Facultat d'informàtica de Barcelona

Visió per Computador

Pràctica 2

Reconeixement automàtic de dígits manuscrits

Jordi Izquierdo Herrera

Joan Francesc Muntaner González

Juny, 2020

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índex

1 Introducció

L'objectiu de la pràctica és reconèixer els dígits (0-9) presents en una sèrie d'imatges. Desenvoluparem un programa en Matlab que sigui capaç de classificar els dígits correctament.

Per a aconseguir-ho se'ns proporciona el dataset MNIST, obtingut a partir de estudiants de secundària i treballadors de correus dels estats units. Entrenarem i provarem el nostre sistema de reconeixement amb MNIST.

Intentarem explorar diverses metodologies per aconseguir el nostre objectiu però tinguent en ment que els mètodes actuals usen xarxes neuronals arribant a 0.17% d'error rate i que el millor resultat sense xarxes neuronals està al voltant del 0.56% d'error rate.

2 Pre-processament

Abans d'extreure els descriptors de l'imatge pot ser bo tractar la imatge. En el nostre cas, depèn del descriptor, la deixem tal qual, la binaritzem o li apliquem certes operacions morfològiques (com un close).

Revisant la bibliografia, es pot observar com la majoria d'articles al respecte solen optar per tres opcions: cap pre-processament, binarització i deskewing (freqüentment combinant les últimes dues).

3 Descriptors

Tot i què la nostra versió inicial (basada en HOG i SVM) ja va assolir molts bons resultats (99.02%), vam decidir provar diversos mètodes per veure com es comparaven.

3.1 Versió 1 - Histogrames de gradients orientats (HOG)

Són els descriptors que ens han donat millor resultat per si sols. Els descriptors HOG ens donen molta informació sobre els dígits ja que en certa manera ens donen la inclinació per cada bloc del descriptor. Intuïtivament, ens ajuda a detectar les formes dels dígits segons les inclinacions que presenten (per exemple el 1 presentarà més inclinació vertical que altres, etc).

Aquest descriptor proporciona moltes opcions de tunning, com poden ser agafar signe o no (rang 0-180 o 0-360), el tamany de cada cel·la o de cada bloc, el número de bins (nº d'orientacions que tenim en compte), i l'overlap entre blocs. Nosaltres hem optat per usar signed orientation (0-360) (que ens diferencia el pas entre negre-blanc i blanc-negre), CellSize 5x5, BlockSize 2x2, Overlapping

de mig block, 8 bins. Cosa que ens dóna un vector de **288 característiques per imatge**.

3.2 Versió 2 - Píxels i forats

A aquesta versió hem usat diferents descriptors que ens han acabat donant un vector de 82 característiques.

Suma per columna i fila

Així, tenim 40 característiques que consisteixen en la suma per cada fila i per cada columna (ja que la imatge és de 20x20) de la imatge. Per a calcular aquest descriptor no hem binaritzat la imatge ja que ens dóna més informació no fer-ho.

Distància als marges

A més a més, hem afegit el descriptor de distància als marges. Aquí, ja amb la imatge binaritzada, simplement comptem quants de píxels negres hi ha fins que trobem el primer píxel blanc. Tot i que tenim 4 marges (superior, inferior, esquerre i dret), hem descartat la distància als marges superior i inferior després de provar-les i veure que aportaven molt poca informació addicional (no milloraven la precisió). Així, només tenim en compte els marges esquerre i ret, que comprenen 20 característiques més cadascun.

Forats

Hem afegit dos descriptors que ens donen informació respecte a els forats que pot haver-hi a les imategs. El primer de tots, el nombre de forats, obtingut restant al nombre d'Euler el nombre de components connexes de la imatge binaritzada a la que hem aplicat un close per tancar forats petits que no ens siguin rellevants. Finalment, hem afegit l'àrea total dels forats de la imatge, restant l'àrea de la imatge a l'àrea plena (FilledArea), ambdues propietats extretes usant regionprops.

3.3 Versió 3 - Transformada d'Hough

La transformada de Hough permet la detecció de formes parametritzables com poden ser rectes o cercles. Precisament intentarem utilitzar aquesta capacitat per a distingir entre els dígits. Utilitzem la funció Hough() per detectar totes les línies de la imatge. De la funció s'obté una matriu on les files representen distància des de l'origen fins a la línia al llarg d'un vector perpendicular a la línia i les columnes el angle respecte el eix de coordenades. Fem una mitjana de les columnes per obtenir els descriptors de la imatge.

3.4 Versió 4 - Fourier

Podem utilitzar la transformada de Fourier per analitzar la forma del perímetre de la imatge, utilitzant el vector resultant com a descriptor. Com que cada imatge té un nombre de descriptors diferents, utilitzem la funció resample per obtindre vectors de mateixa mida que poguem utilitzar com a descriptors.

3.5 Versió 5 - Versions 1+2

Aquesta versió simplement consisteix d'ajuntar els descriptors de les versions 1 i 2, donant-nos un vector de característiques de 370 elements.

