Laboratory 4: Face Detection by the Viola-Jones' algorithm and deep-learning

Jesus Miguel Adrian Matos

July 4, 2024

Abstract

nothing

1 Inputs:

Our inputs are

- An image 2D to perform face detection.
- The OpenCV file **haarcascade_frontalface_alt.xml** that contains the feature functions to detect faces.
- Scale update option that updates the size of the windows (pixel array).

The **windows** are the array of pixels where the feature functions are applied, so they have many scales to search for faces of many sizes.

2 Outputs:

The code returns us an matrix $[xywidthheight]$ with a dimension of $n \times 4$ of the detected objects such that:

- n: Number of objects detected.
- x: Coordinate of the row where the object in the image begins.
- y: Coordinate of the column where the object in the image begins.
- $width$: Width of the detected object.
- $height$: Height of the detected object.

We will call each row of this matrix **Rectangle** $Rectangle^i$, in figure 1 we can see the example of a $Rectangle^i$ matrix within an image.

3 First Part (ViolaJones):

The code for Viola Jones is made up of the following files

- **ConvertHaarcasadeXMLOpenCV.m**: This function is required to translate OpenCV feature functions from XML to syntax Matlab can understand.
- **ObjectDetection.m**: This is the beginning of the detection where the input parameters are received and the function to return the **Rectangle** matrices is called.
- **GetIntergralImages.m**: This function computes the **integral matrix** for the image.

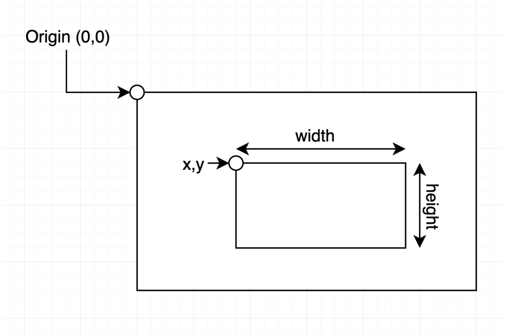


Figure 1: *Rectangle*^{i} matrix.

3.1 Code:

Let's start by describing what is important about the codes:
The first code called would be **ObjectDetection.m** Listing 2:
On line 3 of the Listing 2, we have the default options:

- **ScaleUpdate**: This tells us that the scale of the current **window** will increase from 1 to 1.2 in each iteration, that is, if the current scale is X, the next scale will be $1.2X$.
- **Resize**: Resizes the image so that the longest side (width or height) is equal to 384.
- **Verbose**: this is to show the calculations of the iterations.

On line 31 of the Listing 2, here we load the image and convert it into an array, but if the image is color then the pixes of the array has 3 values in eache pixel (r,g,v).

On line 35 of the Listing 2, this function extracts the feature functions already trained by **OpenCV**, which are our **Strong Classifiers** when examining each **window**, as shown in class slide Figure 2.

On line 37 of the Listing 2,we compute the **integral matrix** using the function **GetIntergralImages.m** (see Listing 1)

On line 1 of the Listing 1, we convert the image array to a double-type.

On lines from 2 to 12 of the Listing 1, the image resize mentioned is performed (option resize) and the reason for the ratio between the real size of the image and its current size is saved $Ratio = size(Picture, 2)/384$.

