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

Jesus Miguel Adrian Matos

July 4, 2024

Abstract

This laboratory deals with face detection algorithms. The work is organised as follows:

- Section 1: we explain the inputs.
- Section 2: we explain the structure of the results, and the shape of the resulting matrix
- Section 3: we explain the code for ViolaJones.
- Section 4: we explain .

1 Inputs:

To develop the task of Laboratory 4, 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 a matrix $[x\,y\,width\,height]$ 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.

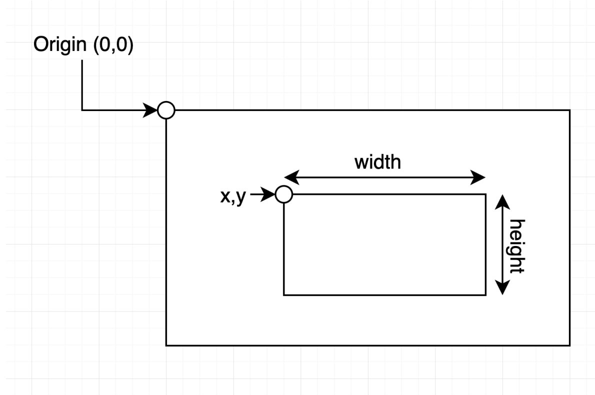


Figure 1: *Rectangle*^{i} matrix.

3 First Part (ViolaJones):

The code for Viola Jones is composed by the following files

- **ConvertHaarcasadeXMLOpenCV.m**: This function is required to translate OpenCV feature functions from XML to syntax Matlab in order to be compiled.
- **ObjectDetection.m**: This is the beginning of the detection, where the input parameters are received and invokes the functions **GetIntergralImages.m** and **HaarCasadeObjectDetection.m**.
- **GetIntergralImages.m**: This function computes the **integral matrix** for the image.
- **HaarCasadeObjectDetection.m**: This function uses the Cascade Classifier to calculate the **Rectangle** matrices.

3.1 Code:

Let start by describing what is important about the codes.
The first code called is **ObjectDetection.m** Listing 3.
On line 3 of the Listing 3, 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. Thus, 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**: To show the calculations of the iterations.

On line 31 of the Listing 3, we load the image and convert it into an array, but if the image is coloured then the pixels of the array has 3 values in each pixel (r, g, v).

On line 35 of the Listing 3, this function extracts the feature functions already trained by **OpenCV**, which are our **Strong Classifiers** when examining each **window**, as shown in lecture slide in Figure 2.

On line 37 of the Listing 3,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). The reason for the ratio between the real size of the image and its current size is saved $Ratio = size(Picture, 2)/384$.

On lines from 14 to 16 of the Listing 1, here we convert the image to garyscale, which means, that our pixels no longer have 3 values (r,g,b) buy have a single value.

On lines from 18 to 39 of the Listing 1, here we compute the **integral matrix** for the image, in the Figure 3 you can see how compute the **integral matrix**.
On lines from 41 to 62 of the Listing 1, here we compute the **integral matrix** for $I2$, where $I2$ is

$$\text{let } I = \begin{pmatrix} i_{11} & i_{12} & \cdots & i_{1n} \\ i_{21} & i_{22} & \cdots & i_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ i_{m1} & i_{m2} & \cdots & i_{mn} \end{pmatrix} \Rightarrow I2 = \begin{pmatrix} (i_{11})^2 & (i_{12})^2 & \cdots & (i_{1n})^2 \\ (i_{21})^2 & (i_{22})^2 & \cdots & (i_{2n})^2 \\ \vdots & \vdots & \ddots & \vdots \\ (i_{m1})^2 & (i_{m2})^2 & \cdots & (i_{mn})^2 \end{pmatrix} \tag{1}$$

On lines from 64 to 66 of the Listing 1, here we only add the **height** and **width** of the current image (with a maximum side like 384), and the **ratio** of change of the image resize to structure $IntegralImages$.
On line 39 of the Listing 3, we call the function $HaarCasadeObjectDetection$, which applies cascade classification to find the **Rectangle** matrices that contain the faces that appear in the image.
On line from 1 to 7 of the Listing 2, we calculate the initial window size.

