1-1-2022

Face Detection Using Web Images

Chris Zhao

FACE DETECTION USING WEB IMAGES

A Thesis

Presented for the

Master of Science

Degree

The University of Mississippi

by

CHRIS ZHAO

May 2022

ABSTRACT

The current facial recognition algorithms struggle with accuracy on real world cases. Haar cascade based algorithms are fast, but require fine tuning per image in order to achieve the best results. When tasked with images where there are multiple faces at different locations, the current algorithms seem to underreport the number of faces. This study attempts to produce a more accurate classifier through the use of taking the maximum result of multiple Haar cascade classifiers with differing parameters. To do this, a web image scraper was written to gather real world images from Google images and Flickr. These images were analyzed using the OpenCV library utilizing multiple Haar cascade classifiers and the maximum of these classifiers was taken. The result is a more accurate classifier, as most of the inaccuracies were due to undercounting, rather than overcounting.

LIST OF ABBREVIATIONS AND SYMBOLS

API Application Programming Interface

CNN Convolutional Neural Network

DNN Deep Neural Network

ACKNOWLEDGEMENTS

I would like to thank Dr. Yixin Chen for his help in this project.

TABLE OF CONTENTS

LIST OF FIGURES

I. LITERATURE REVIEW

Many of the current face detection algorithms are driven by the Viola-Jones object detection framework developed in 2001 [1]. This framework works by first selecting many Haar features that correspond to the properties of a face. These features are selected with the intent of identifying the front of the face, so they are similar to the eye and nose regions of the face [2] and [3]. After selecting these features, we compute an integral image that contains thousands of simple features in the image. This allows for features to be quickly calculated and utilized in the next step. Adaboost is then used to boost the performance of these simple features into one strong feature. To help computation time, a cascade of classifiers is used to minimize the amount of classification done on non face areas. From the cascade of classifiers, each stage decreases the false positive rate and the detection rate [4]. This results in a relatively quick and accurate algorithm for images containing frontal faces.

Other methods involve using a convolutional neural network (CNN) to develop a classifier that is better at detecting faces with tilt or occlusion. One method involves using the YOLO (You only look once) model, an object detection system that applies a single neural network to the image to give predictions on objects [5]. This method results in a much better accuracy on some test images, boasting up to 89.65% accuracy from their test set [6]. The test set included images where the Haar cascade classifier struggled, mainly images that contained changes in the head tilt, lighting conditions, and facial expressions. For non frontal face focused

images, CNN and deep neural network (DNN) based models perform noticeably better than traditional Haar cascade classifiers [7].

II. WEB IMAGE SCRAPER

Generally, face detection algorithms are trained and tested on “easy” images where there is only one face which is the focus in the image. This causes a large variance between the testing results and the real world implementation results. In order to gather images that are closer to real world examples, I decided to make a web image scraper. For the image scraper, I decided to begin using Google image search, as that seemed to be the easiest, as it could provide the largest variety of images to use. I initially began with an implementation using Selenium, as I was already familiar with using it for tasks like this. I quickly made a script that would open up a web browser instance to automatically search the keyword on Google images and download the first 100 images. However, it was not really efficient, as it was essentially just emulating a human user right clicking on every image and saving it. I then decided to look into doing this through their application programming interface (API), and found their custom search engine API. This gave me a much more elegant approach, as I could directly get a thousand images per query and then download them one by one. It also gave a bit more control, allowing me to only get images of a certain size or file type.

One issue with Google images was the maximum of one thousand results per query using the API, which would be not enough for a large dataset. To get one thousand results, I would need to query the API ten times, as each query only returned one hundred images at a time. This was further limited by a cap of one hundred queries per day, meaning gathering some datasets

would require a few days to finish. Another larger issue was that I would find a large amount of stock images returned, just due to the nature of Google images. An example of this was when I was testing the query “family vacation.” Instead of returning images from a family’s holiday vacation, Google images returned clipart or vector images that contained the text “Family Vacation.” These types of images do not provide any useful data for face detection. Because of this, I decided to try to apply the same web scraping techniques to public social media images.

