Finding a Face in a Sea of Faces

Tidewater Big Data Enthusiasts
Chuck Cartledge
Developer

June 7, 2020

Contents

List of Tables

List of Figures

1 Introduction

Often times when a group gets together, for whatever reason, there will be a group picture at the end to commemorate the good times had by all. If this "sea of faces" gets published, in hard or soft copy, there may be a one or two line caption giving the name of the group and perhaps where and when the image was created. Six months or a year later, the image has only marginal value to the people who were there, and almost no value to those who were not there, because it is just a sea of faces.

We are interested in finding a low cost (very little human time) method of providing a way to add value to the soft copy of the image, so the image will have greater value later. We have developed a Python script that uses a Haar facial detection cascade to create a clickable HTML image map. The image map can be incorporated into other HTML pages to support a dynamic and valuable web experience.

The Python script, Haar file, and sample images are included in this report.

2 Purpose

Picking out faces in a crowded image to create an image map can be time consuming and error prone. An automated, or semi-automated tool that can assist a human in this type of tedious task would be beneficial. The utility and usability of current online sites varies considerably.

Our purpose is to automate the heavy lifting as much as possible, to keep the end product as agnostic as possible, and to minimize the time a human spends "diddling" the image to a minimum. To those ends, we:

1. Decided to use a Haar cascade facial detection approach because it is fast and well known,
2. Decided to use Python 3 because it available for both Windows and Unix distributions,
3. Decided to use native image viewers to support the facial detection process, and
4. Tried to make the application as simple as possible.

3 Finding faces

Detecting faces depends on several factors and actions acting together. We will start with a quick over view of the program internals, followed by how to execute the Python script from the command line, what the script presents to the user as it is detecting faces, and finally what the user has to do to accept those detected areas.

3.1 Program overview

The embedded Python script uses the `cv2.CascadeClassifier` as the heavy lift engine for detecting faces using a Haar front face cascade. The script was written, and tested on a Ubuntu Linux distribution, and should run with only minor changes in a Windows environment.

The script has one global dictionary, and several functions. The default global dictionary is:

```
ffGlobals = {
    "scaler": 0.1,
    "scaleFactor": 1,
    "HaarFile": "./haarcascade-frontalface-default.xml",
    "htmlFile": "./temp.html",
    "inFile": "./ws-dl-group-1.jpeg",
    "markedUpFile": "./tempMarkedUp.jpg",
    "monitorFile": "./tempMonitoring.jpg"
}
```

The `ffGlobals` dictionary (see Table 1) contains the default input, output, various temporary image file names, the percentage of change when an image is increased, or decreased (the default value seems to work reasonably well), and `scaleFactor` for internal use. Most of the `ffGlobals` values can be changed via command line arguments (see Table 2).

3.2 Program execution

Normal program execution using all default values:

```
python findFaces.py
```

Program execution using previously stored values:

```
python findFaces.py -L ffGlobalsFile
```

Program execution using previously stored values, and changing the destination:

```
python findFaces.py -L ffGlobalsFile -o different.html
```

Image processing and facial detection has two distinct phases:

1. Detecting "face-like" pixel patterns in the image.

Table 1: The `ffGlobals` dictionary. Various input and output files defaulted by the script. These values will need to be changed for the script to execute in a Windows environment. Temporary files are removed after program execution.

Name	**Value**	**Usage**
scaler	0.1	The amount that the working image will be increased, or decreased before preforming facial detection. Can be replaced by command line argument.
scaleFactor	1	A scaling factor used for internal purposes.
HaarFile	./haarcascade-frontalface-default.xml	The Haar file used to detect faces.Can be replaced by command line argument.
htmlFile	./temp.html	Where the HTML file will be created. Can be replaced by command line argument.
inFile	./ws-dl-group-1.jpeg	The source image. Can be replaced by command line argument.
markedUpFile	./tempMarkedUp.jpg	A temporary image file used to see how the final web page will look/operate. Can be replaced by command line argument.
monitorFile	./tempMonitoring.jpg	A temporary image file used to see how many faces have been found. Can be replaced by command line argument.

Table 2: Program command line arguments. Command line options and arguments can be in any order. All commands, except `R`, are followed by one textual arguments. Saving will most likely be used as the last command. Loading will most likely be used as the first command.

