

Pràctica VC

Classificador de Series



Guillem Cabré
Jordi Calsina

Spring 2024-25

Índex

1	Introducció	3
2	Classificador de series	4
2.1	Metodologia	4
2.1.1	Extreure característiques	4
2.1.2	Creació de taula de característiques	4
2.2	Característiques de classificació	5
2.2.1	Descriptor HSV	5
2.2.2	Descriptor d'histogrames de color	6
2.2.3	Descriptor de densitat de vores	7
2.2.4	Descriptor de textura	10
2.2.5	Descriptor de característiques geomètriques	11
2.3	Fine tuning	12
2.4	Model Final	13
2.5	Intent de detecció d'outliers	14
2.6	Funcionament de la aplicació	15
2.7	Conclusions	16
3	Detecció de personatges	18
3.1	Algorisme	18
3.2	Obtenció de més imatges	19
3.3	Obtenció del TRAIN	20
3.4	Descriptor utilitzats	21
3.4.1	Descriptor d'histogrames normalitzats de HSV	21
3.4.2	Descriptor de densitat de vores	21
3.4.3	Descriptor LBP	22
3.4.4	Anàlisi de significança	22
3.4.5	Justificació de l'elecció	23
3.5	Classificador utilitzat	23
3.6	Resultats obtinguts	25

1 Introducció

Aquest projecte de Visió per Computador té com a objectiu desenvolupar un sistema capaç de classificar automàticament imatges relacionades amb sèries d'animació. El treball s'ha estructurat en dues parts diferenciades:

- **Classificació de sèries:** donada una imatge, el sistema ha de determinar a quina de les 8 classes de diferents sèries animades pertany: *Barrufets, Bob Esponja, Gat i Gos, Gumball, Hora de Aventuras, Oliver y Benji, Padre de família, Pokemon, Southpark i Tom i Jerry*.
- **Detecció de personatges:** en una segona fase, el projecte s'ha centrat en abordar la detecció específica de personatges dins les imatges. En el nostre cas, ens hem focalitzat en el personatge de *Bob Esponja*, amb l'objectiu de localitzar-lo dins d'una imatge independentment del context o la posició.

Aquest plantejament ens ha permès treballar tant en classificació general d'imatges com en detecció més fina d'elements concrets dins d'una escena. A través de diverses iteracions hem desenvolupat un sistema robust que combina tècniques de classificació supervisada, extracció de descriptors visuals i una interície simple però funcional per a l'usuari.

2 Classificador de series

Per començar, la primera part de la pràctica consisteix en: donada una imatge o un conjunt d'imatges classificar-les en les 8 diferents classes. Aquestes són:

- Barrufets
- Oliver y Benji
- Bob Esponja
- Padre de familia
- Gat i Gos
- Pokemon
- Gumball
- Southpark
- Hora de Aventuras
- Tom y Jerry

A part també s'ha considerat una classe extra **outlier**, que se li assignarà a aquelles imatges que no pertanyin a cap de les classes anteriors. Per exemple, si al programa se li adjunta una imatge d'una persona, la hauria de marcar com a **outlier**, ja que no forma part de cap de les classes.

2.1 Metodologia

Primerament havíem de separar el conjunt de *TRAIN*, que conté les imatges d'entrenament, en dos subconjunts *TRAIN* i *TEST*, el segon té imatges per provar el model. El que vam fer es moure un 30% de les imatges de *TRAIN* a *TEST*, per d'aquesta manera analitzar el rendiment del nostre model amb imatges independents a les fetes servir a la fase d'entrenament. Això ho vam aconseguir amb un script de *Python* (*split_train_test.py*), que feia bàsicament la tasca que acabem d'explicar. Per cada classe d'imatges s'agafa un 30% de les imatges de forma aleatòria i es mouen a la carpeta *TEST*. El directori de imatges de test té un total de 465 imatges, i el d'entrenament 1099.

2.1.1 Extreure característiques

Per extreure les característiques numèriques de cada imatge i preparar-les per ser utilitzades en el classificador de sèries, fem servir la següent funció:

```

1 function row = extreureCaracteristiques(I)
2     IMAGE_SIZE = 128;
3     I = imresize(I, [IMAGE_SIZE, IMAGE_SIZE]);
4     Ihsv = rgb2hsv(I);
5
6     % Calcular les caracteristiques
7     meanS = mean(Ihsv(:,:,2), 'all');
8     sdS = std(Ihsv(:,:,2), 0, 'all');
9     % ...
10
11    % Montar un struct amb les dades aconseguides
12    row = struct();
13    row.meanS = meanS;
14    row.sdS = sdS;
15    % ...
16 end

```

Listing 1: Funció per extreure característiques d'una imatge.

Aquesta funció redimensiona la imatge a una mida fixa ($128p \times 128p$) i calcula diverses característiques com la mitjana i desviació estàndard de components del color en HSV, un histograma de colors... A continuació s'explicaran les característiques. Podeu veure el contingut d'aquest fitxer en detall en el fitxer anomenat **extreureCaracteristiques.m**.

2.1.2 Creació de taula de característiques

El següent fragment de codi recorre totes les imatges del conjunt *TRAIN*, extreu les seves característiques i construeix una taula preparada per entrenar el classificador:

```

1 rows = [];
2
3 % Itera per totes les classes d'imatges
4 for i = 1:length(traindirs)
5   name = traindirs(i).name;
6   if startsWith(name, '.'); continue; end
7
8 % Itera per totes les imatges
9 files = dir(fullfile(traindir, name, '*.jpg'));
10 for j = 1:length(files)
11   imgPath = fullfile(files(j).folder, files(j).name);
12   I = imread(imgPath);
13
14   row = extreureCaracteristiques(I);
15   row.Label = string(name);
16
17 % Annexa la informació de la imatge a la llista
18 rows = [rows; row];
19 end
20
21 % Converteix la llista d'structs a una taula
22 T = struct2table(rows);
23

```

Listing 2: Creació de la taula de característiques del conjunt TRAIN.

Aquest procés genera una taula on cada fila conté les característiques extretes d'una imatge i la seva etiqueta associada (la seva classe). Al mateix fitxer `classifier.m`, es disposa d'un codi pràcticament idèntic per generar la taula de característiques corresponent al conjunt TEST, que s'utilitza posteriorment per fer la classificació i avaluar els resultats. Vegeu el fitxer sencer a `classifier.m`.

2.2 Característiques de classificació

A continuació s'exposaran les característiques amb les quals s'entrena i es prediu en el nostre model. Recorrem les característiques de forma iterativa i evaluarem l'impacte d'aquestes. Les exposarem en les diverses iteracions que ha tingut el desenvolupament del projecte.

2.2.1 Descriptors HSV

Inicialment, per fer un esbós, vam fer servir simplement la desviació estàndard i la mitjana de la saturació de la imatge, i el mateix amb el camp *value*, que representa la lluminació del píxel. Per fer això vam convertir la imatge en la seva representació en format *HSV*, i posteriorment vam extreure cada un dels camps per tractar-los per separat. Vam ignorar el camp *Hue*, perquè és cíclic i no ens donaria bons resultats perquè si suposem que $Hue \in [0, 1]$, aleshores els valor que s'apropin als dos extrems del rang seran iguals. Tot i que es podria fer una transformació per tal de convertir aquest valor cíclic en dues components lineals mitjançant les funcions $\cos(2\pi \cdot Hue)$ i $\sin(2\pi \cdot Hue)$, hem decidit no incloure aquest camp en el nostre cas. El motiu és que considerem que el tot de color global de la imatge no té gaire impacte en la classificació que volem fer, ja que els fotogrames d'una sèrie rarament tenen un color dominant clar i tampoc sembla que hagi de seguir cap tendència coherent al llarg del temps.

A partir d'això vam generar el nostre primer experiment mitjançant l'app *Classification Learner* de *Matlab*. Els resultats que vam obtenir no eren gaire bons però ja ens van ajudar a guiar el nostre projecte. Veieu els resultats a continuació:

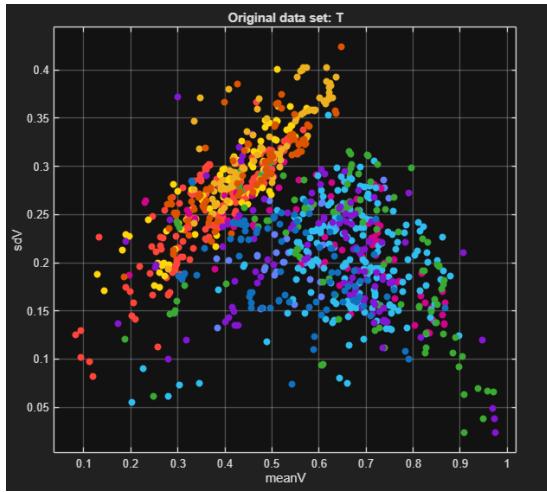


Figura 1: Gràfic de dispersió de la primera iteració

		Test Confusion Matrix for Model 2.16									
		barufets	bob-esponja	gat+gos	gumball	hora-de-adventuras	oliver-y-benji	padre-de-familia	pokemon	southpark	tom-y-jerry
True Class	Predicted Class	33	31	3	1	30	2	6	1	1	1
	barufets	33	31	3	1	30	2	6	1	1	1
bob-esponja	31	33	3	1	30	2	6	1	1	4	
gat+gos	3	1	34	1	30	2	6	1	2	3	
gumball	1	30	2	6	1	34	8	2	1	1	
hora-de-adventuras	3	1	34	8	2	1	34	8	1	1	
oliver-y-benji	1	10	3	60	8	3	60	8	1	1	
padre-de-familia		1	8	26	1	8	26	1	1	1	
pokemon	2	4	1	1	1	26	1	26	1	1	
southpark		1	1	1	1	1	1	16	16	1	
tom-y-jerry	3	2	2	1	1	1	1	1	1	67	