I attempted Facebook first, as it is one of the most well known social media platforms, in order to get more photos uploaded by real people. Facebook was very difficult to get public images from. Their public API had been changed since the Cambridge Analytica scandal and it was essentially impossible to access public images without approval while using a Facebook business account. It was only possible to scrape images from pages that the user owned, or if the user directly approved the app on Facebook. Because of this, I switched to working on a Flickr implementation, as Flickr allowed for a lot more access through their public API.

Flickr had a few advantages compared to any of the other platforms I tried previously. First, the amount of images per query was increased to four thousand, allowing for a much larger dataset than the previous Google images implementation. Flickr’s API also has a less restrictive limit on queries, allowing for 3600 queries per hour. The amount of stock photos were also dramatically reduced, as Flickr was a site where users uploaded their own photos. Flickr’s API also provided much more robust parameters that utilize a photo’s metadata. This would allow me to query images based on the date or location taken, as seen in Figure 2 below.

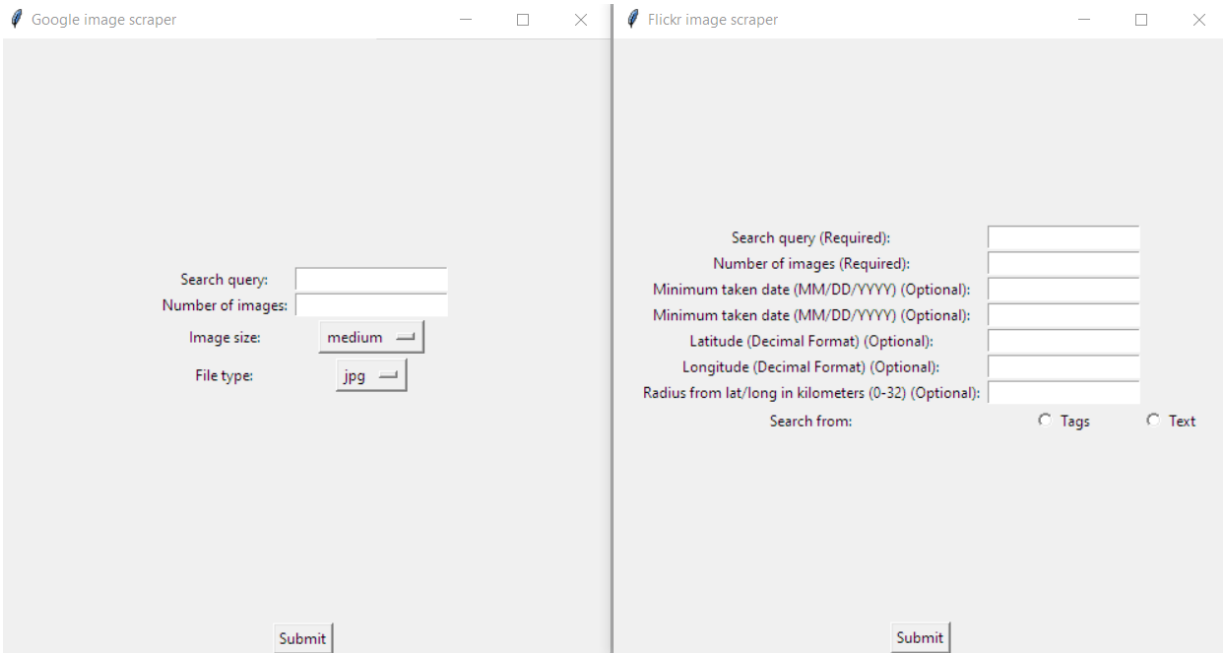

Figure 1. Comparison between Google image and Flickr API parameters

These parameters helped filter the images further, as including the image taken date helped filter out images that were taken on very old cameras, with extremely poor image quality. Unfortunately, Flickr is relatively limited in the resolution of the images through its API, so all the images gathered were of 500x500 resolution. This seemed to be mostly fine for images with below ten faces, but when there were a lot of faces in close proximity, the algorithm would struggle to differentiate them as multiple faces. The queries that I focused on primarily tended to include multiple people, for example, “family reunion” or “family vacation.” This allowed me to have a wide variety of subjects for face detection, and also have to detect multiple faces per image. From these queries, some of the images returned would still contain no faces, however these images were still useful as they usually had landscapes and backgrounds that were similar to the images containing faces. This allowed me to test if the face detection algorithms created false positives in images that contained no faces.