Arg.	**Default**	**Explanation**
?	N/A	Show program usage, and exit.
D	N/A	Show current values in `ffGlobals`.
i	`ffGlobals[''inFile'']`	The path to the input image.
o	`ffGlobals[''htmlFile'']`	The path and name of the output HTML file.
H	`ffGlobals[''HaarFile'']`	The path and name of the Haar detection file.
m	`ffGlobals[''monitorFile'']`	The path and name of the image file used to monitor program execution.
M	`ffGlobals[''markedUpFile'']`	The path and name of the image file used to show the image map while being constructed.
s	`ffGlobals[''scaler'']`	How much to increase/decrease the image, 0.1 is 10%.
S	N/A	Where to save the current `ffGlobals` for later use. Should be the last set of arguments. Saved data can be reloaded with the `L` option.
L	N/A	Where to load a previous set of `ffGlobals`. This option is the complement of the `S` option.
R	N/A	Reset `ffGlobals` to the original program values.

2. Accepting, or rejecting the detections.

These phases are explained in the following sections.

It is recommended that an external image viewer display `ffGlobals[''monitorFile'']` to monitor program execution.

3.3 Detecting faces

The Haar detector works on pixels, and their relationship to pixels around them. If adjacent pixels don't have the relationship that the Haar[1] detector is expecting then the pixels are rejected. When a human looks at an image, the human may see patterns there that Haar does not. Either the pixel pattern is the wrong size, or the differences in the pixels are not distinct enough.

For our purposes, we look at the pixel patterns being the wrong size. So we change the size of the image, run the detector again, and evaluate the results. The software changes a copy of the original image by $\pm 10\%$ (within the bounds of integer math).

The current facial detections are written to the `ffGlobals[''monitorFile'']` file after each detection attempt. The `ffGlobals[''monitorFile'']` should be viewed with an image browser to monitor the current detection state (see Figures 1 through 7).

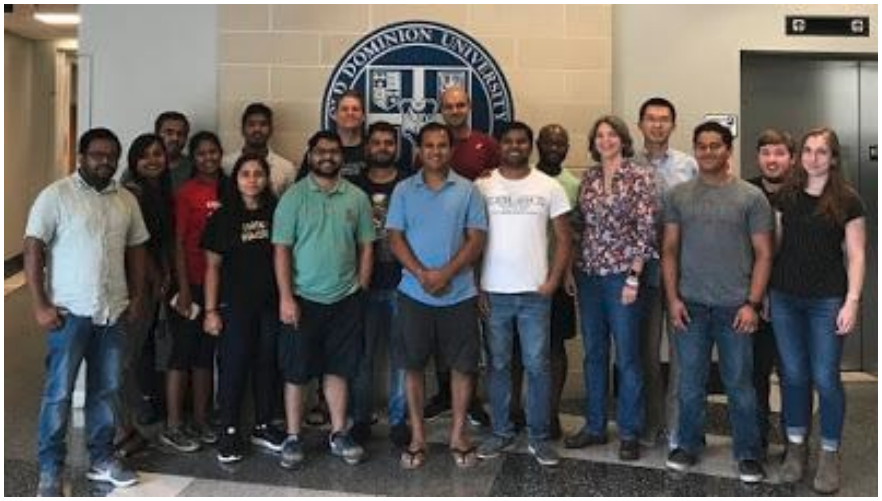


Figure 1: Iteration #01 of facial detection. Original image. No faces detected.

[1]Additional Haar detectors are available from: `https://github.com/opencv/opencv/tree/master/data`

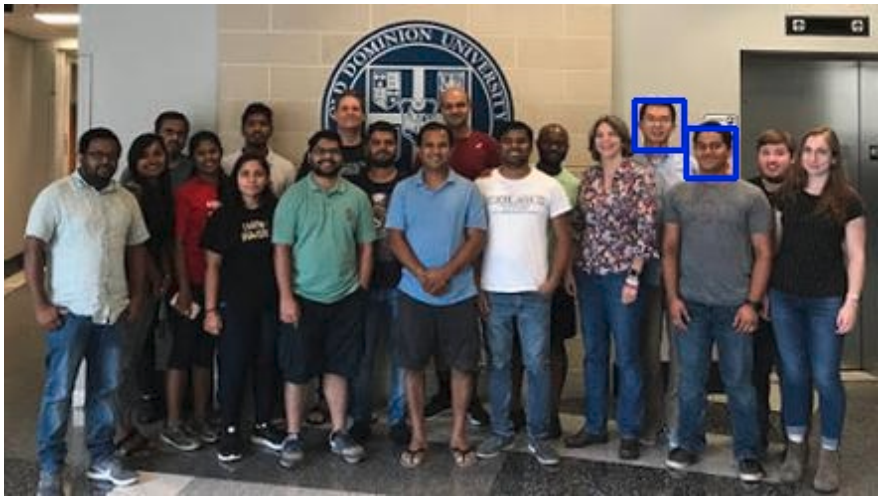


Figure 2: Iteration #02 of facial detection. Increased image size by 10%, 2 faces detected. (The processed image is greater than previous image, even though they appear the same size on the page.)