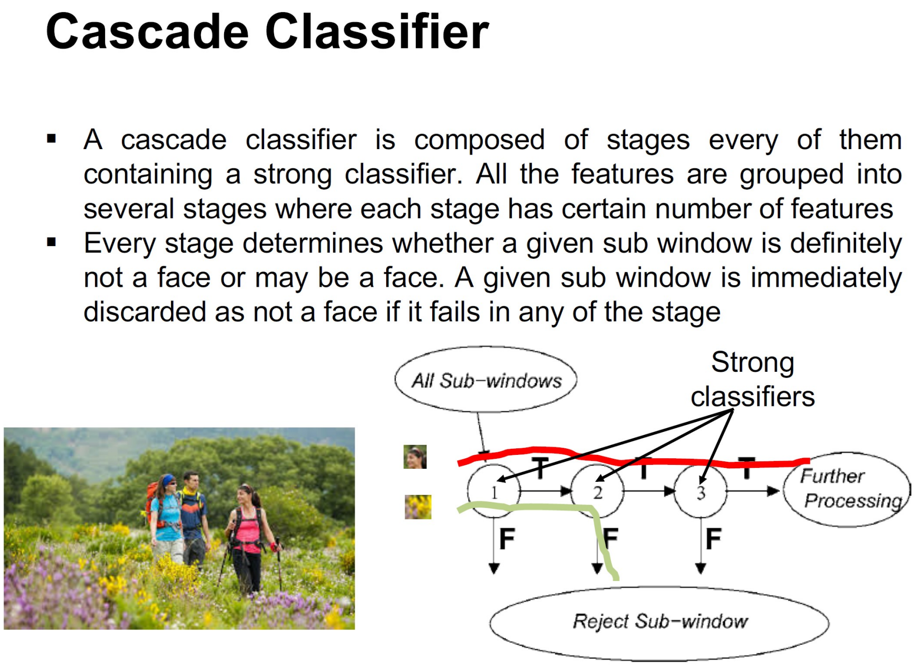
```
1   Picture=im2double(Picture);
2
3   if(Options.Resize)
4   if (size(Picture ,2) > size(Picture ,1)),
5           Ratio = size(Picture ,2) / 384;
6   else
7           Ratio = size(Picture ,1) / 384;
8   end
9           Picture = imresize(Picture, [size(Picture ,1) size(Picture ,2) ]/
                Ratio);
10  else
11          Ratio=1;
```

```
end

if(size(Picture,3)>1),
        Picture=0.2989*Picture(:,:,1) + 0.5870*Picture(:,:,2)+ 0.1140*
           Picture(:,:,3);
end

s=zeros([size(Picture,1),size(Picture,2)]);
ii=zeros([size(Picture,1),size(Picture,2)]);
for x=1:size(Picture,1)
        for y=1:size(Picture,2)
                if(x==1 && y==1)
                        s(x,y)=Picture(x,y);
                        ii(x,y)=s(x,y);
                elseif(x==1 && y>1)
                        s(x,y)=s(x,y-1)+Picture(x,y);
                        ii(x,y)=s(x,y);
                elseif(x>1 && y==1)
                        s(x,y)=Picture(x,y);
                        ii(x,y)=ii(x-1,y)+s(x,y);
                else
                        s(x,y)=s(x,y-1)+Picture(x,y);
                        ii(x,y)=ii(x-1,y)+s(x,y);
                end
        end
end
IntegralImages.ii=ii;

IntegralImages.ii=padarray(IntegralImages.ii,[1 1], 0, 'pre');

s2=zeros([size(Picture,1),size(Picture,2)]);
ii2=zeros([size(Picture,1),size(Picture,2)]);
for x=1:size(Picture,1)
        for y=1:size(Picture,2)
                if(x==1 && y==1)
                        s2(x,y)=Picture(x,y)^2;
                        ii2(x,y)=s(x,y);
                elseif(x==1 && y>1)
                        s2(x,y)=s(x,y-1)+Picture(x,y)^2;
                        ii2(x,y)=s(x,y);
                elseif(x>1 && y==1)
                        s2(x,y)=Picture(x,y)^2;
                        ii2(x,y)=ii2(x-1,y)+s2(x,y);
                else
                        s2(x,y)=s2(x,y-1)+Picture(x,y)^2;
                        ii2(x,y)=ii2(x-1,y)+s2(x,y);
                end
        end
end
IntegralImages.ii2=ii2;

IntegralImages.ii2=padarray(IntegralImages.ii2,[1 1], 0, 'pre');

IntegralImages.width = size(Picture,2);
IntegralImages.height = size(Picture,1);
IntegralImages.Ratio=Ratio;
```

Listing 1: Code **GetIntergralImages.m**

```
        function Objects = ObjectDetection(Picture,FilenameHaarcasade,Options)

        defaultoptions=struct('ScaleUpdate',1/1.2,'Resize',true,'Verbose',true);

        functionname='ObjectDetection.m';
        functiondir=which(functionname);
        functiondir=functiondir(1:end-length(functionname));
        addpath([functiondir '/SubFunctions'])

        if(ischar(Picture))
                if(~exist(Picture,'file'))
                        error('face_detect:inputs','Image not Found');
                end
        end
        if(~exist(FilenameHaarcasade,'file'))
                error('face_detect:inputs','Haarcasade not Found');
        end

        if(~exist('Options','var')), Options=defaultoptions;
        else
                tags = fieldnames(defaultoptions);

        for i=1:length(tags),
                if(~isfield(Options,tags{i})), Options.(tags{i})=defaultoptions.(tags{i}); end
        end
                if(length(tags)~=length(fieldnames(Options))),
                        warning('image_registration:unknownoption','unknown options found');
                end
        end

        if(ischar(Picture))
                Picture = imread(Picture);
        end

        HaarCasade=GetHaarCasade(FilenameHaarcasade);

        IntergralImages= GetIntergralImages(Picture,Options);

        Objects = HaarCasadeObjectDetection(IntergralImages,HaarCasade,Options);

        if(nargout==0)
                ShowDetectionResult(Picture,Objects);
        end
```

Listing 2: Code **ObjectDetection.m**



Cascade Classifier

- A cascade classifier is composed of stages every of them containing a strong classifier. All the features are grouped into several stages where each stage has certain number of features
- Every stage determines whether a given sub window is definitely not a face or may be a face. A given sub window is immediately discarded as not a face if it fails in any of the stage

Figure 2: Cascade Classifier