Figura 2: Matriu de confusió de la primera iteració

Ara explicarem els resultats. Per verificar que els resultats no eren linealment dependents vam mirar les diferents combinacions del *Scatter Plot*, el resultat que no voldríem esperar seria aconseguir una tendència lineal. Veiem com en el nostre cas no és així. Cada punt de color, és una imatge de test de classes diferents (per exemple, groc representa Gat i Gos). Per altra banda, la *Confusion Matrix*, o matriu de confusió, ens mostra en la diagonal les vegades que el model ha encertat la predicció, i els altres valors ens diuen quants cops la imatge pertany a una classe x (*True Class*) i el model l'ha predit erròniament en una classe y (*Predicted Class*). En aquest experiment es va fer servir el model KNN amb $N = 1$. Amb les imatges de test vam aconseguir el resultat de la matriu.

Evidentment no és un resultat òptim ja que té una precisió del 76.6%, però ja és un bon punt de partida.

2.2.2 Descriptors d'histogrammes de color

Una altra idea era fer servir l'histogramma de color de la imatge. En aquest punt vam considerar-ho. Sembla una bona opció ja que l'apartat visual de cada sèrie influeix en l'histogramma, i aquest és únic per a cada una.

Per aconseguir això, el que vam fer va ser treure la il·luminació de la imatge dividint d'aquesta manera $r = \frac{R}{R+G+B}$, $g = \frac{G}{R+G+B}$ i $b = \frac{B}{R+G+B}$. Passant de l'espai RGB a rgb. D'aquesta manera no ens afecten els canvis d'il·luminació, així les imatges una mica més a les fosques donen resultats similars a imatges amb llum. A més, veiem que podem fer una optimització prescindint d'un dels tres canals. Anem a posar l'exemple del color blau b . Veiem que $b = 1 - r - g$. Per tant, generarem els histogrammes amb només dos canals.

Un dels factors a tenir en compte a l'hora de generar l'histogramma és el número de *buckets* (o cubells) que farem servir. Com que els píxels de la imatge estan compostos per tuples de tres colors, cada un d'aquest estarà entre els valors 0 i 255, per tant haurem de discretitzar encara més el rang per tenir informació més estable. Vegeu a continuació a la Figura 3 una col·lecció de matrius de confusió, que ens expliquen els resultats per diferents nombres de *buckets*.

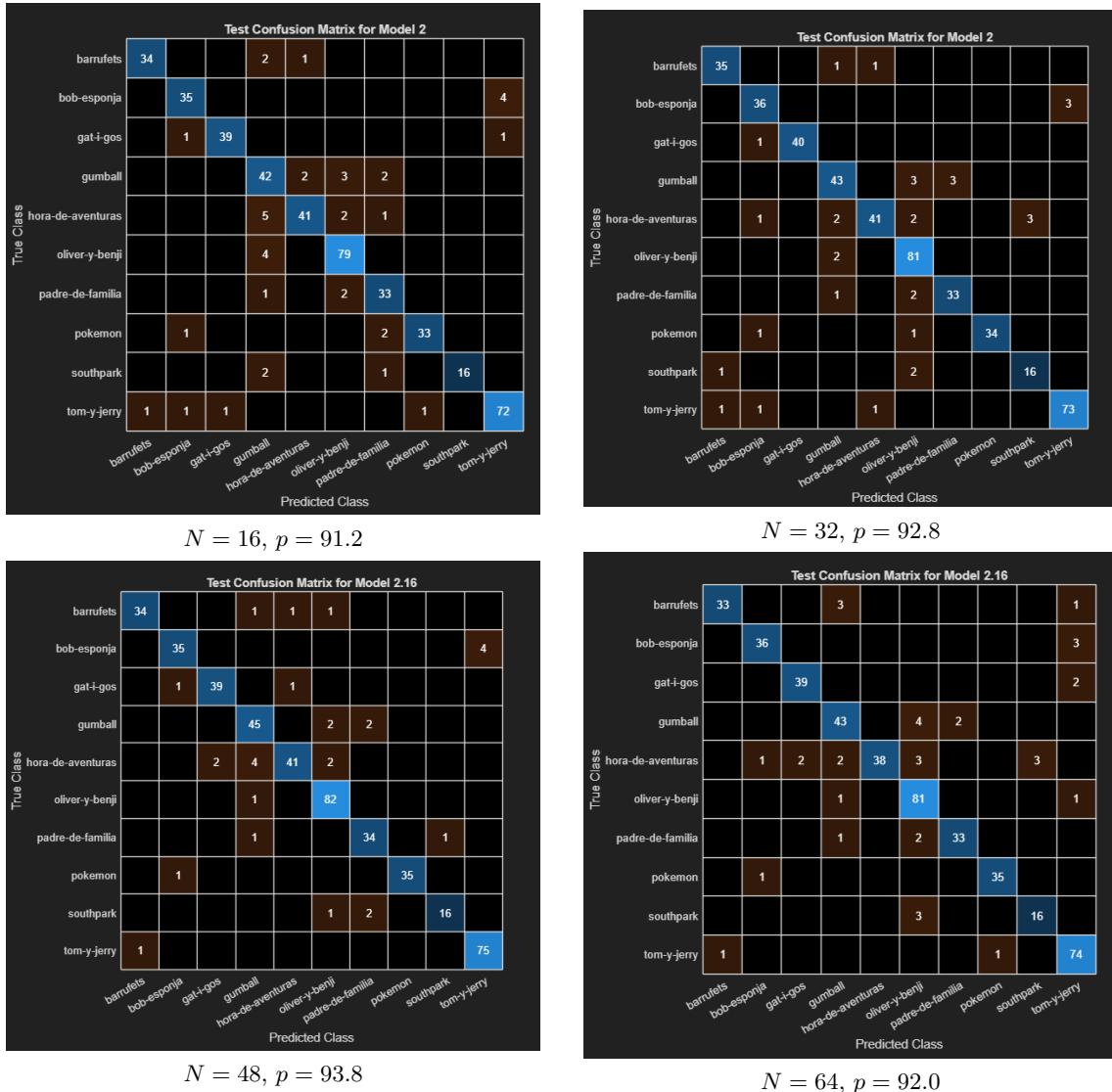


Figura 3: Matrius de confusió de la segona iteració per diferents valors de N , sigui N el nombre de buckets del hisotgrama. p és la presició aconseguida.

Amb aquestes matrius de confusió veiem que el nombre de *buckets* que ens dóna un millor resultat és quan $N = 48$, per tant procedirem amb aquesta característica tunejada d'aquesta manera.

2.2.3 Descriptor de densitat de vores

Una altra característica que ens pot interessar és fer ús de la densitat de vores de la imatge. Per fer això farem servir l'algorisme de Canny¹.

En aquest cas, no és necessari ajustar els paràmetres de l'algorisme de Canny, ja que l'objectiu no és obtenir un contorn òptim per a cada imatge, sinó extreure una mesura relativa de la quantitat de vores. Com que utilitzem la densitat de vores com una característica comparativa entre imatges, l'ús dels valors per defecte permet mantenir la coherència entre les. L'algorisme de Canny treballarà sobre la imatge en blanc i negre, i posteriorment s'agafarà la mitjana de píxels de contorn amb la funció `mean`. Vegeu a continuació el mètode per calcular la densitat de vores de la imatge.

¹ `function edgeDensity = getEdgeDensity(I)`

¹ El mètode de Canny detecta vores suavitzant la imatge amb un filtre Gaussià, calculant el gradient d'intensitat i direcció, i aplicant una supressió de no-màxims seguida d'uns llindars amb histèresi per definir les vores finals.

```

2 % Converteix la imatge a escala de grisos i detecta les vores amb
   Canny
3 edges = edge(rgb2gray(I), 'Canny');
4 % Calcula la densitat de vores com la mitjana dels valors booleans
5 edgeDensity = mean(edges, 'all');
6 end

```

Listing 3: Funció per calcular la densitat de vores d'una imatge.

Per tal de mostrar la rellevància del descriptor de densitat de vores, hem realitzat una prova. Donades dues imatges de sèries diferents (*Hora de Aventuras* i *Oliver i Benji*), hem aplicat l'algorisme de Canny per extreure'n les vores. A continuació, hem calculat la densitat de vores per a cada imatge. A la Figura 4, es poden observar a l'esquerra les imatges originals i a la dreta les corresponents imatges amb les vores detectades. És evident que l'aspecte visual de les dues sèries és molt diferent, i això es reflecteix clarament en el resultat de l'extracció de vores.