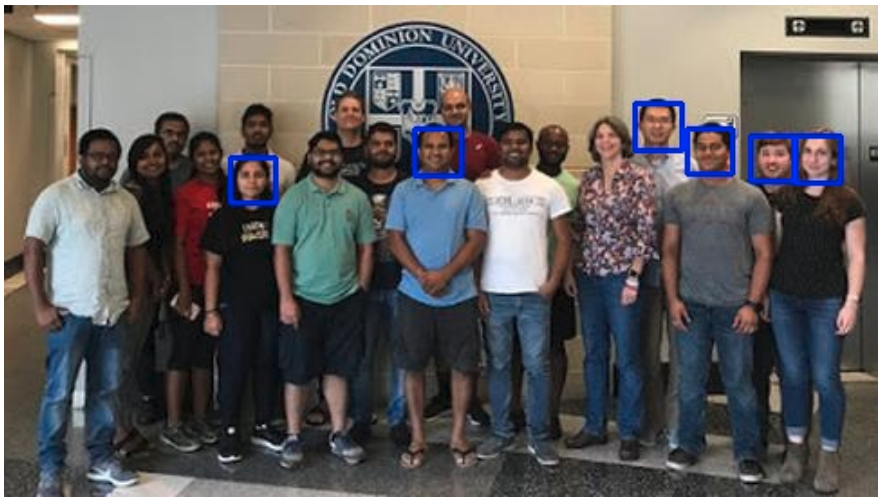


Figure 3: Iteration #03 of facial detection. Increased image size by 10%, 6 faces detected. (The processed image is greater than previous image, even though they appear the same size on the page.)

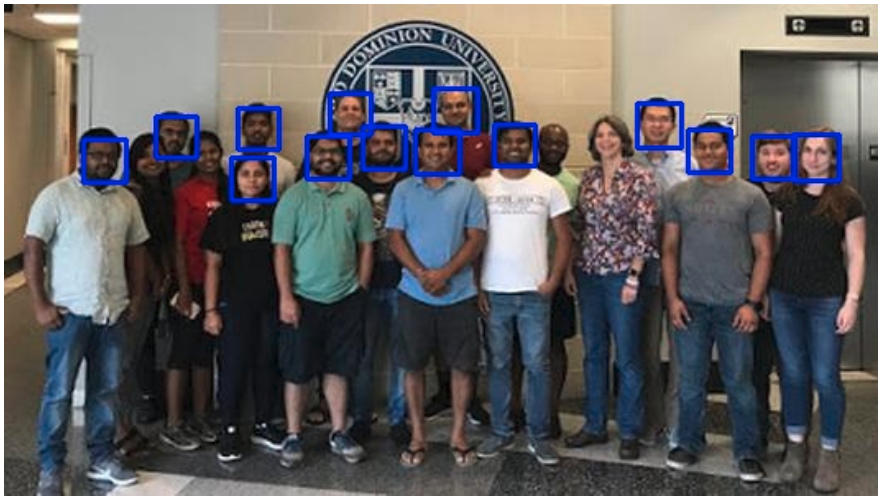


Figure 4: Iteration #04 of facial detection. Increased image size by 10%, 14 faces detected. (The processed image is greater than previous image, even though they appear the same size on the page.)

