PRÁCTICA 1 - DETECCIÓN AUTOMÁTICA DE EQUIPOS DE FUTBOL EN IMÁGENES

Data

```
imagesMilan = getTeamImages('./soccer/acmilan/', '*.jpg', false);
imagesChelsea = getTeamImages('./soccer/chelsea/', '*.jpg', false);
imagesJuventus = getTeamImages('./soccer/juventus/', '*.jpg', false);
imagesLiverpool = getTeamImages('./soccer/liverpool/', '*.jpg', false);
imagesMadrid = getTeamImages('./soccer/madrid/', '*.jpg', false);
imagesPSV = getTeamImages('./soccer/psv/', '*.jpg', false);
imagesBarcelona = getTeamImages('./soccer/barcelona/', '*.jpg', false);

otherImages = getTeamImages('./soccer/others/', '*.jpg', false);

allImages = getAllImages('./soccer/', '*.jpg', false);
```

Experiment: Obtaining the best model

Basic Barça Model Definition

Basic definitions in order to obtain the possible thresholds:

```
% Preprocessed RGB model images
bcnImageModelRGB1 = imagePreProcessingRGB(imread('./soccer/models/39.jpg'));
bcnImageModelRGB2 = imagePreProcessingRGB(imread('./soccer/models/42.jpg'));
bcnImageModelRGB3 = imagePreProcessingRGB(imread('./soccer/models/43.jpg'));

% Preprocessed HSV model images
bcnImageModelHSV1 = imagePreProcessingHSV(imread('./soccer/models/39.jpg'));
bcnImageModelHSV2 = imagePreProcessingHSV(imread('./soccer/models/42.jpg'));
bcnImageModelHSV3 = imagePreProcessingHSV(imread('./soccer/models/43.jpg'));
```

Obtaining the best thresholds

Trying with:

- Posible color models: RGB and HSV.
- Possible bins: 2^n where n in [1,10]
- Possible distance functions: Euclidean and Chi-Square.

```
bins = [2 4 8 16 32 64 128];

[n, m] = size(bins);
withOutliersResults = cell(4, m);
withoutOutliersResults = cell(4, m);

for i= 1:m
    i
    histModelRGB1 = histogramRGB(bcnImageModelRGB1, bins(i));
    histModelRGB2 = histogramRGB(bcnImageModelRGB2, bins(i));
    histModelRGB3 = histogramRGB(bcnImageModelRGB3, bins(i));

    histModelHSV1 = histogramHSV(bcnImageModelHSV1, bins(i));
    histModelHSV2 = histogramHSV(bcnImageModelHSV2, bins(i));
    histModelHSV3 = histogramHSV(bcnImageModelHSV3, bins(i));

    % 3 modelos a, b, c => a*0.2 + b*0.4 + c*0.6
    bcnModelRGB = {{histModelRGB1, histModelRGB2, histModelRGB3}, {0.2, 0.4, 0.6}};
    bcnModelHSV = {{histModelHSV1, histModelHSV2, histModelHSV3}, {0.2, 0.4, 0.6}};

    %RGB euclidean
    withoutOutliersResults{1, i} = globalExperiment(imagesBarcelona, allImages, bcnModelRGB, bi
    %RGB chi
    withoutOutliersResults{2, i} = globalExperiment(imagesBarcelona, allImages, bcnModelRGB, bi
    %HSV euclidean
    withoutOutliersResults{3, i} = globalExperiment(imagesBarcelona, allImages, bcnModelHSV, bi
    %HSV chi
    withoutOutliersResults{4, i} = globalExperiment(imagesBarcelona, allImages, bcnModelHSV, bi
end
```

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
```

```
accuracyWithoutOutliersResults = arrayfun(@(x) x.accuracy, cell2mat(withoutOutliersResults))
```

```
accuracyWithoutOutliersResults = 4×7
    0.5821    0.6786    0.7536    0.6036    0.6214    0.5750    0.6214
    0.4714    0.5286    0.5643    0.6893    0.7286    0.7500    0.7500
    0.6393    0.6107    0.6750    0.6500    0.5250    0.5036    0.4679
    0.5536    0.5643    0.6250    0.7071    0.7214    0.7321    0.7571
```

The best obtained results are with the following parameterization:

- Color model: HSV
- Distance Function: Chi Square
- Threshold: 0.5691
- Nº of bins: 128

Barça Model Definition