Els valors de densitat obtinguts són: 0.0392 per *Hora de Aventuras* i 0.0806 per *Oliver i Benji*. Aquesta diferència és significativa i demostra que la densitat de vores pot actuar com un descriptor útil per distingir entre estils visuals de diferents sèries.

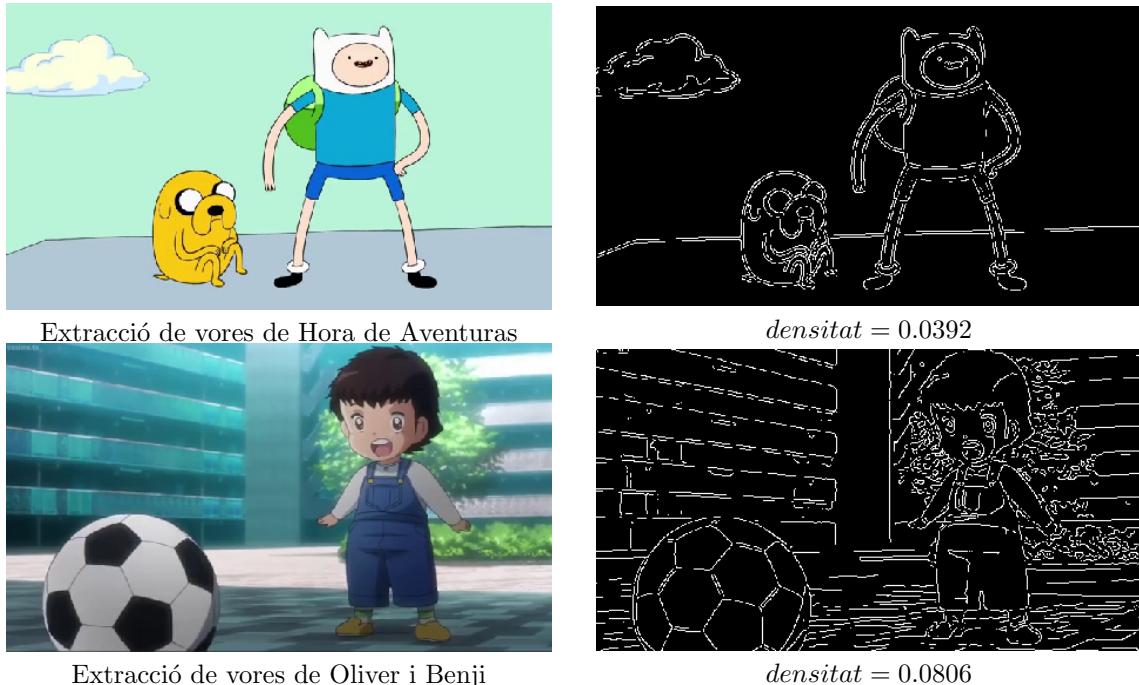


Figura 4: Comparació de les diferents densitats de vores de diferents sèries.

Un cop afegit aquest descriptor, un altre cop farem servir *Classification Learner* per experimentar amb els resultats. Vegeu a continuació els resultats obtinguts.

Test Confusion Matrix for Model 2.16									
True Class	barrufets	34			2	1			
	bob-esponja		35						4
	gat-i-gos			39		1			1
	gumball		1		44		2	2	
	hora-de-aventuras			2	3	42	1		1
	oliver-y-benji				1		82		
	padre-de-familia				1			34	1
	pokemon		1					35	
	southpark						2		17
	tom-y-jerry	1							75

Figura 5: Matriu de confusió de la tercera iteració

Un cop tenim el nostre model entrenat amb les imatges de TRAIN, quan executem el model amb la col·lecció de TEST, aconseguim una precisió màxima en KNN amb $N = 1$. Aquesta precisió és del 94.0%, que és una mica superior que la que vam aconseguir en la segona iteració.

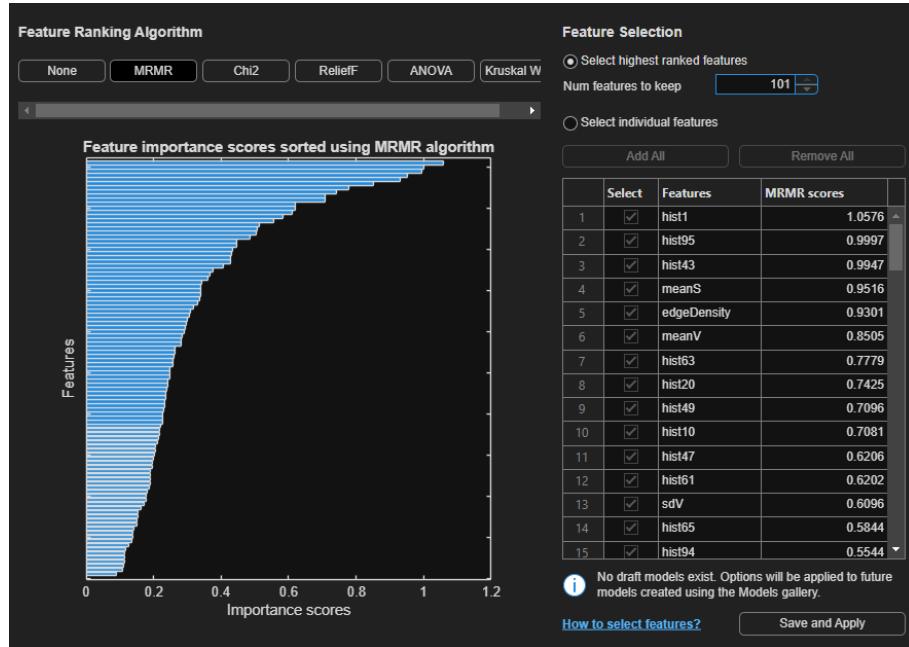


Figura 6: Puntuació d'importància de les característiques obtinguda amb l'algorisme MRMR.

Per tal de valorar quines característiques són més rellevants per al procés de classificació, hem fet servir l'algorisme MRMR (*Minimum Redundancy Maximum Relevance*). Aquest mètode assigna una puntuació a cada característica en funció de dues propietats: la seva rellevància respecte a la variable objectiu (en aquest cas, la classe de la imatge) i la seva redundància respecte a la resta de característiques. D'aquesta manera, MRMR prioritza característiques que aporten informació nova i útil per a la classificació.

A la Figura 6 es pot veure el resultat d'aplicar MRMR, on les característiques s'han ordenat segons la seva importància. En aquest gràfic es pot observar que la característica `edgeDensity` apareix en cinquena posició, confirmant que contribueix significativament al model.

2.2.4 Descriptors de textura

En aquesta quarta iteració hem volgut incorporar informació sobre la textura de les imatges. Per fer-ho, hem calculat diferents descriptors de textura a partir de la matriu de co-ocurrència de nivells de gris (GLCM, de l'anglès *Gray Level Co-occurrence Matrix*). Aquests descriptors inclouen el contrast, la correlació, l'energia i l'homogeneïtat de la textura. Vegeu a continuació el codi utilitzat per obtenir aquestes característiques:

```

1 function tex = getTextureFeatures(I)
2     Igray = im2gray(I);
3     % Generar matrius GLCM amb diferents desplaçaments
4     offsets = [0 1; -1 1; -1 0; -1 -1];
5     glcm = graycomatrix(Igray, 'Offset', offsets, 'Symmetric', true);
6     stats = graycoprops(glcm, {'Contrast', 'Correlation', 'Energy', ...
7         'Homogeneity'});
8     % Fer la mitjana de les mesures per obtenir valors globals
9     tex = struct(... ...
10        'contrast', mean(stats.Contrast), ...
11        'correlation', mean(stats.Correlation), ...
12        'energy', mean(stats.Energy), ...
13        'homogeneity', mean(stats.Homogeneity) ...
14    );
end

```

Listing 4: Funció per extreure característiques de textura mitjançant GLCM.

El mètode genera quatre matrius de co-ocurrència per a diferents orientacions (0° , 45° , 90° , 135°), de manera que es capta la textura en diverses direccions. Hem agafat orientacions menors de 180° perquè quan cridem a `graycomatrix`, posem la flag `Symmetric`, per tant tots els angles simètrics a aquells també es faran servir. A partir d'aquestes, es calculen propietats típiques de la GLCM i se'n fa la mitjana per obtenir valors globals independents de l'orientació. Aquestes mesures prenenen captar patrons de repetició, rugositat i estructura dins de la imatge.

Un com tenim la característica a les taules, fem ús altra vegada de *Classification Learner*, entrenem el model i el provem amb TEST. I obtenim un 94.4% de precisió. Vegeu la matriu de confusió a continuació.

Test Confusion Matrix for Model 2									
True Class	barrufets	33			3				1
	bob-esponja		36						3
	gat-i-gos			39		1			1
	gumball				1	45		2	1
	hora-de-adventuras				2	2	42	1	1
	oliver-y-benji						83		
	padre-de-familia							34	1
	pokemon		1					35	
	southpark						1	1	17
	tom-y-jerry	1							75

Figura 7: Matriu de confusió de la quarta iteració

Quan analitzem el gràfic MRRM, veiem que aquests descriptors no tenen gaire rellevància. Tot i això els farem servir, ja la correlació de textura està en la vintena posició més rellevant. I la homogeneïtat de textura en la posició 35. Per tant, no tenen un paper força important, i son prescindibles.