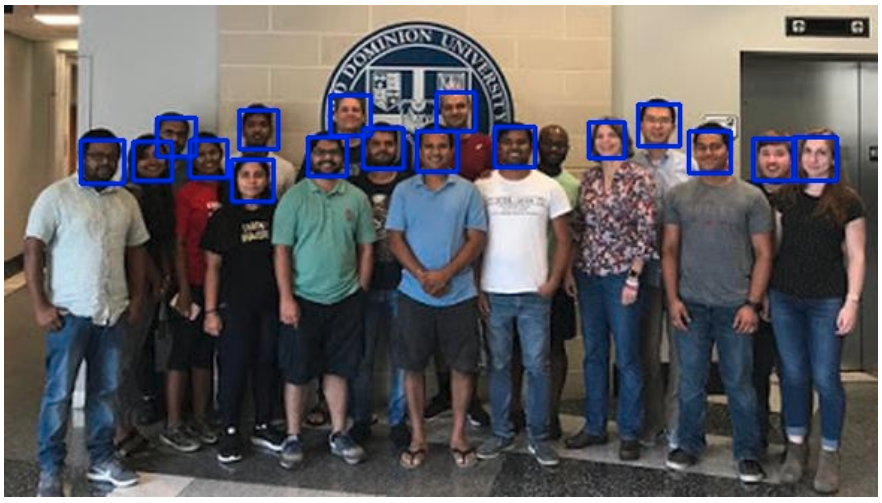


Figure 5: Iteration #05 of facial detection. Increased image size by 10%, 17 faces detected. (The processed image is greater than previous image, even though they appear the same size on the page.)

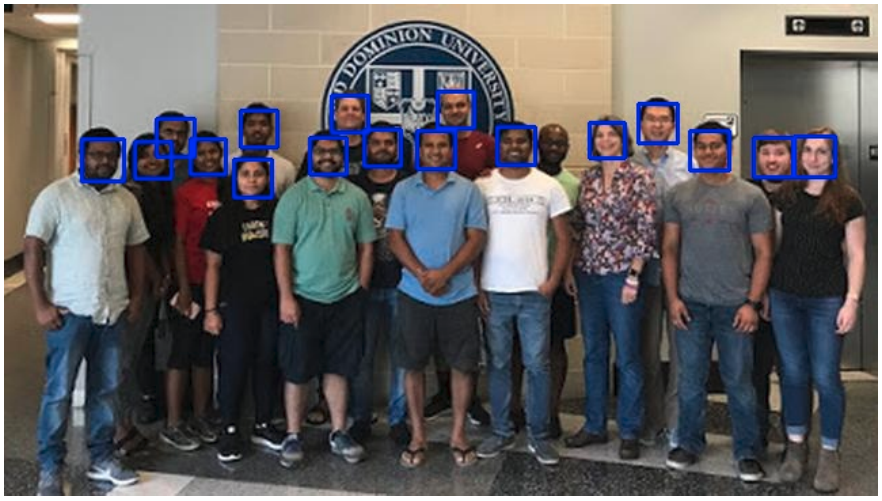


Figure 6: Iteration #06 of facial detection. Increased image size by 10%, 17 faces detected. An undetected face is still visible. (The processed image is greater than previous image, even though they appear the same size on the page.)

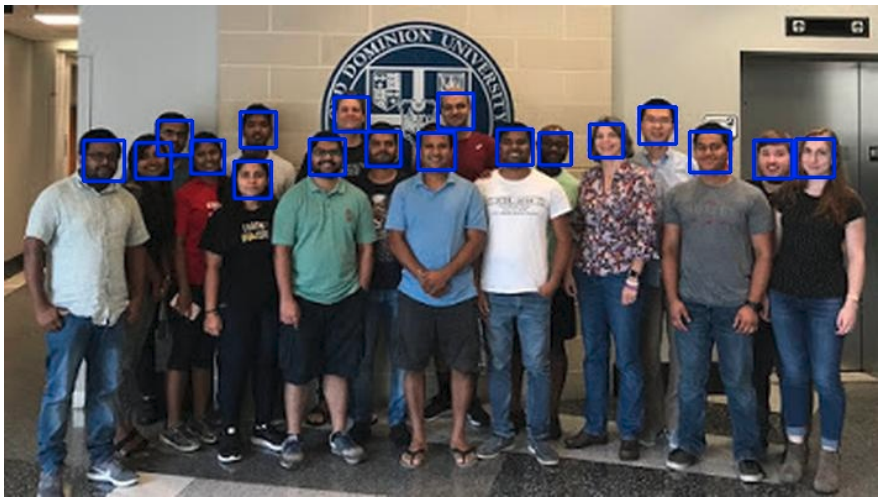


Figure 7: Iteration #07 of facial detection. Increased image size by 10%, 18 faces detected. All faces have been detected. (The processed image is greater than previous image, even though they appear the same size on the page.)

3.4 Accepting faces

After an acceptable number of faces have been detected, the user is queried about each detection as to whether or not the detected area represents a face. The mechanics are:

1. The original image is presented with the current candidate area outlined.
2. The user is asked whether or not to accept the area. A "y" means yes. A "n" means no.
3. If the user elects to accept the area:
   (a) Information is asked about the accepted area.
   (b) Information is written to `ffGlobals[''htmlFile'']`.
   (c) The area is outlined on the `ffGlobals[''markedUpFile'']` image file.
4. If the user elects to not accept the area, then the next area is presented.
5. If the user elects to end the program ("e") then the `ffGlobals[''htmlFile'']` is finalized and the program ends.
6. Otherwise, the next area is queried, and the process continues.