4 Anàlisi de significància i no-redundància

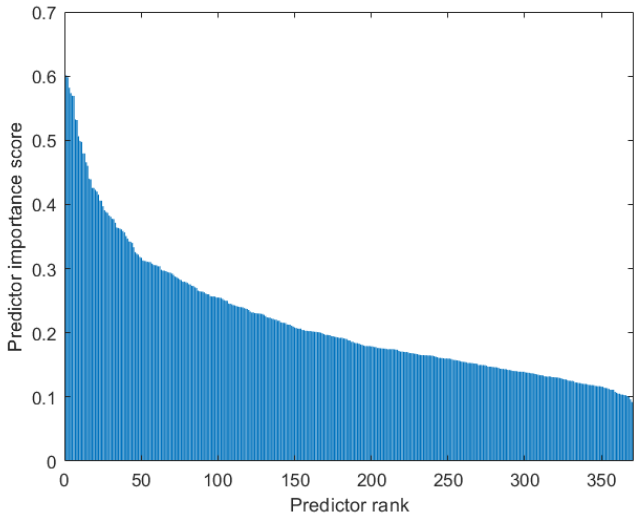
Farem un anàlisi del vector de característiques de 370 posicions més la label de cada imatge. Recordem que el vector conté el següent a cada posició:

Posició	Característica
1	Àrea dels forats
2	Número de forats
3-22	Suma per fila
23-42	Suma per columna
43-62	Distància al marge esquerre
63-82	Distància al marge dret
83-370	HoG
371	Label

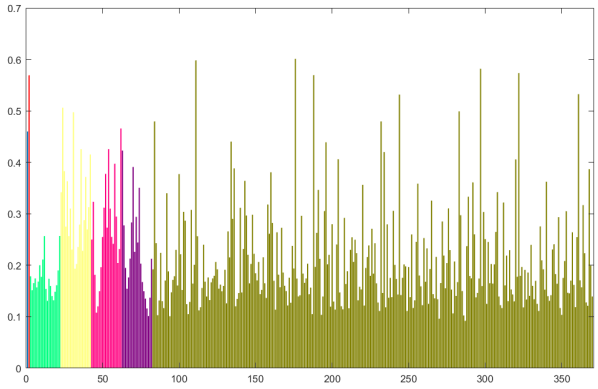
A continuació utilitzem la funció de Matlab *fscmrmr*[1] que aplica l'algoritme mRMR (minimum redundancy maximum relevance) per ordenar les característiques segons estiguin correlades amb la variable de resposta (label) i siguin redundants amb les altres. Això és ja que ens interessen les variables més independents (minimum redundancy) i més significatives (maximum relevance), que expliqui la resposta.

A continuació adjuntem la gràfica de les scores dels 370 descriptors en relació a la label ordenades descendentment.

[1] https://es.mathworks.com/help/stats/fscmrmr.html



La següent gràfica mostra les scores per a cada grup de descriptors:



Per a estudiar quines característiques són les més importants ho dividirem en els intervals:

- $>$ 0.4 Característiques molt bones
- 0.4-0.3 Característiques bones
- 0.3-0.25 Característiques significants
- 0.25-0.2 Característiques mitjanes
- 0.2-0.13 Característiques no molt rellevants
- $<$ 0.13 Característiques dolentes (últimes 49)

Característiques molt bones

Aquí hi tenim 24 característiques, ordenades: 176, 111, 297, 322, 188, 2, 361, 244, 24, 283, 31, 232, 84, 62, 1, 134, 196, 36, 54, 63, 234, 42, 204, 320. Tenim 15 features de HoG, 1 de l'àrea dels forats, 1 del número de forats, 4 de suma per columna, 2 de distància al marge esquerre i 1 de distància al marge dret.

Distribució de característiques

En general, a la següent taula podem veure-hi la distribució de les features segons la score.

Interval	>0.4	(0.3,0.4)	(0.25,0.3)	(0.25,0.2)	(0.2,0.13)	<0.13
Total	24 (6.5%)	38 (10.2%)	42 (11.4%)	63 (17.02%)	154 (41.6%)	49 (13.24%)
1	1 (100%)	-	-	-	-	-
2	1 (100%)	-	-	-	-	-
3-22	-	-	2 (10%)	2 (10%)	16 (80%)	-
23-42	4 (20%)	6 (30%)	5 (25%)	4 (20%)	1 (5%)	-
43-62	2 (10%)	5 (25%)	4 (20%)	4 (20%)	3 (15%)	2 (10%)
63-82	1 (5%)	2 (10%)	3 (15%)	5 (25%)	7 (35%)	2 (10%)
83-370	15 (5.2%)	25 (8.7%)	28 (9.7%)	48 (16.7%)	127 (44.1%)	45 (15.6%)

Podem observar-hi, que hi ha molts descriptors de HoG que no aporten molta informació, això posiblement sigui degut a que estigui agafant gradient de cel·les buides de la imatge.

A més, veiem que hi ha característiques dolentes de distància als marges laterals, això segurament s'explica per quan no troba informació (i per tant val 20).

Finalment, les diferències entre sumes per columna i fila, ens indica que hi ha informació redundant i per això una té molt millor score que l'altra, ja que recordem que aquest algorisme penalitza la redundància.

Anàlisi Pearsson

El coeficient de correlació de Pearsson mesura la dependència lineal entre dues variables i és independent de l'escala (a diferència de la covariància). Aíxi, podem utilitzar-lo per detectar dependències lineals entre els nostres descriptors i trobar redundàncies, o per detectar quins tenen una dependència quasi nul·la de la variable de resposta (labels). Tot i això, només analitza relacions lineals i no ens assegura independència o dependència ja que podria haver-hi relacions no lineals. Podem calcular la matriu de correlació de pearsson amb Matlab amb la comanda *corrcoef*[2].

Primer, buscarem els descriptors amb **poca rellevància**, és a dir, aquells que tinguin molt poca correlació amb la variable resposta. No indagarem molt per

[2]https://es.mathworks.com/help/matlab/ref/corrcoef.html

aquest camí, ja que intuïm que hi ha dependències no lineals.

Trobem que hi ha només 10 descriptors que tinguin una correlació absoluta de menys de un 1%, 2 d'ells corresponents a suma per columna, particularment, les columnes 3 i 20; això segurament s'expliqui per l'absència general de píxels als extrems de les imatges. Els altres 8 descriptors corresponen a HoG, segurament provenents de llocs on el valor del gradient no és rellevant per a l'imatge.