2.2.5 Descriptors de característiques geomètriques

En aquesta iteració, explorem l'ús de característiques geomètriques per representar la forma dels objectes presents a les imatges. Per fer-ho, utilitzem una segmentació binària mitjançant l'algorisme d'Otsu i calculem diverses propietats morfològiques de les regions segmentades. A continuació es mostra la funció utilitzada per obtenir aquestes característiques:

```

1 function shp = getShapeFeatures(I)
2     Igray = im2gray(I);
3     % Segmentació per llindar d'Otsu
4     level = graythresh(Igray);
5     BW = imbinarize(Igray, level);
6     % Neteja: omplir forats i eliminar components petits
7     BW = imfill(BW, 'holes');
8     BW = bwareaopen(BW, 50);
9     % Propietats geomètriques
10    props = regionprops(BW, 'Area', 'Eccentricity', 'Extent', 'Solidity');
11    if isempty(props)
12        % Si no hi ha regions, assignar zeros
13        shp = struct('area', 0, 'eccentricity', 0, 'extent', 0, 'solidity', 0)
14        ;
15    else
16        % Mitjana de cada propietat
17        shp = struct(...,
18            'area', mean([props.Area]), ...
19            'eccentricity', mean([props.Eccentricity]), ...
20            'extent', mean([props.Extent]), ...
21            'solidity', mean([props.Solidity]) ...
22        );
23    end
end

```

Listing 5: Funció per obtenir característiques geomètriques de forma.

Les propietats geomètriques utilitzades són:

- **Area:** l'àrea total de la regió segmentada.
- **Eccentricity:** una mesura de com d'allargada és la regió.
- **Extent:** la proporció entre l'àrea i la capsula mínima contenidora.
- **Solidity:** la proporció entre l'àrea i la seva convex hull.

Un cop afegides aquestes característiques al conjunt de descriptors, tornem a entrenar el model amb *Classification Learner*. Els resultats obtinguts amb la col·lecció de TEST indiquen una precisió màxima del 93.8% amb el classificador KNN amb $N = 1$, un valor lleugerament inferior al de la quarta iteració.

Test Confusion Matrix for Model 3										
True Class	barrufets	34			2					1
	bob-esponja		35							4
	gat-i-gos			39		1				1
	gumball				46		1	2		
	hora-de-aventuras		2	1	1	41	3	1		
	oliver-y-benji				1		82			
	padre-de-familia				1		1	34		
	pokemon		1						35	
	southpark						2			17
	tom-y-jerry	1	2							73
	barnieis									

Figura 8: Matriu de confusió de la quarta iteració

Tot i que el rendiment global ha disminuït lleugerament respecte a la iteració anterior, hem decidit conservar aquestes característiques en el conjunt final. La raó és que, segons l'anàlisi realitzat amb l'algorisme MRMR, les característiques de forma poden ser útils per al procés de selecció de descriptors rellevants. Per exemple, la característica `shapeExtent` apareix a la posició 16 del rànquing MRMR, cosa que indica que, malgrat la seva contribució limitada al rendiment global en aquesta etapa, pot ser útil quan es combini amb altres descriptors en fases posteriors.

Així doncs, seguim refinant el conjunt de característiques candidates, tenint en compte no només el rendiment puntual, sinó també el potencial de complementarietat entre descriptors.

2.3 Fine tunning

Un cop escollits els nostres classificadors, altra vegada, farem ús de la aplicació *Classification Learner* per elegir quin model serà l'encarregat de la classificació, tunejar els paràmetres d'aquest i també fer un filtratge dels descriptors. Ja que, com és ben sabut, no sempre tenir molts descriptors ens donarà un millor resultat.

Per començar executarem tots els models de classificadors, que estan a la pestanya *All*. Acte seguit, els provarem amb el conjunt d'imatges de **TEST**.

Classificador	Precisió (%)
Tree: fine tree	80.2
Linear Discriminant	85.6
SVM: linear SVM	89.7
SVM: quadratic SVM	92.5
SVM: cubic SVM	93.1
KNN: fine KNN	93.8
KNN: medium KNN	79.1
KNN: cosine KNN	80.6
KNN: weighted KNN	88.2
Ensamble: bagged trees	90.3
Ensamble: subspace KNN	91.4
Neural Network: medium neural network	93.5

Taula 1: Precisió dels diferents models de classificació

Hem ressaltat 3 dels models que són els que millors resultats ens donen. Aquests models són:

- **KNN: fine KNN:** classifica segons els veïns més propers, amb distància euclidiana i pocs veïns (en aquest cas només 1).

- **SVM: cubic SVM:** separa les dades amb un hiperplà no lineal mitjançant una funció cubica.
- **Neural Network: medium neural network:** model amb diverses capes ocultes per aprendre patrons complexos.

Com que aquests són els models que millors resultats ens han donat, ara examinarem què passa quan ajustem paràmetres. El primer que farem és veure com afecta la reducció del nombre de descriptors. Començarem per 109 descriptors, que són tots els que genera el nostre programa i anirem traient aproximadament de 20 en 20, per analitzar quina tendència segueix. Vegeu a continuació la taula.

Classificador	Nombre de descriptors				
	109 (tots)	90	70	50	30
SVM: cubic SVM	93.8	93.5	93.5	94.4	91.8
KNN: fine KNN	93.1	93.3	95.1	93.8	93.1
Neural Network: medium neural network	93.5	91.4	91.0	91.6	91.4

Taula 2: Precisió dels models seleccionats segons el nombre de descriptors utilitzats

Veiem com el model que ens dona millors resultats és el KNN (*K nearest neighbours*) amb la variant *fine KNN*. A priori sembla que els resultats són excel·lents amb les imatges de TEST, però donem-nos compte de com funciona KNN. El que fa aquest algorisme és predir de la següent manera: donades les característiques d'una imatge a predir, busca els k exemples més propers dins el conjunt d'entrenament (segons una mètrica de distància, com ara la distància euclidiana) i assigna l'etiqueta més comuna entre aquests.

Aquest mecanisme fa que el model sigui molt sensible a les dades d'entrenament. En particular, la variant *fine KNN*, que empra un valor petit de K (com ara $K = 1$), pot memoritzar massa bé les dades d'entrenament, adaptant-se fins i tot al soroll o a petits detalls que no generalitzen bé. Això el fa propens a *overfitting*, ja que pot obtenir una precisió molt alta en el conjunt d'entrenament (i potser també en un conjunt de test ja que les imatges s'han extret del mateix lloc). Ens donaríem compte d'aquest problema si agaféssim una imatge qualsevol d'internet i intentéssim predir-la. Tot i que el nostre model es comportaria millor que una màquina aleatòria, els resultats ni molt menys s'acostarien al 95% que hem aconseguit ara.

2.4 Model Final

Anteriorment hem explicat com iteràvem sobre un model bàsic per així anar millorant el nostre model i incrementant la precisió d'aquest. A continuació explicarem els resultats aconseguits i discutirem el model. Per començar, ja havíem dit anteriorment que aquest model funcionava amb la variant de KNN: *fine KNN*, amb tots els descriptors mencionats anteriorment i agafant els 70 descriptors més rellevants segons l'algorisme MRMR. En aquest model provem la col·lecció de dades de *TEST* i aconseguim els següents resultats, matriu de confusió (Figura 1) i les curves ROC (Figura 10) respectivament.

		Test Confusion Matrix for Model 4									
		barrafets	bob-esponja	gat+gos	gumball	hora-de-aventuras	oliver+y+benji	padre-de-familia	pokemon	southpark	tom+y+jerry
True Class	barrafets	35			2						
	bob-esponja		36								3
True Class	gat+gos			40							1
	gumball			1	44	2		1	1		1
True Class	hora-de-aventuras		2		1	41	1	1		3	
	oliver+y+benji					83					
True Class	padre-de-familia						34			2	
	pokemon							36			
True Class	southpark								19		
	tom+y+jerry	1	1							74	
		barrafets	bob-esponja	gat+gos	gumball	hora-de-aventuras	oliver+y+benji	padre-de-familia	pokemon	southpark	tom+y+jerry

Figura 9: Matriu de confusió del model final

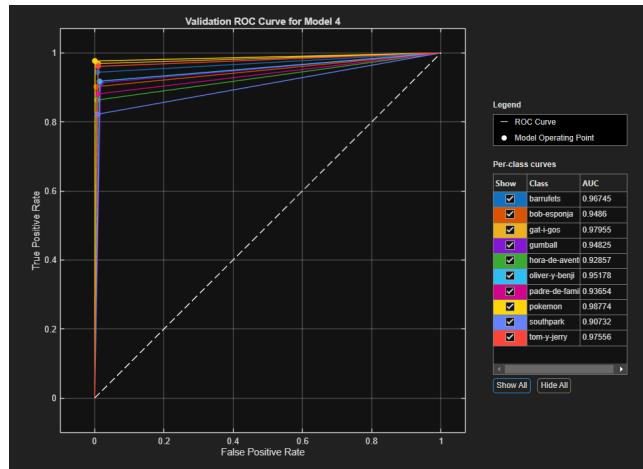


Figura 10: Curves ROC del model final

Tant un gràfic com l'altre, ens mostren una precisió molt elevada. Per una banda podem veure com a la matriu de com la diagonal conté la immensa majoria d'entrades. I per altra banda, el gràfic de les corbes ROC obtingudes presenta una forma peculiar: línies molt verticals i trams abruptes. Això és degut a dues raons: KNN amb $N = 1$, que com assigna prediccions binàries, en lloc de probabilitats, resulta en pocs punts de llindar; i per altra banda el dataset de TEST té molta preposició i hi ha molts punts errors, resultant en corbes ROC amb poca informació.