On line 11 of the Listing 2, we calculate the number of total iterations, that is, the number of different **window** sizes that there exist with the $Options.ScaleUpdate$.

On line from 13 to 35 of the Listing 2, we compute the **Rectangle** matrices using the properties of the **integral matrix** 4.

On line 9 of the Listing 2, we transform the **Rectangle** matrices to the actual size of the image.

3.2 results:

Here we show an image where some faces are clearly visible, which are enclosed in the **Rectangle**s that can be seen in Figure 5.

And in figure 6 we can see how many faces were found per scale.

3.3 Listing of the Codes

```
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;
12 end
```

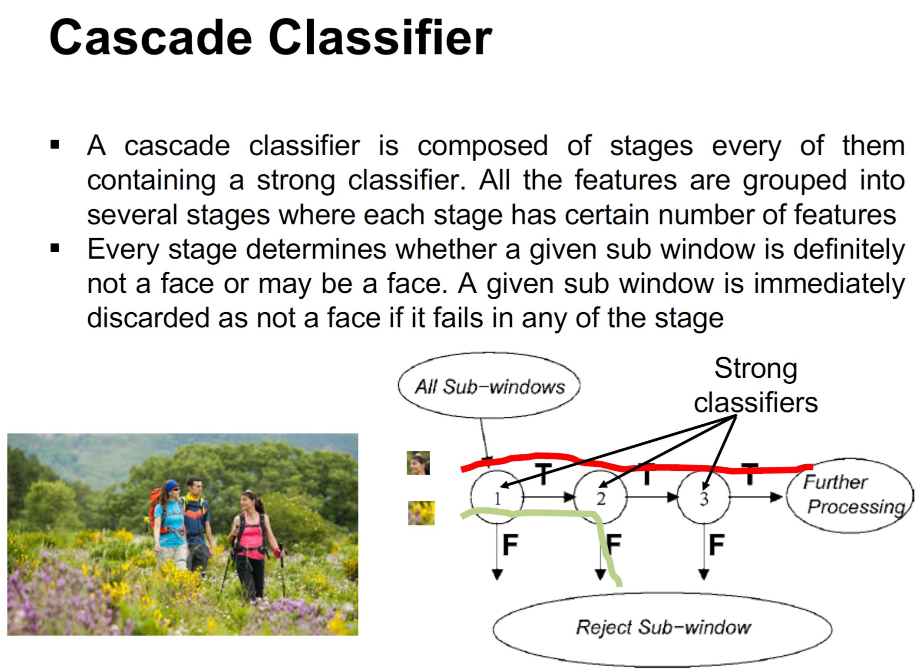


Figure 2: Cascade Classifier

```matlab

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;

```

Integral Image

In an integral image ii(x,y), the value at pixel i(x,y) contains the sum of pixels above and to the left of (x,y)

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

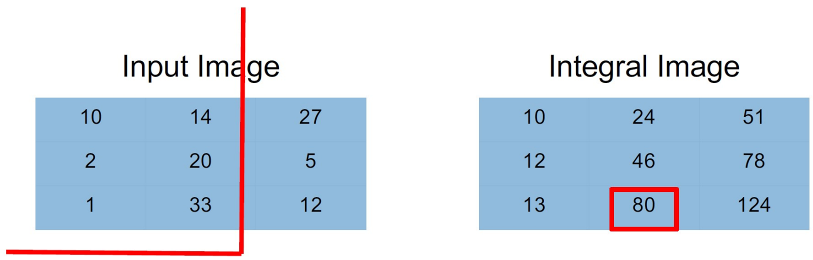
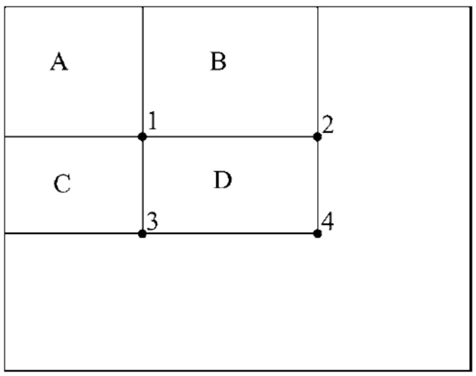