Ara intentarem trobar redundàncies, és a dir, **dependències lineals entre els predictors**, per tant buscarem els valors màxims de la matriu de correlació sense tenir en compte la variable de resposta. Trobem que hi ha 4 descriptors que corresponen a característiques HoG que tenen un dependència lineal de més d'un 95%.

Si fem l'anàlisi sense tenir en compte els descriptors de HoG veiem que trobem 14 descriptors amb un 85% de correlació entre ells. Alguns d'ells están correlats ja que no aporten informació gaire rellavant com pot ser l'exemple de la suma de la primera columna amb la distància de la primera fila per l'esquerra o aquesta amb la distància per la dreta, cap d'ells tindrà informació rellevant en la majoria dels casos, i per tant hi podem establir una dependència. Igualment la suma de les últimes columnes veiem que no és rellevant. Un altre cas és la distància al marge dret, que en molts casos a les últimes files serà similar (quan hi aparegui una línia horitzontal).

PCA

Després d'aquest anàlisi veiem que hi ha unes quantes dependències que es podrien eliminar i es podria reduir la dimensionalitat del problema. Fent un anàlisi de components principals (PCA) usant la funció *pca*[3], obtenim que amb un vector de 183 components (aproximandament la meitat de tamany) som capaços d'explicar un 99.9939% de la variància i a partir del 184 cada component explica menys de un 0.0001% d'aquesta. A més, amb 87 components del pca es pot explicar un 99.95% de la variància. No hem aplicat aquesta reducció de dimensionalitat ja que hem pogut provar tot el dataset sense problemes, però pot comportar una reducció important del temps d'entrenament.

5 Classificadors

El mètode a procedir ha estat provant diferents classificadors amb una mostra reduïda del set amb l'App de Classification Learner amb 5-Fold validation. En general, els que han funcionat millor han estat els SVMs polinòmics, Ensemble Subspace k-NN, i k-NN.

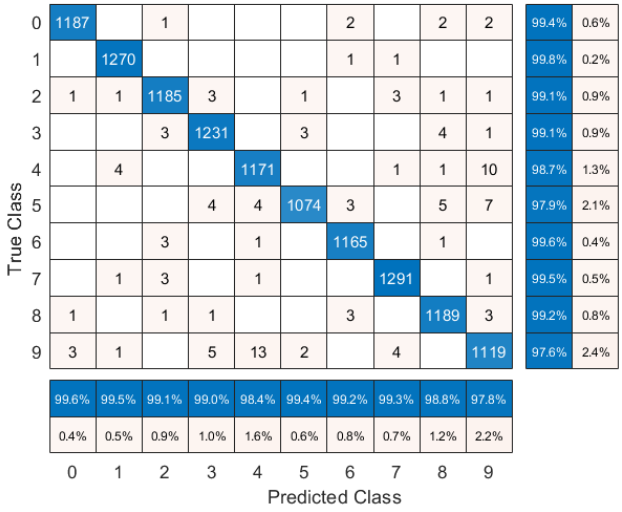
Posteriorment hem exportat els models i els hem entrenat amb les primeres

[3] https://es.mathworks.com/help/stats/pca.html

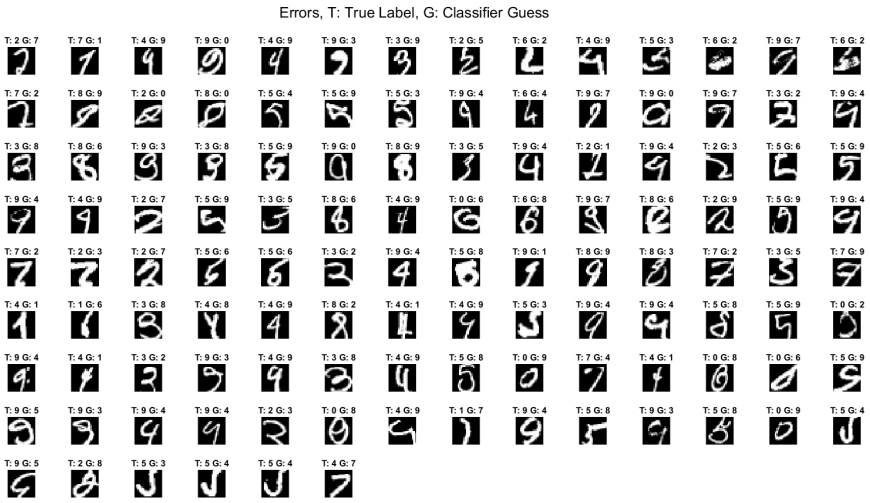
48000 imatges del dataset (80%). Seguidament, els hem testejat a les altres 12000 imatges restants. N'exposem per tant, tant la validation accuracy com la precisió del millor classificador per a cada cas, que ha resultat ser un SVM sempre.

5.1 Versió 1

Recordem que la versió 1 constava de 288 descriptors extrets a partir de HOG. El millor resultat que hem obtingut amb aquests descriptors ha estat aplicant un SVM quadràtic (ordre 2) amb una box constraint de 988.914... i codificació "onevsall" (la resta de paràmetres per defecte). Amb aquestes condicions, hem obtingut una **precisió del 99.0167%** amb una validation accuracy de 99.02% de l'entrenament. Hem obtingut la següent matriu de confusió a la qual podem apreciar que la major part de confusió prové de dificultats per diferenciar entre el 4, el 5 i el 9.