Tot i això, aquests gràfics ens ensenyen que la classe més crítica de totes és **southpark**, ja que la matriu de confusió ens mostra que la columna d'aquesta classe és la que presenta més errors. Això vol dir que de tant en tant marca una entrada com pertanyent de Southpark quan no ho és. I també, a les corbes ROC, la corba que apareix més baixa és la d'aquesta classe. Per resoldre això teníem dues solucions, o triar un descriptor que tingui molt pes per Southpark, o bé intentar detectar personatges d'aquella sèrie, per així millorar la robustesa. Una altra classe conflictiva és **hora-de-aventuras**, que s'equivoca a predir-la molt de tant en tant.

2.5 Intent de detecció d'outliers

En un intent de millorar la robustesa del sistema de classificació, vam explorar la possibilitat de detectar *outliers* mitjançant un llindar basat en la distància mínima del classificador KNN. La idea era que, si una imatge del conjunt de test quedava massa lluny de qualsevol exemple del conjunt d'entrenament, fos descartada com a exemple atípic no fiable. El model es va entrenar amb el següent codi, utilitzant $K = 1$:

```
1 mdl = fitcknn(T, 'Label', 'NumNeighbors', 1);
```

Listing 6: Entrenament del model KNN per intentar detecció d'outliers.

A l'hora de fer la predicció, s'obtenien etiquetes raonables (amb precisió similar al model anterior):

```
1 predictedLabels = predict(mdl, Ttest);
```

Tanmateix, per implementar la detecció d'outliers, vam provar de calcular la distància a l'exemple d'entrenament més proper, i descartar aquells exemples de test amb una distància superior a un cert llindar (per exemple, 5000u.d.). Malauradament, aquesta estratègia no va donar bons resultats, ja que:

- Els classificadors no són uniformes: hi ha classes amb dispersió molt diferent, de manera que una distància “normal” per a una classe pot ser un outlier en una altra.
- El model obtingut amb **Classification Learner** no retorna informació suficient per a aquest propòsit: només s'obté la predicció final i els **scores**, els quals no reflecteixen cap distància real, sinó una mena de confiança relativa. Havíem de fer servir els models com **fitcknn**, que eren molt més difícils de tunejar que un model de **Classification Learner**, ja que aquesta app ens permet veure fàcilment les matrius de confusió, gràfics MRMR, executar tests...

Per tot això, hem decidit descartar aquest mètode de detecció d'*outliers*, ja que no aporta una millora objectiva ni fiable al procés de classificació.

Som plenament conscients que el fet de detectar *outliers* és importantíssim quan estem classificant dades. Però és cert també que amb les limitacions del projecte ens ha sigut molt complicat poder donar aquesta extensió del programa. Un parell de formes raonables per la detecció que vam pensar hagués estat:

- **Augment de mostra TRAIN:** una forma hagues estat augmentar les imatges d'entrenament perquè en el cas de fer servir KNN les distàncies homogeneïtzessin. És a dir, tenir els espais de classes més densos, per així poder fer servir un *threshold* de KNN més ajustat.
- **Afegir la classe *outlier*:** una altra forma seria afegir una novena classe a l'entrenament. Aquesta classe hauria de tenir imatge de tot tipus i mitjançant l'experimentació decidiríem la quantitat d'imatges d'aquesta classe. Un bon punt de partida seria que tingués un nombre similar a les imatges de cada classe. Però com hem vist també hauríem d'augmentar el conjunt d'imatges d'entrenament ja que les classes no son prou denses en l'espai de solucions.

2.6 Funcionament de la aplicació

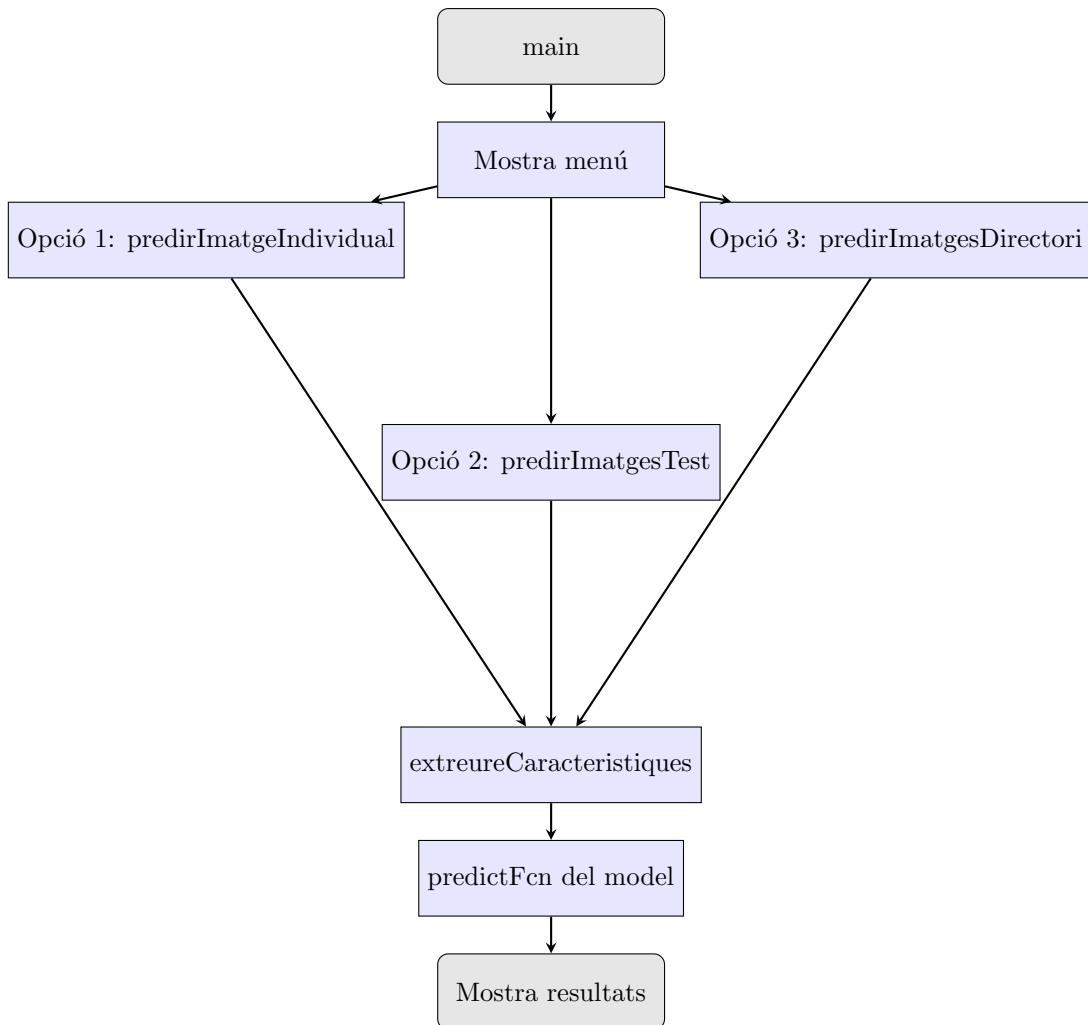
La nostra aplicació, per l'usuari funciona tota directament des de l'arxiu `main.m`. Per executar la classificació d'imatges, cal o bé amb el botó de *Run* de matlab (o prement F5), o bé executar la funció `main` des de la consola de MATLAB:

```
1 >> main
```

Aquesta funció carrega el model preentrenat `seriesClassificationModel.mat` i presenta a l'usuari un menú amb tres opcions:

- **Opció 1:** Classificar una imatge individual seleccionada manualment.
- **Opció 2:** Classificar totes les imatges del directori `./TEST`, calculant el nombre d'encerts, errors i la precisió.
- **Opció 3:** Classificar totes les imatges (amb extensions `.jpg`, `.jpeg` i `.png`) d'un directori seleccionat per l'usuari, mostrant la predicció una per una.
- **Opció 4:** Obté les estadístiques de totes les imatges del directori seleccionat. És a dir, com la opció 3, però en comptes de mostrar les imatges una per una et retorna el nombre de imatges que s'ha predit per cada classe.