Figure 3: Integral Image calculation

```
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);
```

Integral Image

Integral image allows us computing the sum of all pixels inside any given rectangle (any scale or location) by using only *four values* at the corners of the rectangle, and in *constant time*



The sum of all pixel in D is:

=1+4 –(2+3)

=A+(A+B+C+D)-(A+C+A+B)

=D

Figure 4: Integral Image properties

```
IntegralImages.Ratio=Ratio;
```

Listing 1: Code **GetIntergralImages.m**

```
ScaleWidth = IntegralImages.width/HaarCasade.size(1);
ScaleHeight = IntegralImages.height/HaarCasade.size(2);
if(ScaleHeight < ScaleWidth ),
        StartScale =  ScaleHeight;
else
        StartScale = ScaleWidth;
end

Objects=zeros(100,4); n=0;

itt=ceil(log(1/StartScale)/log(Options.ScaleUpdate));

for i=1:itt
        % Current scale
        Scale =StartScale*Options.ScaleUpdate^(i-1);

        if(Options.Verbose)
                disp(['Scale : ' num2str(Scale) ' objects detected : '
                   num2str(n) ])
        end

        w = floor(HaarCasade.size(1)*Scale);
        h = floor(HaarCasade.size(2)*Scale);
```

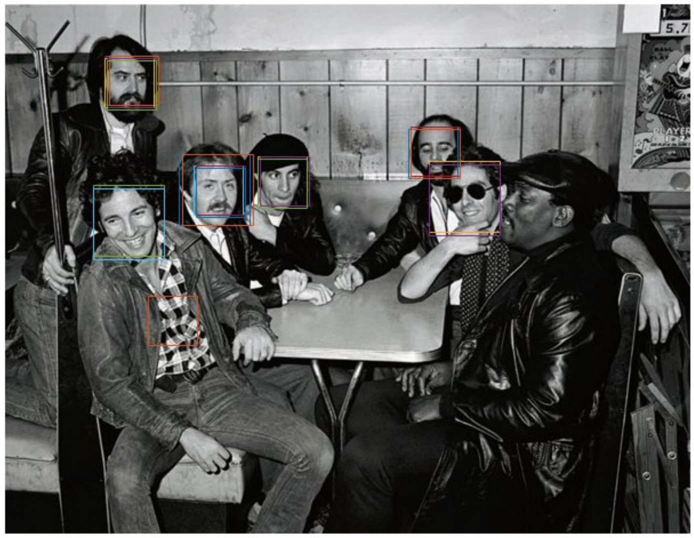


Figure 5: This image contains the detected **Rectangle**

```
        step = floor(max( Scale, 2 ));

        [x,y]=ndgrid(0:step:(IntegralImages.width-w-1),0:step:(IntegralImages.height-h-1)); x=x(:); y=y(:);

        if(isempty(x)), continue; end

        [x,y] = OneScaleObjectDetection( x, y, Scale, IntegralImages, w, h, HaarCasade);

        for k=1:length(x);
                n=n+1; Objects(n,:)=[x(k) y(k) w h];
        end
end

Objects=Objects(1:n,:);

Objects=Objects*IntegralImages.Ratio;
```

Listing 2: Code **HaarCasadeObjectDetection.m**

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

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

```
Scale : 23.15 objects detected : 0
Scale : 19.2917 objects detected : 0
Scale : 16.0764 objects detected : 0
Scale : 13.397 objects detected : 0
Scale : 11.1642 objects detected : 0
Scale : 9.3035 objects detected : 0
Scale : 7.7529 objects detected : 0
Scale : 6.4607 objects detected : 0
Scale : 5.384 objects detected : 0
Scale : 4.4866 objects detected : 0
Scale : 3.7389 objects detected : 0
Scale : 3.1157 objects detected : 0
Scale : 2.5964 objects detected : 6
Scale : 2.1637 objects detected : 6
Scale : 1.8031 objects detected : 23
Scale : 1.5026 objects detected : 23
Scale : 1.2521 objects detected : 23
Scale : 1.0434 objects detected : 23
```

Figure 6: These are the **Rectangle** found in each scale

```

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 3: Code **ObjectDetection.m**