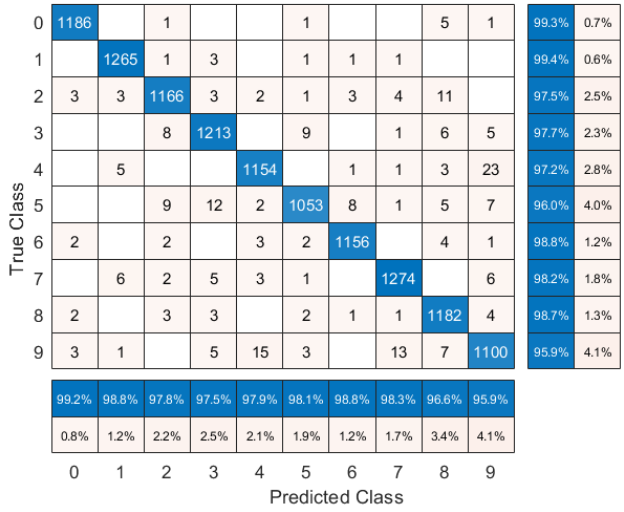


Aquesta versió sobre el *testset* de 12000 imatges només ha fallat a 118 imatges, que adjuntem a continuació. Podem observar-hi que alguns nombres directament són impossibles d'acertar fins i tot per un humà ja que són molt ambigus.



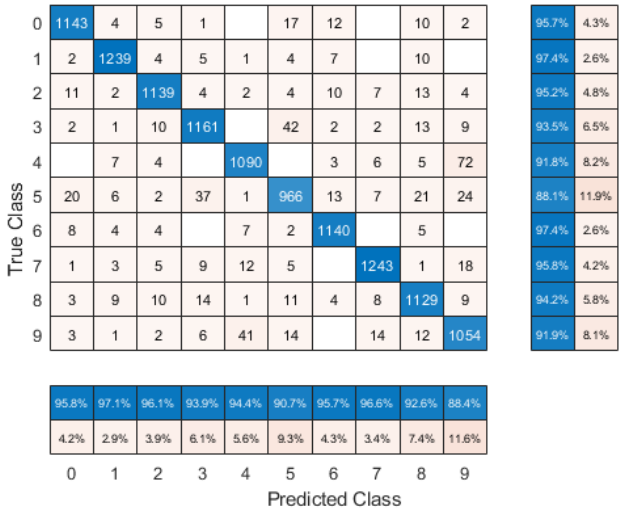
5.2 Versió 2

Recordem que la versió 2 constava de 82 descriptors extrets a partir dels píxels per columna i fila, la distància als laterals i dos descriptors de forats. El millor resultat obtingut amb aquests descriptors ha estat aplicant un SVM cúbic (ordre 3) amb la resta de paràmetres per defectes. Amb aquestes condicions, hem obtingut una **precisió del 97.9083%** amb una validation accuracy de 97.84% de l'entrenament. Hem obtingut la següent matriu de confusió a la qual podem apreciar que la major part de confusió prové de dificultats per diferenciar entre el 4, el 5 i el 9.



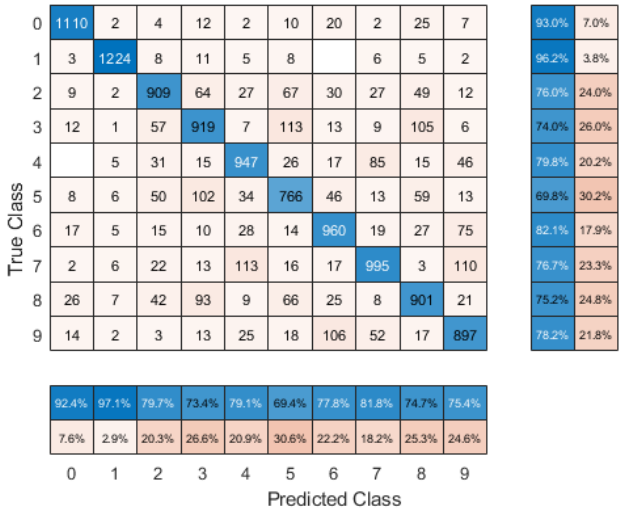
5.3 Versió 3

Utilitzant la transformada de Hough, tenim 56 descriptors descrivint la matriu de Hough. El millor resultat obtingut aplicant SVM quadràtic té una **precisió del 94.201%** amb una validation accuracy del 93.94%. Observant la matriu de confusió, podem observar que el classificador té especialment problemes amb el número 5.



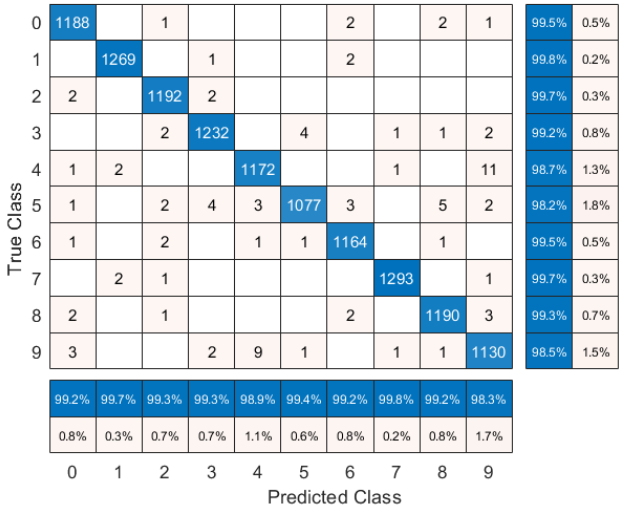
5.4 Versió 4

Utilitzant els elements de la transformada de Fourier com a descriptors, podem observar que el resultats no són prou bons. Això és degut a que amb unes imatges tant petites i simples, la transformada de Fourier dona molt pocs descriptors, en ocasions menys que deu. Amb el classificador SVM quadràtic obtenim una **precisió de 80.23%** i una validation accuracy del 78.99% veiem que aquest mètode no és molt efectiu, de fet l'únic nombre que prediu amb una certa fiabilitat és l'1 i el 2. S'ha obtingut la següent matriu de confusió:



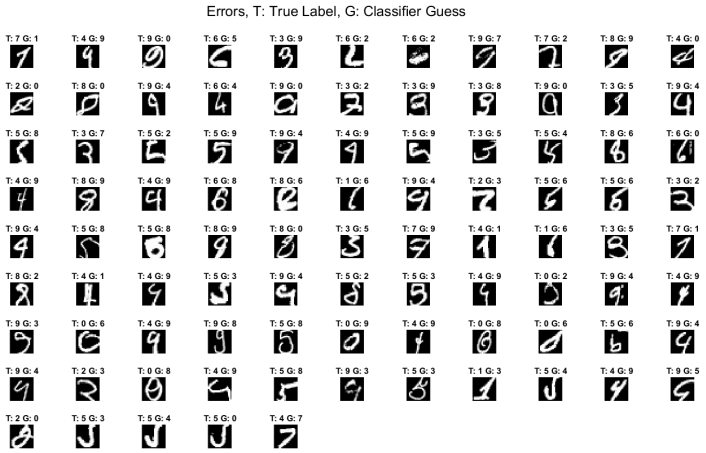
5.5 Versió 5 (1+2)

Com és lògic, al afegir més descriptors hem obtingut una petita millora, així, entrenant les 370 característiques amb un SVM polinòmic de grau 5, hem obtingut una validation accuracy de 99.14% i una **precisió de 99.225%**, lo que efectivament ens suposa el millor resultat. A sota posem la matriu de confusió corresponent.



Es pot apreciar que tot i així seguim tenint problemes amb els 4, 9 i 5.

Adjuntem les 93 imatges que fallen:



6 Comparació entre versions

La següent taula presenta una comparació dels resultats obtinguts amb les diferents versions. Podem observar-hi com la versió amb la que hem obtingut més precisió ha estat la versió 5.

Method	1 (HoG)	2 (Pixels & holes)	3 (Hough v1)	4 (Fourier)	**5 (1+2)**
Accuracy	99.0167%	97.9083%	94.201%	80.23%	**99.225%**

A continuació mostrem una taula comparativa del true positive rate de cada nombre segons el mètode, no incloem el Fourier perquè té molt mal resultat:

Digit	1 (HoG)	2 (Pixels & holes)	3 (Hough v1)	5 (1+2)
0	99.4%	99.3%	95.7%	99.5%
1	99.8%	99.4%	97.4%	99.8%
2	99.1%	97.5%	95.2%	99.7%
3	99.1%	97.7%	93.5%	99.2%
4	98.7%	97.2%	91.8%	98.7%
5	97.9%	96.0%	88.1%	98.2%
6	99.6%	98.8%	97.4%	99.5%
7	99.5%	98.2%	95.8%	99.7%
8	99.2%	98.7%	94.2%	99.3%
9	97.6%	95.9%	91.9%	98.5%

Com s'observa, la versió 5 millora el rendiment de tots els dígits excepte el 6. També es pot observar com el mètode 2 quasi arriba a la precisió dels altres en els nombres 0, 1 i 8. Finalment, és curiós com a totes les versions els pitjors casos sempre són el 5 i el 9, seguit del 4. Segons la traça del nombre, aquests 3 són fàcilment confundibles (si es tanca el 4 per dalt o el cinc per la dreta és molt similar a un 9).

Bibliografia Consultada

[1] Andrea Giuliodori, Rosa Lillo, and Daniel Peña. Handwritten digit classification. 07 2011.

[2] Subhransu Maji and Jitendra Malik. Fast and accurate digit classification. Technical Report UCB/EECS-2009-159, EECS Department, University of California, Berkeley, Nov 2009.

[3] Abdul Hafiz and Gm Bhat. Handwritten digit recognition using slope detail features. *International Journal of Computer Applications*, 93:14–19, 05 2014.

[4] Yi Lu, Steve Schlosser, and Michael Janeczko. Fourier descriptors and handwritten digit recognition. *Machine Vision and Applications*, 6, 01 1992.

[5] Munish Kumar and Deepika Ahuja. Handwritten numerals recognition using hough transformation technique. *International Journal of Advanced Research in Computer Science*, 3:118–120, 01 2012.

[6] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

A Codi Propi

A continuació adjuntem totes les funcions que hem fet servir.

A.1 Extracció de Descriptors

Histograma de Gradients Orientats (HoG)

```
function z = HOG(im)
    z = extractHOGFeatures(im,'CellSize',[5 5],'
        ↪ UseSignedOrientation',true,'NumBins',8);
end
```

Píxels per fila i columna

Implementem la suma dels valors dels píxels per a cada columna i fila de la matriu.

```
function z = pixels(im)
    a = sum(im,1);
    b = sum(im,2)';
    z = [a, b];
end
```

Distància als marges

Aquí el codi comentat seria si volguéssim obtenir també la distància als marges superiors i inferiors.

```
function z = sideEntries(im)
    l = zeros(20,1);
    r = zeros(20,1);
%     t = zeros(20,1);
%     s = zeros(20,1);
    for i=1:20
        a = find(im(i,:),1);
        b = find(im(i,:),1,'last');
%         c = find(im(:,i),1);
%         d = find(im(:,i),1,'last');
        if (isempty(a))
            l(i) = 20;
            r(i) = 20;
        else
            l(i) = a(1);
            r(i) = b(1);
        end
%         if (isempty(c))
%             t(i) = 20;
```

```
%               s(i) = 20;
%           else
%               t(i) = c(1);
%               s(i) = d(1);
%           end
    end
%     z = [l' r' t' s'];
    z = [l' r'];
end
```