Un cop seleccionat el mode d'execució, el programa carregarà la imatge o les imatges, extraurà les característiques i es farà una crida al classificador ja entrenat per predir la o les imatges. En el cas de test, no es mostraran les imatges individuals amb la seva etiqueta predicta, sinó que es farà un estudi de la mitjana de totes les imatges, retornant el nombre d'encerts, el nombre de fallides i la precisió total. Per la tercera opció si que es mostrarà imatge per imatge i per cada una es dirà el resultat que ha donat el predictor. Vegeu a continuació un diagrama del flux del programa (és una simplificació).



El fitxer `classifier.m` com ja s'ha esmentat anteriorment, és el responsable de generar les taules d'entrenament i de test que es requereixen per entrenar i testejar el model. També, el fitxer responsable d'extreure les característiques és: `extreureCaracteristiques.m`. Aquest fitxer es farà servir ja sigui per entrenar el model, o per predir. Per acabar, el fitxer `seriesClassification.mat` és model classificador, que conté la informació per predir.

2.7 Conclusions

En conjunt, el nostre projecte ha assolit els objectius plantejats amb un rendiment satisfactori. Hem aconseguit desenvolupar un classificador de sèries amb una precisió elevada (fins al 95.1%) utilitzant descriptors de color, textura, vores i forma, i aplicant diferents estratègies de selecció de característiques. A més, hem construït una aplicació funcional i fàcil d'utilitzar per a la classificació d'imatges noves.

Tot i aquests resultats positius, cal reconèixer que hi ha aspectes millorables. Un dels principals punts febles ha estat la gestió dels *outliers*. Malgrat haver explorat la estratègia de llindar de distància en KNN, no va oferir una solució prou robusta. En aquest sentit, es podria haver invertit més temps en buscar mètodes alternatius, com ara models de detecció d'anomalies més sofisticats (per exemple *One-Class SVM*) o augmentar significativament la varietat de dades d'entrenament per densificar els espais de representació.

D'altra banda, tot i haver aconseguit bones precisions amb el model fine KNN, aquest és particularment susceptible a overfitting, tal com hem comentat. Per tant, tot i que els resultats amb les imatges de test han estat bons, caldria verificar la generalització amb imatges totalment externes i no provinents del mateix entorn. En aquest context, potser seria convenient explorar models més

generalitzables com xarxes neuronals.

Finalment, el procés iteratiu que hem seguit ha estat molt encertat: hem anat afegint descriptors progressivament, analitzant la seva rellevància i refinant el model de forma justificada. Això ha permès construir un pipeline sòlid, comprensible i replicable. Tot i les limitacions, considerem que el treball realitzat és bo i demostra coneixement sobre les tècniques aplicades i una capacitat crítica per avaluar-les i millorar-les.

3 Detecció de personatges

Notem que, a la carpeta respectiva sobre la part del projecte de detecció de personatges, s'hi troba un README que en detalla l'estructura de directoris d'aquesta part del projecte, i les diferents components que s'utilitzen. També notem, que el personatge que detectem és el Bob Esponja, de la mateixa sèrie Bob Esponja, valgui la redundància.

3.1 Algorisme

Hem plantejat la detecció del Bob Esponja de la següent forma: En primer lloc, obtindrem un model que, donat un patch d'una imatge (una window de 128×128 píxels), ens detecti amb certesa si el patch pertany o no pertany a un Bob Esponja. Cal mencionar que li direm patch a una regió rectangular de la imatge. Per donar-ne un exemple, a continuació mostrem patches que pertanyen al Bob Esponja, i patches que no pertanyen al Bob Esponja (aquestes imatges són part de les que hem utilitzat per entrenar el model):

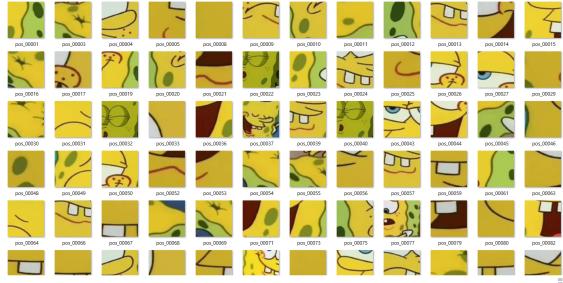


Figura 11: Patches positius



Figura 12: Patches negatius

Un cop que hem obtingut un model que és capaç de distingir si un patch (ens repetim, una window de 128×128) pertany al Bob Esponja (positiu) o no (negatiu), ho utilitzarem per detectar si a la imatge hi és o no el Bob Esponja globalment, per exemple a la imatge següent. Resumidament, ho fem de la següent forma:



Figura 13: Imatge positiva de mostra

El programa itera sobre % dels patches de la imatge, o sigui els windows de mida 128×128 de l'imatge (comentem que el % es pot configurar, i que uns % elevats fan que l'execució sigui molt lenta). Llavors, per cada patch, (extraiem les característiques) i utilitzant el model anterior, prediem si el patch pertany al Bob Esponja o no.

Finalment, en cas de detectar que a la imatge hi ha un nombre mínim (*threshold*) de patches del Bob Esponja, diem a l'usuari que sí apareix el Bob Esponja, i no en cas contrari. Notem que aquest *threshold* és configurable, i que és preferible que sigui major que 1 per tal d'estalviar-nos falsos positius. Per exemple, amb la imatge de mostra, i comprovant un 15% de les windows de 128×128 i requerint com a mínim 2 patches de Bob Esponja, el resultat del programa és:

```

1 Sample rate used to select 128x128 windows: 0.15
2 Minimum number of windows with spongebob occurrence needed: 2
3 Result: SpongeBob found! Number of spongebob windows found: >3

```

3.2 Obtenció de més imatges

Un dels problemes que sorgeix, és que la base de dades que es disposa juntament amb l'enunciat, és molt reduïda per tan sols fer un test. Per exemple, inicialment només disposem de menys de 5 imatges (considerablement diferents) on no aparegui el Bob Esponja, i a més de la resta de 35 imatges sempre hi apareix el Bob Esponja. A més, també ens interessa disposar de més imatges, per poder obtenir grans quantitats de patches que pertanyen i no pertanyen al Bob Esponja (TRAIN), que és necessari per tal d'entrenar un model amb classificadors que ho predigui amb certesa.

D'aquesta forma, és clar que necessitem augmentar la base de dades. Per aconseguir això, hem descarregat 6 vídeos (fragments de capítols) de Bob Esponja trobats a *Youtube* (a la següent llista pública: <https://www.youtube.com/playlist?list=PLaMHyq8hhBW1pu0zcGpRtNikKGigiFTAa>), i descarregats a la següent pàgina web pública <https://www.socialplug.io/es/free-tools/descargar-video-youtube>.

Finalment, un cop descarregats els 6 vídeos (de no més de 4 minuts), utilitzem un script de *Python* per tal d'extreure aleatoriament N fotogrames dels vídeos descarregats. (Comentem que el script utilitza les següents llibreries públiques de *Python*: argparse; cv2). Utilitzarem les imatges dels 3 primers vídeos per augmentar les imatges de la base de dades per obtenir els patches pel TRAIN, i les imatges dels altres 3 vídeos per les imatges de TEST. Comentem que les noves imatges i script de *Python* es troben a la carpeta `spongebob_dataset/extended_spongebob_dataset`. A continuació mostrem part de les noves imatges pel TRAIN i pel TEST respectivament:



Figura 14: Mostra de noves imatges per obtenir patches pel TRAIN



Figura 15: Mostra de noves imatges pel TEST

3.3 Obtenció del TRAIN

Tal com hem comentat anteriorment, necessitem un model que ens predigui si un patch (window de 128×128) pertany a un Bob Esponja o no. D'aquesta forma, necessitem una base de dades TRAIN bastant gran, i que tingui les imatges classificades segons si són patches o no de Bob Esponja. Per aconseguir això, considerem les imatges de la base de dades original més les noves que hem afegit (explicat a l'apartat anterior), i etiquetem cada imatge manualment mitjançant el *imageLabeler* que ofereix *Matlab*. A continuació mostrem una imatge per exemplificar el procés que hem fet per cada imatge:



Figura 16: Mostra del procés d'etiquetatge manual

Comentem, que no etiquetem tot el Bob Esponja com a tal, sinó que només la part més representativa (el quadrat central groc per dir-ho d'alguna forma). Això ho fem ja que els patches que pertanyen a aquesta regió central creiem que són els més senzills de que un classificador predigui. També comentem que el projecte d'etiquetatge es troba al projecte, a la carpeta `scripts_to_get_TRAIN`.

Un cop tenim cada imatge etiquetada, exportem el projecte i obtenim una matriu d'ocurrències (és a la mateixa carpeta) del Bob Esponja. Llavors, gràcies a un script de *Matlab* (és a la mateixa carpeta), es genera una carpeta 'positives' amb N patches del Bob Esponja i 'negatives' amb N patches que no. Aquest script utilitza funcions del *Computer Vision Toolbox* relacionades amb l'*imageLabeler* per poder extreure les dades.

Finalment, hem obtingut les dades necessàries TRAIN per l'entrenament del model desitjat. Comentem que a TRAIN disposem de 5000 patches positius i 5000 negatius.