A few of these operations are displayed (see Figures 8, and 9). The process is repeated in the same manner for all detection areas.

(a) Queried area. (b) After accepting queried area.

Figure 8: Query and acceptance of detection area #01.

(a) Queried area. (b) After accepting queried area.

Figure 9: Query and acceptance of detection area #06.

4 Results

The Python script was tested on three different cases that seem to represent "normal" types of images. Each case is qualitatively evaluated at different stages along the way (see Table 3).

Table 3: Reality and detection confusion matrix. A qualitative assessment of what the detector finds in the image, and what is really there.

		Image	
		True	**False**
Detections	**True**	Faces that are detected and are in the image.	Faces that are detected, but **not** in the image.
	False	Faces that are **not** detected, but are in the image.	Non-faces that are **not** in the image. (The null hypothesis.)

4.1 A trivial case

A trivial case based on the image file:`27731313535-2312bd3924-w.jpg` [3]. A collection of reasonably sized faces, all facing the camera.

		Image	
		True	**False**
Detections	**True**		
	False	Initial image. No faces detected.	

		Image	
		True	**False**
Detections	**True**	Script was able to find all faces using default values.	
	False		Increasing the image size will cause one false face to be detected.

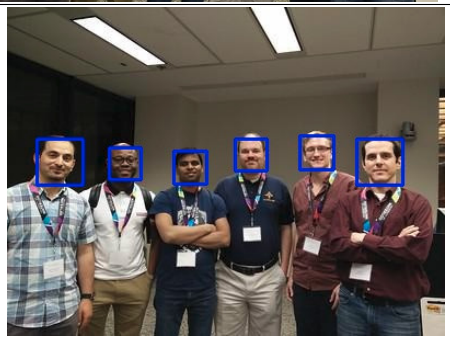


		Image	
		True	**False**
Detections	**True**	A quick acceptance of all found faces results in clickable faces.	None.
	False	None.	None.

4.2 An interesting case

An interesting case based on the image file:`ws-dl-group-1.jpeg` [1]. A collection of smaller sized faces, all facing the camera.

		Image	
		True	**False**
Detections	**True**		
	False	Initial image. No faces detected.	

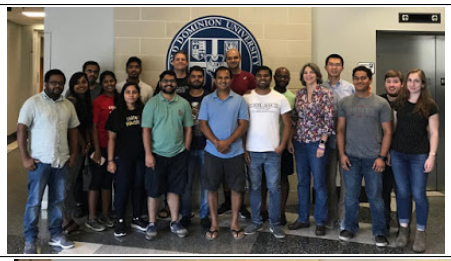


		Image	
		True	**False**
Detections	**True**	Script was unable to find all faces using default values. Successive increases in image size began to detect faces.	
	False		

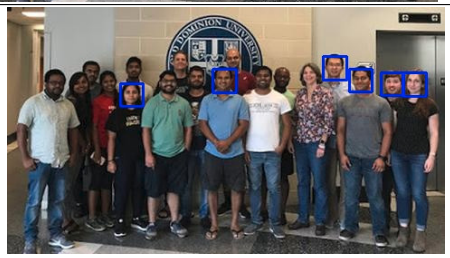


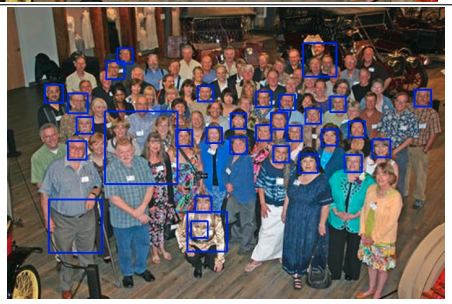
		Image	
		True	**False**
Detections	**True**	A quick acceptance of all found faces results in clickable faces.	None.
	False	None.	None.

4.3 A difficult case

A difficult case based on the image file:`homepage.jpg` [2]. A collection of much smaller sized faces, most facing the camera.

		Image	
		True	**False**
Detections	**True**		
	False	Initial image. No faces detected.	

		Image	
		True	**False**
Detections	**True**	Script was unable to find all faces using default values. Successive increases in image size began to detect faces.	
	False	Starting to see "non-faces" being detected.	