Forats

Al següent codi es veu com obtenim els dos descriptors dels forats. Donada la imatge im obtenim *sh* i *numRegions-bweuler(imCL)*. Com es pot observar, fem un close de la imatge binària per eliminar petits forats negres que siguin irrelevants.

```
    imBW=imbinarize(im);
    ee = strel('disk',1);
    imCL = imclose(imBW,ee);
    [~, numRegions] = bwlabel(imCL);
    stats = regionprops(imBW,'Area','FilledArea');
    if(size(stats)>1)
        for j=1:size(stats)
            a = stats(j).FilledArea;
            b = stats(j).Area;
            sh = sh + a -b;
        end
    else
        a = stats.FilledArea;
        b = stats.Area;
        sh = a -b;
    end
    % sh representaria la superficie dels forats
    % numRegions-bweuler(imCL) representa el nombre de
        ↪ forats
    sh, (numRegions-bweuler(imCL))
```

Fourier

```
function z = fourier(im)
    %figure,imshow(im);
    % binaritzem la imatge i la invertim
    bw = im;
    %figure, imshow(bw),title('Binaritzat')
```

```
    % Close
    se = strel('disk',5);
    cl = imclose(bw, se);
    %imshow(cl),title('close');
    % obtenim contorn de la imatge
    ero=imerode(cl,strel('disk',1));
    cont=xor(ero,cl);
    %figure,imshow(cont), title('contorns')
    % obtenim coordenades del contorn
    [fila, col] = find(cl,1); % Busquem el primer p xel
    B = bwtraceboundary(cl,[fila col],'E',8,50,'clockwise'); %direccio est a l'a
    [mida, ~]=size(B);
    B = resample(B,30,mida);
    B = round(B);
    %figure,imshow(aux),title ('contorns a partir de coordenades')
    % B cont  les coordenades
    % Fourier
    mig=mean(B);
    B(:,1)=B(:,1)-mig(1);
    B(:,2)=B(:,2)-mig(2);
    % A complexes
    s= B(:,1) + 1i*B(:,2);
    % Vector dim parell
    [mida, ~]=size(B);
    if(mida/2~=round(mida/2))
        s(end+1,:)=s(end,:); %dupliquem l'ultim
        mida=mida+1;
    end
    z=fft(s);
    z=abs(z);
    %figure,plot(log(z)),title('fourier');
end
```

A.2 Funcions d'entrenament de classificadors

Obtingudes exportant models a partir de la *Classification Learner App*.

A.2.1 Versió 1

```
function [trainedClassifier, validationAccuracy] = trainClassifier2(trainingData
% [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
```

```
%  Input:
%      trainingData: a matrix with the same number of columns and data type
%       as imported into the app.
%
%  Output:
%      trainedClassifier: a struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: a function to make predictions on new
%       data.
%
%      validationAccuracy: a double containing the accuracy in percent. In
%       the app, the History list displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a matrix containing only the predictor columns used for
% training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 19-May-2020 02:32:51


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2',

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'c
predictors = inputTable(:, predictorNames);
response = inputTable.column_289;
isCategoricalPredictor = [false, false, false, false, false, false, false, false
```

```
% Train a classifier
% This code specifies all the classifier options and trains the classifier.
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 2, ...
    'KernelScale', 1, ...
    'BoxConstraint', 988.9140448531597, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    predictors, ...
    response, ...
    'Learners', template, ...
    'Coding', 'onevsall', ...
    'ClassNames', [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x) svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained model exported from Classifi
trainedClassifier.HowToPredict = sprintf('To make predictions on a new predictor
↪ yfit = c.predictFcn(X) \nreplacing ''c'' with the name of the variable that is

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2',

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'c
predictors = inputTable(:, predictorNames);
response = inputTable.column_289;
isCategoricalPredictor = [false, false, false, false, false, false, false, false

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

A.2.2 Versió 2

```
function [trainedClassifier, validationAccuracy] = trainClassifierversion2_01(tr
% [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: a matrix with the same number of columns and data type
%       as imported into the app.
%
%  Output:
%      trainedClassifier: a struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: a function to make predictions on new
%       data.
%
%      validationAccuracy: a double containing the accuracy in percent. In
%       the app, the History list displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a matrix containing only the predictor columns used for
% training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 03-Jun-2020 03:32:18


% Extract predictors and response
% This code processes the data into the right shape for training the
```

```
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2',

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'c
predictors = inputTable(:, predictorNames);
response = inputTable.column_83;
isCategoricalPredictor = [false, false, false, false, false, false, false, false

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    predictors, ...
    response, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x) svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained model exported from Classific
trainedClassifier.HowToPredict = sprintf('To make predictions on a new predictor
↪ yfit = c.predictFcn(X) \nreplacing ''c'' with the name of the variable that is

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2',

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'c
predictors = inputTable(:, predictorNames);
response = inputTable.column_83;
isCategoricalPredictor = [false, false, false, false, false, false, false, false

```

```
% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

A.2.3 Versió 3 i 4

D'aquestes versions no adjuntem els trainClassifier ja que són molt similars als altres, simplement canvia el número de paràmetres i la templateSVM.