Un petit comentari:

Per últim, eliminem un petit subconjunt de patches de 'positives', que es corresponen amb els ulls del Bob Esponja, com aquesta:



Figura 17: Ulls del Bob Esponja

Eliminem aquest tipus de patches positius de TRAIN. Això és donat que aquests són irrelevants (ja que en general per no dir sempre, si apareixen els ulls del Bob Esponja, apareix la seva pell groga i porosa) i no faran altra cosa que complicar el classificador. Llavors, hem decidit eliminar-les de TRAIN, mitjançant un script en *Matlab* (és a la mateixa carpeta), que elimina els patches de Bob Esponja de la carpeta 'positives' que, simplificant, tinguin en conjunt més blanc i blau que groc. D'aquesta forma, s'eliminen les imatges dels 'ulls' del Bob Esponja. Els detalls en concret són documentats al mateix script.

Després de fer aquest procés, ens resulten 3591 patches positius de Bob Esponja, i per tal que el classificador no estigui esbiaixat, només utilitzarem 3591 patches també negatius. Exemples de les imatges a 'positives' i 'negatives' són les Figures 11 i 12, respectivament.

3.4 Descriptors utilitzats

A continuació detallarem els descriptors utilitzats i justificació dels escollits, per tal d'obtenir un model que predigui si un patch és de Bob Esponja o no ho és:

3.4.1 Descriptor d'histogrames normalitzats de HSV

De forma similar a la detecció de sèries, es genera un histograma per a cada canal H , S i V , dividint els valors dels píxels en `binCount` intervals equidistants. Però comentem, que cada histograma és normalitzat per tal que la suma de tots els contenidors sigui 1. En el nostre programa, utilitzem 32 `binCounts`. Comentem que utilitzem un histograma de colors ja que és invariant a canvis de geometria però a il·luminació, i en el cas del Bob Esponja la il·luminació no canvia molt dràsticament. També comentem que no utilitzem un histograma d'orientacions de gradient (HOG), ja que podria interessar per detectar la forma del Bob Esponja, però aquesta pot canviar dràsticament.

3.4.2 Descriptor de densitat de vores

De la mateixa forma que en la detecció de sèries, primer convertim la imatge a escala de grisos, i després s'aplica l'algorisme de detecció de contorns de Canny, utilitzat anteriorment (a la primera part del projecte). Ens repetim, que la densitat de contorns es defineix com la fracció de píxels que resulten detectats com a "contorn" respecte el total de píxels del patch:

$$\text{edgeDensity} = \frac{\text{nombre de píxels en edge}}{\text{nombre total de píxels}}.$$

Aquesta descriptor ens dóna informació sobre la textura, i pensem que pot ser útil alhora de tenir en compte els porus que té la pell dels patches del Bob Esponja.

3.4.3 Descriptor LBP

Aquest descriptor ens dóna informació sobre la textura i patrons de la imatge. També creiem que pot ser d'interès pels porus del Bob Esponja. Es calcula el descriptor LBP (*Local Binary Pattern*) per a la imatge en escala de grisos, i es retorna un histograma amb 59 contenidors que es corresponen amb *uniform patterns*.

Més en detall, per a cada píxel (centre) es comparen els seus $P = 8$ veïns ordenats circularment: si el veí és \geq al del centre s'assigna un 1, sinó un 0. Llavors això genera un codi binari de 8 bits per a cada píxel, que es normalitza tenint en compte només els *uniform patterns* (no entrem en detall, però bàsicament només es consideren els patrons en què el nombre de transicions $0 \rightarrow 1$ o $1 \rightarrow 0$ és com a màxim 2). Finalment, el resultat és un histograma de 59 contenidors on cada contingidor h_j compta quantes vegades apareix el patró j -èssim: $LBP = [h_1, h_2, \dots, h_{59}]$.

En resum, el descriptor és un histograma que resumeix la freqüència dels patrons de textura per cada píxel.

3.4.4 Anàlisi de significança

Per cada descriptor, en mesuram mesura per la seva dependència amb la variable objectiu (classe) mitjançant l'estadístic χ^2 . No entrem en detall del funcionament exacte d'això, però bàsicament només cal saber que com més alta sigui la puntuació obtinguda, més informatiu és el descriptor (és més discriminant i permet predir més fàcilment). Gràcies al *Feature Selection* del *Classification Learner* de *Matlab*, ens resulta el següent rànquing:

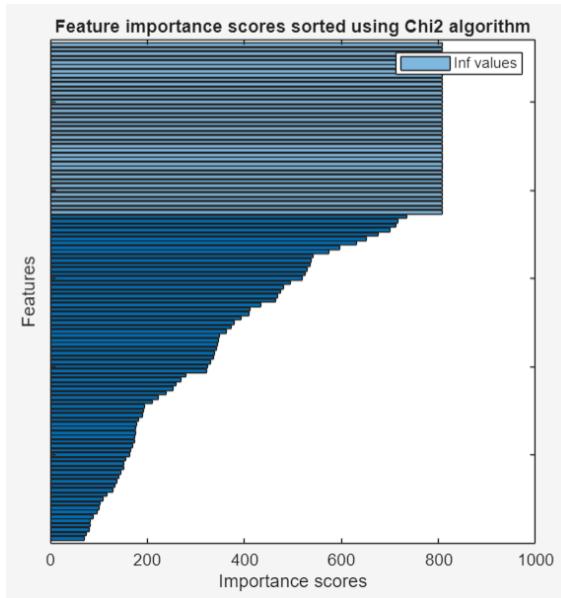


Figura 18: Anàlisi de significança utilitzant Chi2

Es pot pensar que les últimes característiques tenen molt poca importància i es podrien eliminar, ara bé, si entrem més en detall, resulta que aquestes característiques de diferents contingidors de diferents histogrames. Per exemple, els contingidors de color blau de l'histograma de Hue, alguns de Saturació, altres de Value, i també certs patrons del LBP. (Comentem que la densitat de contorn té bastanta importància; es troba la sèptima posició).

Llavors, com són contingidors concrets d'histogrames diferents, optem per no eliminar contingidors en particular. De totes maneres, podem veure això empíricament: provem d'utilitzar un *Fine Tree*; en primer lloc utilitzant tots els descriptors; en segon lloc només utilitzant els 70 primers descriptors en funció de Chi2; i finalment només utilitzant els 40 primers descriptors (els de significança

infinita). Els resultats són els següents:

1 Tree	Accuracy (Validation): 98.1%
Last change: Fine Tree	113/113 features
2 Tree	Accuracy (Validation): 98.1%
Last change: Fine Tree	70/113 features
3 Tree	Accuracy (Validation): 98.1%
Last change: Fine Tree	40/113 features

Figura 19: Accuracy utilitzant un diferent número de descriptors.

D'aquesta manera, veiem que utilitzar tots els descriptors (utilitzar els histogrames sencers) no perjudica la precisió (*accuracy*), optem per utilitzar-los tots, donat que en casos reals i més complexos o enganyosos (TEST), la resta de contenidors poden ser d'ajuda.

3.4.5 Justificació de l'elecció

Intuïtivament és bastant clar quines característiques permeten diferenciar si una imatge és un patch del Bob Esponja o no. A continuació en detallarem el motiu darrere de l'elecció de cada descriptor:

- **Histogrames per H,S,V:** És clar que en general, un patch de Bob Esponja és en gran mesura groc i amb saturació elevada. Però també, conté principalment altres colors, com un cert 'groc-marró', que es correspon amb el color dels porus de la seva pell; o el blanc que es correspon a les dents o ulls; o el negre dels contorns.
- **Densitat de contorns:** És clar que en general als patches de Bob Esponja hi ha uns quants porus. Llavors amb la densitat de contorns esperem detectar principalment aquests porus (que és clar que en la imatge de contorns els porus apareixen, com es mostra a la figura).
- **Local Binary Pattern:** Similar a l'argument que fem pel cas del descriptor de densitat de contorns, l'ideia és utilitzar el LBP (en conjunt de la densitat de contorns) per detectar informació relacionada amb els porus.

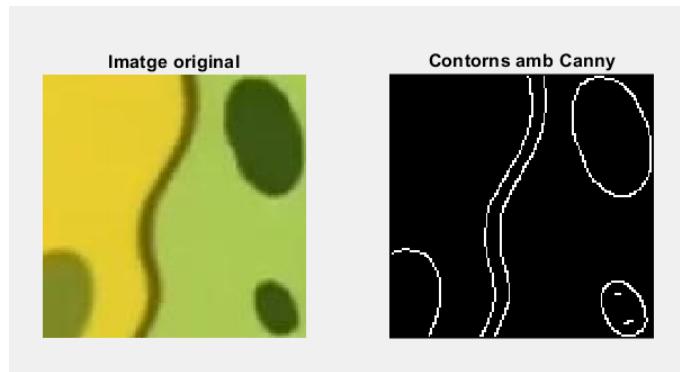


Figura 20: Contorns de Canny en els porus del Bob Esponja

3.5 Classificador utilitzat

Mitjançant el *Classification Learner* que ens ofereix *Matlab* (al igual que hem fet servir a la primera part del projecte), i per tal de decidir quin classificador utilitzar, entrenem tots els classificadors (excepte pels d'aprenentatge profund), i obtenim les següents precisions:

	2.1 Tree	Accuracy (Validation): 98.0%
	Last change: Fine Tree	113/113 features
	2.2 Tree	Accuracy (Validation): 97.9%
	Last change: Medium Tree	113/113 features
	2.3 Tree	Accuracy (Validation): 95.1%
	Last change: Coarse Tree	113/113 features
	2.4 KNN	Accuracy (Validation): 97.9%
	Last change: Fine KNN	113/113 features
	2.5 KNN	Accuracy (Validation): 97.9%
	Last change: Medium KNN	113/113 features
	2.6 KNN	Accuracy (Validation): 97.1%
	Last change: Coarse KNN	113/113 features
	2.7 KNN	Accuracy (Validation): 97.9%
	Last change: Cosine KNN	113/113 features
	2.8 KNN	Accuracy (Validation): 96.2%
	Last change: Cubic KNN	113/113 features
	2.9 KNN	Accuracy (Validation): 98.0%
	Last change: Weighted KNN	113/113 features
	2.10 Efficient Logi...	Accuracy (Validation): 68.2%
	Last change: Efficient Logistic Regression	113/113 features
	2.11 Efficient Line...	Accuracy (Validation): 74.7%
	Last change: Efficient Linear SVM	113/113 features

Figura 21: Precisió en el TRAIN dels diferents classificadors.