The software returned 121 detections, clearly more than the approximately 80 faces in the picture.

		Image	
		True	**False**
Detections	**True**	Each detection was "accepted" or "rejected" to cull the false detections.	Approximately 30% of the detections were false.
	False	The user kept increasing the size of the image until all visible faces were detected.	None.

5 Future work

The purpose of this exploration was to see how to do the "heavy lifting" of detecting faces in a "sea of faces" and then to demonstrate doing something with those detections. The base program could be expanded in these areas:

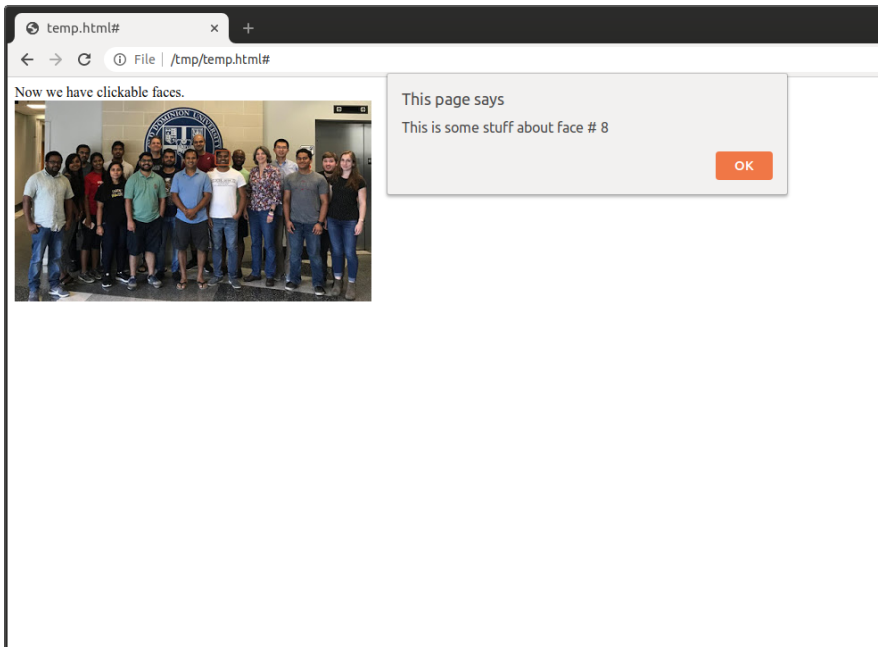
1. Replace the "onclick" URL in the `area` tag with a direct link.
2. Integrate the basic Python functionality into a larger GUI based application.
3. Remove the need for outside image viewers to see the intermediate images.

6 Conclusion

We started our exploration looking to reduce the number of man-hours it took to take a "sea of faces" and make it into something that the viewer could drill down into. We used a Haar based cascade detector in a Python script to detect "faces." Depending on the size of the "faces" the detector would return "true" or "false" faces. The script allowed us to evaluate each of these detections, and select the ones that we wanted to accept as "real"

faces. Finally the script created a web page where we could click on a face and find out something about the face (see Figure 10). The web page source can, and should be modified and then included into a final page so that users can find single face in a sea of faces, and be told something about the face.

Figure 10: A web page with clickable faces. The Python script created hooks for us to go from an unknown, and unknowable face in an image, to face #8, and from that to Sawood Alman. (The link from face #8 to Sawood Alman is outside the scope of this report.)



A Miscellaneous files

A collection of miscellaneous files mentioned in the report.

- 27731313535-2312bd3924-w.jpg – A sample image with a few faces.
- findFaces.py – A python script to find faces, and create the image map.
- haarcascade_frontalfacedefault.xml – The Haar frontal face cascade XML file.
- homepage.jpg – A sample image with lots of faces.
- ws-dl-group-1.jpeg – The Old Dominion Web Science Digital Library (WS-DL) group, circa 2020.

These files can be extracted from this report using something like the Adobe Acrobat Reader application. You may not be able to extract them using a Web browser.

B References

[1] Michael L. Nelson, *2020-01-13: Review of WS-DL's 2019*, `https://ws-dl.blogspot.com/2020/01/2020-01-13-review-of-ws-dls-2019_13.html`, 2020.

[2] Classreport Staff, *Lathrop High School, Fairbanks, AK USA*, `https://classreport.org/usa/ak/fairbanks/lhs/1970/`, 2020.

[3] WS-DL Staff, *WS-DL at ODU-CS*, `https://ws-dl.cs.odu.edu/`, 2020.