A.2.4 Versió 5

Aquest al ser la millor versió si que el tornem a adjuntar tot i què és bàsicament igual.

```
function [trainedClassifier, validationAccuracy] = trainTest(trainingData)
% [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: a matrix with the same number of columns and data type
%       as imported into the app.
%
%  Output:
%      trainedClassifier: a struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: a function to make predictions on new
%       data.
%
%      validationAccuracy: a double containing the accuracy in percent. In
%       the app, the History list displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
```

```
%   [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a matrix containing only the predictor columns used for
% training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 04-Jun-2020 02:59:25


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2',

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'c
predictors = inputTable(:, predictorNames);
response = inputTable.column_371;
isCategoricalPredictor = [false, false, false, false, false, false, false, false
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 5, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    predictors, ...
    response, ...
    'Learners', template, ...
    'Coding', 'onevsall', ...
    'ClassNames', [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x) svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained model exported from Classifi
trainedClassifier.HowToPredict = sprintf('To make predictions on a new predictor
↪ yfit = c.predictFcn(X) \nreplacing ''c'' with the name of the variable that i
```

```
75
76 % Extract predictors and response
77 % This code processes the data into the right shape for training the
78 % model.
79 % Convert input to table
80 inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2',
81
82 predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'c
83 predictors = inputTable(:, predictorNames);
84 response = inputTable.column_371;
85 isCategoricalPredictor = [false, false, false, false, false, false, false, false
86
87 % Perform cross-validation
88 partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 5);
89
90 % Compute validation predictions
91 [validationPredictions, validationScores] = kfoldPredict(partitionedModel);
92
93 % Compute validation accuracy
94 validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

A.3 Execucions completes de cada versió

A.3.1 Execució completa (v1) amb cerca d'errors

```
1 %version1 Main
2 X=60000;
3 [images, labels]= readMNIST("train-images.idx3-ubyte","
    ↪ train-labels.idx1-ubyte",X,0);
4 z = zeros(X,289);
5 for i = 1:X
6     im=images(:,:,i);
7     imBW=imbinarize(im);
8     z(i,:) = [HOG(imBW), labels(i)];
9 end
10 trainFeatures = z(1:48000,:);
11 testFeatures = z(48001:60000,1:288);
12 testLabels = labels(48001:60000);
13 [classifier, valacc] = trainClassifier(trainFeatures);
14 predictionLabels = classifier.predictFcn(testFeatures);
15 confMat = confusionmat(testLabels, predictionLabels);
16 confusionchart(confMat,[0:1:9],'ColumnSummary','column-
    ↪ normalized','RowSummary','row-normalized');
17 acc = 0;
18 for i=1:10
```

```
acc=acc+confMat(i,i);
end
acc/(120)
error = find(predictionLabels~=testLabels);
s = size(error,1);
errorLabeled = cell(s,3);
for i=1:s
    t = error(i);
    errorLabeled{i,1} = images(:,:,48000+t);
    errorLabeled{i,2} = testLabels(t);
    errorLabeled{i,3} = predictionLabels(t);
end
for i=1:s
    subplot(9,14,i);
    imshow(errorLabeled{i,1}),title("T: " + errorLabeled{
        ↪ i,2} + " G: " + errorLabeled{i,3});
end
sgtitle("Errors, T: True Label, G: Classifier Guess")
```

A.3.2 Execució completa (v2)

```
%version2 Main
X=60000;
[images, labels]= readMNIST("train-images.idx3-ubyte","
    ↪ train-labels.idx1-ubyte",X,0);
z = zeros(X,83);
for i = 1:X
    im=images(:,:,i);
    imBW=imbinarize(im);
    ee = strel('disk',1);
    imCL = imclose(imBW,ee);
    [~, numRegions] = bwlabel(imCL);
    stats = regionprops(imBW,'Area','FilledArea');
    if(size(stats)>1)
        for j=1:size(stats)
            a = stats(j).FilledArea;
            b = stats(j).Area;
            sh = sh + a -b;
        end
    else
        a = stats.FilledArea;
        b = stats.Area;
        sh = a -b;
    end
    z(i,:) = [sh, (numRegions-bweuler(imCL)), pixels(im),
        ↪  sideEntries(imBW), labels(i)];
```

```
end
trainFeatures = z(1:48000,:);
testFeatures = z(48001:60000,1:82);
testLabels = labels(48001:60000);
[classifier, valacc] = trainClassifierversion2_01(
    ↪ trainFeatures);
predictionLabels = classifier.predictFcn(testFeatures);
confMat = confusionmat(testLabels, predictionLabels);
confusionchart(confMat,[0:1:9],'ColumnSummary','column-
    ↪ normalized','RowSummary','row-normalized');
acc = 0;
for i=1:10
acc=acc+confMat(i,i);
end
acc/(120)
```

A.3.3 Execució completa (v3)

```
X=48000;
[imTrain, labsTrain]= readMNIST("train-images.idx3-ubyte
    ↪ ","train-labels.idx1-ubyte",X,0);
Ftrain = zeros(X,57);
for i = 1:X
    im=imbinarize(imTrain(:,:,i));
    C=[];
    H= hough(im);
    t=mean(H(:,1));
    C =[C t];
    for th= 1:55
        t=mean(H(th,:)');
        C=[C t];
    end
    Ftrain(i,:) = [C labsTrain(i)];
end
[classifier, valacc] = trainClassifierversion3(Ftrain);

X = 12000;
[imTest, labsTest]= readMNIST("train-images.idx3-ubyte","
    ↪ train-labels.idx1-ubyte",X,48000);
testF = zeros(X,56);
for i = 1:X
    im=imbinarize(imTest(:,:,i));
     C=[];
    H= hough(im);
    t=mean(H(:,1));
    C =[C t];
```