Observant això, estem entre el *Fine Tree*, i el *Weighted KNN*. Ara bé, tal i com s'ha comentat en la documentació de la detecció de sèries, els mètodes KNN són sensibles a les dades d'entrenament, i d'aquesta manera a l'*overfitting*. Ja s'ha estudiat això prèviament. També per afegir, els mètodes *Tree* són molt ràpids (són arbres de decisió, i la velocitat només depèn de la profunditat de l'arbre). Per altra banda, els KNN són més costosos temporalment, i encara ho són més si la base de dades del TRAIN és gran (recordem que en el nostre cas tenim més de 3000 imatges).

Llavors, segons aquests criteris, ens hem decidit per utilitzar el model de *Fine Tree*. A continuació mostrem la seva matriu de confusió i la seva corba ROC:

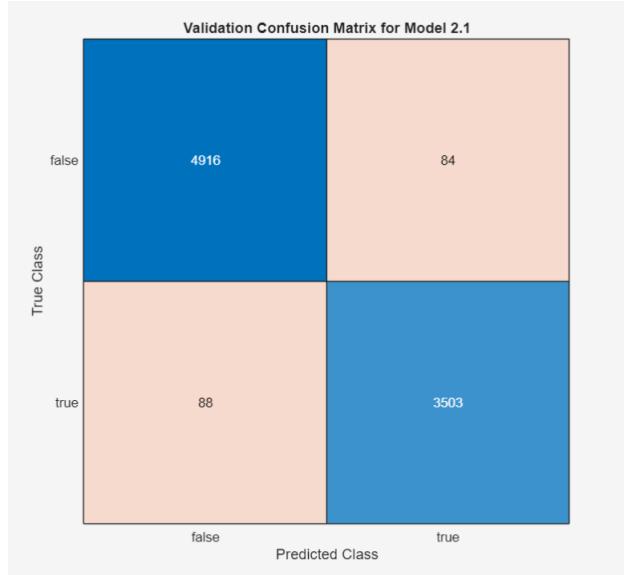


Figura 22: Matriu confusió en el TRAIN utilitzant un Fine Tree.

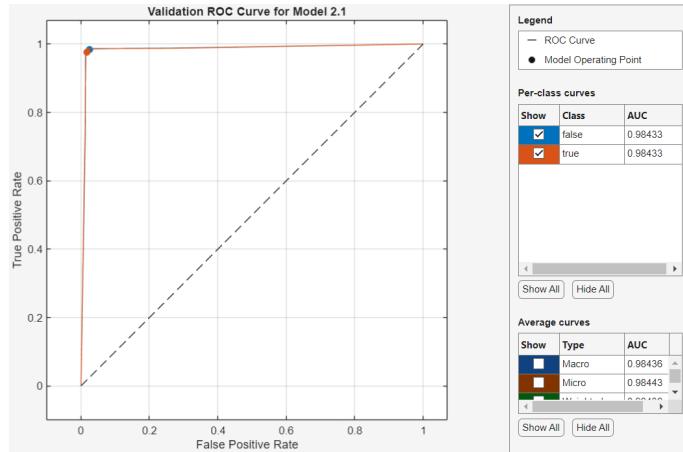


Figura 23: Corba ROC en el TRAIN utilitzant un Fine Tree.

3.6 Resultats obtinguts

Primer comentem que, com hem vist a la secció sobre el funcionament del nostre programa, hi ha dos paràmetres que podem configurar:

- **sample rate:** Una probabilitat entre 0 i 1, que indica quin nombre de patches (windows de 128×128) comprovarem i predirem si són o no patches del Bob Esponja.
- **threshold:** Un enter positiu major que zero, que indica el mínim número de patches del Bob Esponja presents a la imatge, per tal de concloure que a la imatge efectivament hi és o no hi és el Bob Esponja.

Preferiblement, el *sample rate* ha de ser baix (menor que 0.5), ja que en cas contrari, es comproven masses windows d'una mateixa imatge, i el cost temporal és bastant elevat. Per altra banda, els valors concrets del *sample rate* i *threshold* no tenim altra opció que no sigui experimentalment, provant diferents combinacions i avaluant-les sobre les imatges de TEST.

Finalment, hem conclòs, a partir de l'experimentació, que una bona parella de valors és *sample_rate* = 0.15, i *threshold* = 2. Que encara que segurament no són els òptims, són els que ens ha ofert els millors resultats. A continuació es mostra la matriu de confusió i precisió sobre les imatges de

TEST:

Evaluation Results:		
	Pred_Pos	Pred_Neg
True Positives (TP)	15	2
False Negatives (FN)	5	12
False Positives (FP)	5	12
True Negatives (TN)	12	15
Total images	: 34	
Accuracy	: 79.41%	

Actual_Pos	15	2
Actual_Neg	5	12

Figura 24: Resultats finals (matriu confusió, precisió) en TEST; $sample_rate=0.15$; $threshold=2$.

De totes maneres, volem comentar que aquesta solució no és perfecta. Per exemple, en general es poden donar bastants falsos positius, sobre tot si el fons o elements de l'entorn són grocs i amb textura rugosa.

També, molts cops el classificador no prediu gens bé si un patch és del Bob Esponja o no. De totes formes, intentem solucionar (i en general ho aconseguim) gràcies al $threshold$. Per exemple, a continuació mostrem uns exemples de falsos positius (els falsos negatius són poc comuns):

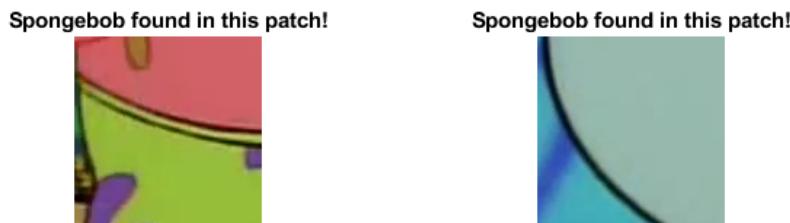


Figura 25: Exemples de patches que són falsos positius.

Igualment, volem comentar que una cosa que considerem molt positiva de la nostra solució, és que a les imatges on sí que hi ha un Bob Esponja, com sabem a quin patch apareix, podem saber la ubicació aproximada d'on a es troba el Bob Esponja. A continuació mostrem unes mostres de casos correctes d'identificació dels patches de Bob Esponja:

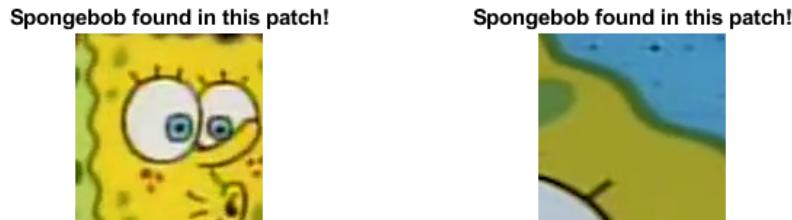


Figura 26: Exemples de patches positius identificats correctament.

Per últim volem remarcar que, els resultats (matriu de confusió i precisió del TEST) no són deterministes. De manera que en diferents execucions es poden obtenir resultats diferents. Això és degut a que el *sample* de windows (donat pel sample_rate) és aleatori.

I com executo aquesta part del projecte?

Tota la informació necessària per executar aquesta part del projecte i entendre l'estructura de directoris és explicada al README a la carpeta `./DeteccioPersonatges`. De totes maneres, comentem breument com fer-ho: Només cal executar el `main.m` (des de Matlab). Un cop s'executa, l'usuari ha d'introduir l'opció 1 o 2:

- Amb l'opció (1), l'usuari podrà seleccionar una imatge al seu sistema de carpetes, i el programa imprimira si creu que hi ha el personatge Bob Esponja o no.
- Amb l'opció (2), l'usuari podrà seleccionar una carpeta de TEST (que contingui subcarpetes `positives` i `negatives` amb aquests noms). El programa llavors, predirà cada imatge, i finalment es mostra la matriu de confusió donada per les imatges de TEST.