```
% Preprocessed HSV model images
bcnImageModelHSV1 = imagePreProcessingHSV(imread('./soccer/models/39.jpg'));
bcnImageModelHSV2 = imagePreProcessingHSV(imread('./soccer/models/42.jpg'));
bcnImageModelHSV3 = imagePreProcessingHSV(imread('./soccer/models/43.jpg'));

threshold = 0.5691;
bins = 128;
```

```
histModelHSV1 = histogramHSV(bcnImageModelHSV1, bins);
histModelHSV2 = histogramHSV(bcnImageModelHSV2, bins);
histModelHSV3 = histogramHSV(bcnImageModelHSV3, bins);

bcnModelHSV = {{histModelHSV1, histModelHSV2, histModelHSV3}, {0.2, 0.4, 0.6}};
```

Experiments:

```
images = getTeamImages('./soccer/barcelonaothers/', '*.jpg', false);
isTeamParameters = containers.Map({'models', 'threshold'}, {bcnModelHSV, threshold});
experiments(images, bins, @imagePreProcessingHSV, @histogramHSV, isTeamParameters, @compareChiD
```

```
Results table:
results = 1×20 cell
```

...

	1	2	3	4	5	6	7	8
1	[1,0.43...	[1,0.55...	[1,0.54...	[1,0.37...	[1,0.42...	[1,0.48...	[1,0.47...	[1,0.50...

```
Positive cases:
Image 1: 0.43308
Image 2: 0.5529
Image 3: 0.54229
Image 4: 0.37527
Image 5: 0.42124
Image 6: 0.48001
Image 7: 0.47106
Image 8: 0.50328
Image 11: 0.39664
Image 12: 0.4973
Image 13: 0.53077
Image 14: 0.56873
Image 15: 0.56088
Image 16: 0.45236
Image 17: 0.37816
Image 18: 0.56806
Image 20: 0.39664
Total positive cases: 17
Negative cases:
Image 9: 0.60137
Image 10: 0.60188
Image 19: 0.62723
Total negative cases: 3
ans = 1×20 cell
```

...

	1	2	3	4	5	6	7	8
1	[1,0.43...	[1,0.55...	[1,0.54...	[1,0.37...	[1,0.42...	[1,0.48...	[1,0.47...	[1,0.50...

Functions

Function to get all the images (in the same format) from a directory:

- selector == true: Do not specify the path (dirBase), a browser opens on a base direction (dirBase)
- selector == false: Specify the absolute path or a relative path to the 'Current Folder).

```
function images = getTeamImages(dirBase, format, selector)
    % Get specific team (folder)
    d = '';
    if (selector)
        d = uigetdir(pwd, dirBase);
    else
        d = dirBase;
    end

    files = dir(fullfile(d, format));
    [rows cols] = size(files);

    for i = 1:rows
        [num ext] = split(files(i).name, '.');
        files(i).sort = str2num(num{1});
    end
    aux = struct2table(files);
    files = table2struct(sortrows(aux, 7));

    images = cell(1,rows);
    for i=1:rows
        image_file = imread(strcat(files(i).folder, '/', files(i).name));
        images{i} = image_file;
    end
end

% Get all the teams (subfolders of a given root folder)
function images = getAllImages(dirBase, format, selector)
    d = '';
    if (selector)
        d = uigetdir(pwd, dirBase);
    else
        d = dirBase;
    end

    dirs = dir(d);
    [rows cols] = size(dirs);
    images = cell(1,1000);
    k = 1;
    for i=1:size(dirs)
        if (~strcmp(dirs(i).name, 'barcelona') && ~strcmp(dirs(i).name, 'models')...
                && ~strcmp(dirs(i).name, '.') && ~strcmp(dirs(i).name, '..') ...
                && ~strcmp(dirs(i).name, 'others') && ~strcmp(dirs(i).name, 'barcelonaothers'))
            files = dir(fullfile(strcat(d, dirs(i).name, '/'), format));

            [rows cols] = size(files);
            for j=1:rows
                image_file = imread(strcat(files(j).folder, '/', files(j).name));
                images{k} = image_file;
                k = k + 1;
            end
        end
    end
    images = images(~cellfun('isempty',images));
end
```

Image preprocessing functions (RGB and HSV):

```
% RGB preprocessing
function impp = imagePreProcessingRGB(im)
    % Image components extraction
    r = im(:,:,1);
    g = im(:,:,2);
    b = im(:,:,3);