```
    for th= 1:55
        t=mean(H(th,:)');
        C=[C t];
    end
    testF(i,:) = C;
end
predictedLabels = classifier.predictFcn(testF);

confMat = confusionmat(labsTest, predictedLabels);
confusionchart(confMat,[0:1:9],'ColumnSummary','column-
    ↪ normalized','RowSummary','row-normalized');
acc = 0;
for i=1:10
    acc=acc+confMat(i,i);
end
acc/120
```

A.3.4 Execució completa (v4)

```
X=48000;
[imTrain, labsTrain]= readMNIST("train-images.idx3-ubyte
    ↪ ","train-labels.idx1-ubyte",X,0);
Ftrain = zeros(X,31);
for i = 1:X
    im=imbinarize(imTrain(:,:,i));
    C= fourier(im);
    C=[C' labsTrain(i)];
    Ftrain(i,:)=C;
end
[classifier, valacc] = trainClassifierversion4(Ftrain);

X = 12000;
[imTest, labsTest]= readMNIST("train-images.idx3-ubyte","
    ↪ train-labels.idx1-ubyte",X,48000);
testF = zeros(X,30);
for i = 1:X
    im=imbinarize(imTest(:,:,i));
    C= fourier(im);
    testF(i,:)=C;
end
predictedLabels = classifier.predictFcn(testF);

confMat = confusionmat(labsTest, predictedLabels);
confusionchart(confMat,[0:1:9],'ColumnSummary','column-
    ↪ normalized','RowSummary','row-normalized');
acc = 0;
```

```
for i=1:10
acc=acc+confMat(i,i);
end
acc
acc/120
```

A.3.5 Execució completa (v5) amb cerca d'errors

```
X=60000;
[images, labels]= readMNIST("train-images.idx3-ubyte","
    ↪ train-labels.idx1-ubyte",X,0);
z = zeros(X,371);
for i = 1:X
    im=images(:,:,i);
    imBW=imbinarize(im);
    ee = strel('disk',1);
    imCL = imclose(imBW,ee);
    [~, numRegions] = bwlabel(imCL);
    stats = regionprops(imBW,'Area','FilledArea');
    if(size(stats)>1)
        for j=1:size(stats)
            a = stats(j).FilledArea;
            b = stats(j).Area;
            sh = sh + a -b;
        end
    else
        a = stats.FilledArea;
        b = stats.Area;
        sh = a -b;
    end
    z(i,:) = [sh, (numRegions-bweuler(imCL)), pixels(im),
        ↪ sideEntries(imBW), HOG(imBW), labels(i)];
end
trainFeatures = z(1:48000,:);
testFeatures = z(48001:60000,1:370);
testLabels = labels(48001:60000);
tic
[classifier5, valacc] = trainTest(trainFeatures);
toc
tic
predictionLabels = classifier5.predictFcn(testFeatures);
confMat = confusionmat(testLabels, predictionLabels);
confusionchart(confMat,[0:1:9],'ColumnSummary','column-
    ↪ normalized','RowSummary','row-normalized');
acc = 0;
for i=1:10
```

```
acc=acc+confMat(i,i);
end
acc/(120)
toc
error = find(predictionLabels~=testLabels);
s = size(error,1);
errorLabeled = cell(s,3);
for i=1:s
    t = error(i);
    errorLabeled{i,1} = images(:,:,48000+t);
    errorLabeled{i,2} = testLabels(t);
    errorLabeled{i,3} = predictionLabels(t);
end
for i=1:s
    subplot(9,11,i);
    imshow(errorLabeled{i,1}),title("T: " + errorLabeled{
        ↪ i,2} + " G: " + errorLabeled{i,3});
end
sgtitle("Errors, T: True Label, G: Classifier Guess")
```

A.4 Main Program

Ja amb els classificadors entrenats. Hem guardat els classificadors ja entrenats a un arxiu .mat ja que tarden molt a ser entrenats alguns. Així incloem un load d'un arxiu que no podem adjuntar, però s'obtén entrenant amb les funcions anteriors. Donada una imatge, retorna la etiqueta que el nostre model prediu. Hem optat per usar el model 1, ja que és més ràpid que el 5 i té un error_rate molt semblant igualment.

```
function number = guessNumber(im)
    T = load('.\Classifiers\versio1_0def_model.mat', '
        ↪ trainedClassifier');
    imBW = imbinarize(im);
    number = T.trainedClassifier.predictFcn(HOG(imBW));
end
```

A.5 Aplicació per Presentació

```
T = load('.\Classifiers\versio1_0def_model.mat', '
    ↪ trainedClassifier');
N = 20;
imagesT = zeros(20,20,20);
zTest = zeros(N,288);
for i=1:N
    index = string(i);
    if(i<10)
```

```
8           index = strcat("0",index);
9       end
10      filename = strcat('./Exemples/num1',index,'.png');
11      imagesT(:,:,i) = imread(filename);
12      im = imagesT(:,:,i);
13      imBW = imbinarize(im);
14      zTest(i,:) = HOG(imBW);
15  end
16  predictions = T.trainedClassifier.predictFcn(zTest);
17  for i = 1:N
18      subplot(4,5,i);
19      imshow(imagesT(:,:,i)),title("Guess: " + predictions(
            ↪ i));
20  end
```

B Funcions de Matlab

Descriptors

- imbinarize
- imclose
- extractHOGFeatures
- sum
- fft
- reshape
- regionprops
- bwlabel
- bweuler
- find
- hough
- mean
- resample (obtinguda del Signal Processing Toolbox)
- round
- bwtraceboundary
- size
- imerode

- xor
- abs

Anàlisi de redundància

- fscmrmr
- corrcoef
- pca

Classificadors i mostrar resultats

- fitcecoc (obtinguda del Classification Learner App)
- predictFcn
- confusionmat
- confusionchart