2+Y.^2)/maxR;

% matrix containing angluar of each cell to image center
theta = atan2(Y,X);
theta(theta < 0) = theta(theta <0) + 2*pi;
```

```
% Initialize variables
FR = zeros(m+1,n+1);
FI = zeros(m+1,n+1);
FDc = zeros((m+1)*(n+1),1);
FDs = zeros((m+1)*(n+1),1);

barRadius = zeros(1,360);
auxRadi = zeros(sz);
auxRadi(ceil(sz(1)/2),ceil(sz(2)/2):sz(2)) = 1;
auxRadi = auxRadi./(radius+1);
for i = 1:360
      barRadius(i) = sum(sum(bw.*auxRadi))./sum(sum(auxRadi));
      auxRadi = imrotate(auxRadi,1,'bilinear','crop');
end

Y = zeros(1,360);
for i = 0:n
      c = cos((i*2*pi/360)*[0:359]);
      FDc(i+1) = dot(barRadius,c)/dot(c,c);
      s = sin((i*2*pi/360)*[0:359]);
      if i ~= 0
      FDs(i+1) = dot(barRadius,s)/dot(s,s);
      else
      FDs(i+1) = 0;
      end
      Y = Y + FDc(i+1)*c + FDs(i+1)*s;
end

end
```

```
% Funcio que recompon el contorn d'una imatge a partir dels descriptors %
de Fourier FDc (cos) i FDs (sin)
function [ I ] = recomponImatge(FDc, FDs, numDescr)
      interval = [0:0.01:2*pi];
      f = @(x) @(c) @(s) @(z) c.*cos(x.*z) + .*sin(x.*z);
      f2 = f(0:numDescr-1);
      f3 = f2(FDc(1:numDescr)');
      f4 = f3(FDs(1:numDescr)');
      radi = sum(f4(interval')');
      X = radi.*cos(interval);
      Y = radi.*sin(interval);
      axis equal
      plot(X,Y)
      axis equal
end
```

4.2. Implementació de la quantificació de la textura

```
function textura = difTextura(Igray, bw, sbl, dsk)
% Igray -> imatge en escala de grisos
% bw -> imatge binaritzada
% sbl -> fspecial('sobel')
% dsk -> strel('disk',15)
      bw = imerode(bw,dsk);
      Isobel =
sqrt(double(imfilter(Igray,sbl)).^2+double(imfilter(Igray,sbl')).^2);
      Isobel = Isobel.*double(bw);
      textura = sum(sum(bw))/sum(sum(Isobel));
end
```

4.3. Implementació del comptatge de nervis

```
function numNervis = comptaNervis(I,bw)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
      dsk = strel('disk',25);
      bw = imerode(bw,dsk);
      sbl = fspecial('sobel');
      Isobel =
sqrt(double(imfilter(I,sbl)).^2+double(imfilter(I,sbl')).^2);
      I2 = Isobel.*double(bw);
      gauss = fspecial('gaussian',7,3);
      I2 = imfilter(I2,gauss);
      bwLoc = localBinarization(I2,6,15,15);
      bwLoc = bwareaopen(bwLoc,20);
      nervis = bwLoc;
      nervis(:,1:250) = 0;
      nervis(:,260:512) = 0;
      comp = bwconncomp(nervis)
      % imshow(bwLoc)
      numNervis = comp.NumObjects;
end
```

4.4. Codi final: HOGs i xarxa neuronal

```
% HOGs parameters
CellSize = [64 64];
BlockSize = [2 2];
BlockOverlap = ceil(BlockSize/2);
NumBins = 9;

I = imread('../data/leaf1/l1nr001.tif');
I = square(I);
I = imresize(I,[512 512]);
I = rgb2gray(I);
features =
extractHOGFeatures(I,'CellSize',CellSize,'BlockSize',BlockSize,'BlockOverl
ap',BlockOverlap,'NumBins',NumBins);
hogFeatureSize = length(features);


%% feature extraction

files = dir('../data/leaf*/*.tif');
n = length(files);

inputs = zeros(hogFeatureSize,n);
targets = zeros(15,n);

tic
for i=1:n
    folder = files(i).folder;
    fn = files(i).name;

    I = imread(strcat(folder,'/',fn));
    I = square(I);
    I = imresize(I,[512 512]);
    I = rgb2gray(I);

    [featureVector,hogVisualization] =
extractHOGFeatures(I,'CellSize',CellSize,'BlockSize',BlockSize,'BlockOverl
ap',BlockOverlap,'NumBins',NumBins);
    inputs(:,i) = featureVector';
    plot(hogVisualization); drawnow;

    class = int32(str2num(fn(2:end-9)));
    targets(class,i) = 1;

    fprintf('Extracting features of %s\n',fn);
end
toc
close all;
```

```
%% train

trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network
hiddenLayerSize = 20;
net = patternnet(hiddenLayerSize, trainFcn);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 0/100;

% Train the Network
[net,tr] = train(net,inputs,targets);
nntraintool('close');

% Test the Network
outputs = net(inputs);
[c,cm] = confusion(targets,outputs);
fprintf('\nTest confusion matrix:\n\n');
disp(cm);
fprintf('accuracy: %f\n',1-c);


%% test

files = dir('../data/test/*.tif');
n = length(files);

truelabels = zeros(15,n);
classout = zeros(15,n);

for i=1:n
    folder = files(i).folder;
    fn = files(i).name;

    I = imread(strcat(folder,'/',fn));
    I = square(I);
    I = imresize(I,[512 512]);
    I = rgb2gray(I);

    featureVector =
extractHOGFeatures(I,'CellSize',CellSize,'BlockSize',BlockSize,'BlockOverl
ap',BlockOverlap,'NumBins',NumBins);
    classout(:,i) = net(featureVector');

    class = int32(str2num(fn(2:end-9)));
    truelabels(class,i) = 1;
```

```
    fprintf('Predicting class of %s\n',fn);
end

[c,cm] = confusion(truelabels,classout);
fprintf('\nValidation confusion matrix:\n\n');
disp(cm);
fprintf('accuracy: %f\n',1-c);
plotroc(truelabels,classout);
```

Aquest codi depèn d'una funció no inclosa a Matlab: *square*. Aquesta funció s'encarrega de crear una imatge quadrada a partir de la imatge d'entrada afegint les files o columnes necessàries.

```
function I = square(I)
    %adds coloums/rows of zeros to create a square image
    sz = size(I);
    if sz(1) < sz(2)
        temp = (sz(2) - sz(1))*0.5;
        if mod(temp,1) > 0
            I = padarray(I,[0 1],'replicate', 'post');
        end
        I = padarray(I,[round(temp) 0],'replicate');
    elseif sz(2) < sz(1)
        temp = (sz(1) - sz(2))*0.5;
        if mod(temp,1) > 0
            I = padarray(I, [1 0],'replicate', 'post');
        end
        I = padarray(I,[0 round(temp)],'replicate');
    end
end
```