    % Color normalization
    I = double(r) + double(g) + double(b);
    Rn = double(r) ./ I;
    Gn = double(g) ./ I;
    Bn = double(b) ./ I;

    impp = uint16(cat(3, Rn, Gn, Bn) * 255);
end

% HSV preprocessing
function img = imagePreProcessingHSV(im)
    img = uint16(rgb2hsv(im) * 255);
end
```

Euclidean Distance (L2) function:

```
function result = compareEuclideanDistance(hist1, hist2)
    [n m] = size(hist1);
    dist = 0;
    for i=1:n
        for j=1:m
            aux = (hist1(i, j) - hist2(i, j));
            dist = dist + (aux*aux);
        end
    end
    result = sqrt(dist);
end
```

Chi-Square Distance function:

```
function result = compareChiDistance(hist1, hist2)
    hist1 = imgaussfilt(hist1,2);
    hist2 = imgaussfilt(hist2,2);
    [n m] = size(hist1);
    dist = 0;
    for i=1:n
        for j=1:m
            aux = hist1(i, j) - hist2(i, j);
            if (hist1(i, j) + hist2(i, j) > 0)
                dist = dist + (aux*aux)/(hist1(i, j) + hist2(i, j));
            end
        end
    end
    result = dist / 2;
end
```

Bidimensional Histogram (without Green/V component) functions (RGB and HSV):

```
% RGB Histogram
function h = histogramRGB(im, bins)
    R = im(:,:,1);
    %G = im(:,:,2);
    B = im(:,:,3);

    h = zeros(bins, bins);
    [n m] = size(R);
    for i = 1:n
        for j = 1:m
            r = R(i, j)/ (256/bins) + 1;
            b = B(i, j)/ (256/bins) + 1;
            if (r > bins)
                r = r - 1;
            end
            if (b > bins)
                b = b - 1;
            end
            h(r, b) = h(r, b) + 1;
        end
    end
    h = h ./ (n*m);
end

%HSV Histogram
function hist = histogramHSV(im, bins)
    H = im(:,:,1);
    S = im(:,:,2);
    %V = im(:,:,3);

    hist = zeros(bins, bins);
    [n m] = size(H);
    for i = 1:n
        for j = 1:m
            h = floor(H(i, j) / (256/bins)) + 1;
            s = floor(S(i, j) / (256/bins)) + 1;
            if (h > bins)
                h = h - 1;
            end
            if (s > bins)
                s = s - 1;
            end
            hist(h, s) = hist(h, s) + 1;
        end
    end

    % histogram normalization
    hist = hist ./ (n*m);
end
```

Function to check if in a given histogram there are some team (for example: barça):

- models: Matrix of pairs (image model histogram in RGB/HSV).
- hist: Image histogram to check (in the same color model as models).
- distF: Function to compute te distance.
- threshold: Distances < threshold will not belong to the team

```
function [is dist] = isTeam(models, hist, distF, threshold)
    dist = calculateDistance(models, hist, distF);
    is = dist < threshold;
end
```

Function to carry out the experiments:

- hsitoBins: Bins to compute te histogram.
- isTeamParameters: Models and threshold of the function isTeam.
- distF: Distance function.

```
function results = experiments(imgs, histoBins, preprocessingF, histogramF, isTeamParameters, d
    [n m] = size(imgs);

    % Obtaining histogram of preprocessed images
    histograms = cell(1, m);
    for i=1:m
        histograms{i} = histogramF(preprocessingF(imgs{i}), histoBins);
    end

    % Compute results (output)
    results = cell(1,m);
    for i = 1:m
        [is dist] = isTeam(isTeamParameters('models'), histograms{i}, distF, isTeamParameters('
        results{i} = [is dist];
    end

    % Show results
    disp('Results table:');
    results

    % Print positive cases
    disp('Positive cases:');
    countPos = 0;
    for i = 1:m
        if (results{1,i}(1))
            disp(['Image ', num2str(i), ': ',  num2str(results{1,i}(2))]);
            countPos = countPos + 1;
        end

    end
    disp(['Total positive cases: ', num2str(countPos)]);
    % Print negative cases
    disp('Negative cases:');
    countNeg = 0;
    for i = 1:m
        if (~results{1,i}(1))
            disp(['Image ', num2str(i), ': ',  num2str(results{1,i}(2))]);
            countNeg = countNeg + 1;
        end
    end
    disp(['Total negative cases: ', num2str(countNeg)]);


end
```

Function to experiment with a group of positive images and negatives images:

- positiveImages: List of images of the positive class
- negativeImages: List of images of the negative class
- models: models to calculate the distance
- bins: number of bins of the histograms
- preprocessingF: preprocessing function
- histogramF: histogram function
- distF: distance function

```
function result = globalExperiment(positiveImages, negativeImages, models, bins, preprocessingF
    % Preprocessing images
    positiveImages = preprocessImages(positiveImages, preprocessingF);
    negativeImages = preprocessImages(negativeImages, preprocessingF);

    % Obtaining histogram of preprocessed images
    positiveHistograms = getHistograms(positiveImages, histogramF, bins);

    negativeHistograms = getHistograms(negativeImages, histogramF, bins);

    % Calculate the distance of every histogram to the models
    positiveDistances = calculateHistogramsDistance(models, positiveHistograms, distF);
    negativeDistances = calculateHistogramsDistance(models, negativeHistograms, distF);

    % Calculate the optimal threshold
    threshold = 0;
    if (withOutliers)
        threshold = calculateThresholdWithOutliers(positiveDistances, negativeDistances);
    else
        threshold = calculateThresholdWithoutOutliers(positiveDistances, negativeDistances);
    end

    % Compute positive results (output)
    [n m1]= size(positiveDistances);
    truePositives = 0;
    for i=1:m1
        if (positiveDistances(i) < threshold)
```

```
            truePositives = truePositives + 1;
        end
    end
    falsePositives = m1 - truePositives;

    result.truePositivesRatio = truePositives / m1;
    result.falsePositivesRatio = 1 - result.truePositivesRatio;

    % Compute negatives results
    [n m2]= size(negativeDistances);
    falseNegatives = 0;
    for i=1:m2
        if (negativeDistances(i) < threshold)
            falseNegatives = falseNegatives + 1;
        end
    end
    trueNegatives = m2 - falseNegatives;

    result.falseNegativesRatio = falseNegatives / m2;
    result.trueNegativesRatio = 1 - result.falseNegativesRatio;

    result.confusionMatrix = [truePositives falsePositives, falseNegatives trueNegatives];

    result.accuracy = (truePositives + trueNegatives)/(m1 + m2);
    result.threshold = threshold;
end
```

Function to calculate the optimal threshold to diferentiate between two classes given the distances of the two group of images to the model

- positiveResults: List of distances of the positive class images to a model
- negativeResults: List of distances of the negative class images to a model

```
function t = calculateThresholdWithOutliers(positiveResults, negativeResults)
    t = (max(positiveResults) + min(negativeResults))/2;
end

function t = calculateThresholdWithoutOutliers(positiveResults, negativeResults)
    positiveResultsWithoutOutliers = filloutliers(positiveResults,'linear', 'percentiles', [10
    negativeResultsWithoutOutliers = filloutliers(negativeResults,'linear', 'percentiles', [10

    t = (max(positiveResultsWithoutOutliers) + min(negativeResultsWithoutOutliers))/2;
end
```

Function to calculate the preprocessing of a list of images

- images: List of images
- preprocessingF: Function to compute the preprocessing.

```
function preprocessedImages = preprocessImages(images, preprocessingF)
    [n m] = size(images);
    preprocessedImages = cell(1, m);
    for i=1:m
        preprocessedImages{i} = preprocessingF(images{i});
    end
end
```

Function to calculate the histogram of a list of images

- images: List of images
- histogramF: Function to compute the histogram.
- bins: number of bins of the histogram

```
function histograms = getHistograms(images, histogramF, bins)
    [n m] = size(images);
    histograms = cellfun(@(x) histogramF(x, bins), images, 'Uni', false);
end
```

Function to calculate the distance of a list of histograms to a group of models

- models: Matrix of pairs (image model histogram in RGB/HSV).
- histograms: List of histograms
- distF: Function to compute te distance.

```
function results = calculateHistogramsDistance(models, histograms, distF)
    [n m] = size(histograms);
    results = zeros(m, 0);

    for i=1:m
        results(i) = calculateDistance(models, histograms{i}, distF);
    end
end
```

Function to calculate the distance between a certain histogram and a sum of ponderated models

- models: Matrix of pairs (image model histogram in RGB/HSV).
- hist: Image histogram to check (in the same color model as models).
- distF: Function to compute te distance.

```
function dist = calculateDistance(models, hist, distF)
    dist = 0.0;
    sumPond = 0.0;
    [n m] = size(models);

    for i = 1:n
        dist = dist + models{2}{i} * distF(hist, models{1}{i});
        sumPond = sumPond + models{2}{i};
    end

    dist = dist/sumPond;
```

```
end
```