I also attempted to gather images from Instagram, a social media platform well known for its image sharing, however its API is very similar to Facebook. I was not able to gather any public images and decided to continue with the images that I gathered from Flickr.

III. FACE DETECTION

After I finished the development of the image scraper, I moved on to the face detection aspect. I began with the use of Python's OpenCV library, as I had already had previous experience with using it for another project. I found that the main method of face detection using OpenCV was with a Haar cascade classifier. I implemented and tested this classifier on the images gathered from the Flickr web scraper and found mixed results. For some images, it would be relatively accurate, however it would be widely off for others. An example of this can be shown in Figure 2 below.

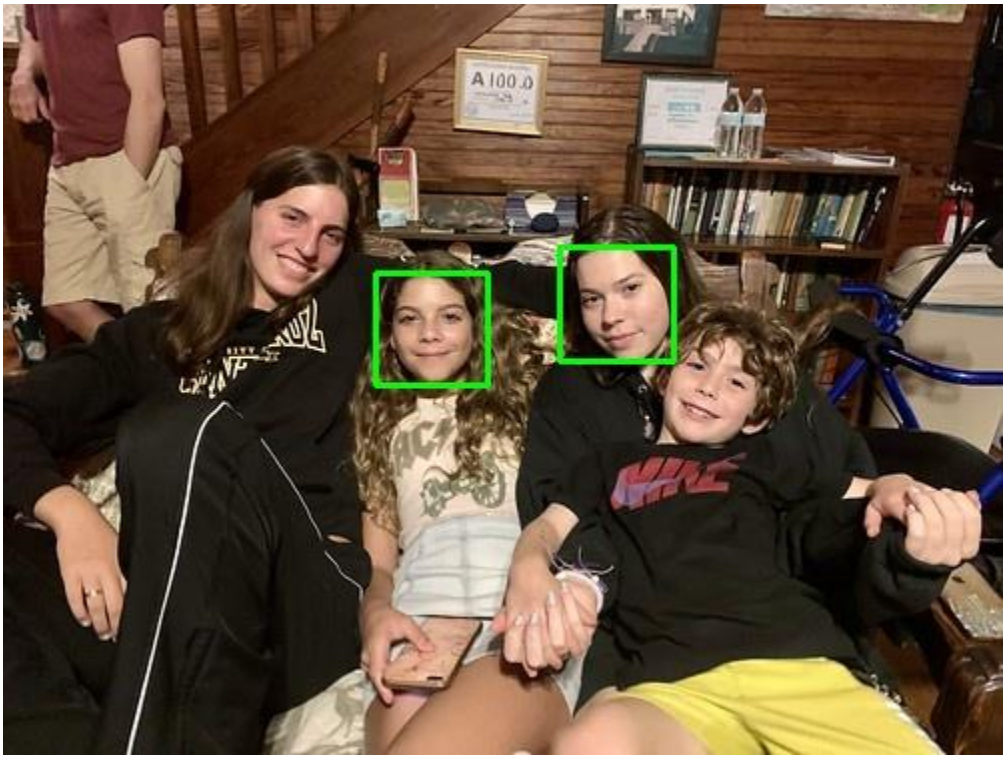

Figure 2. Inaccuracy of Haar classifier with poor parameters [8]

From further testing, it seemed like the parameters for the classifier were the cause of this inaccuracy. By carefully tweaking the parameters, I was able to get a much better result as shown below in Figure 3.

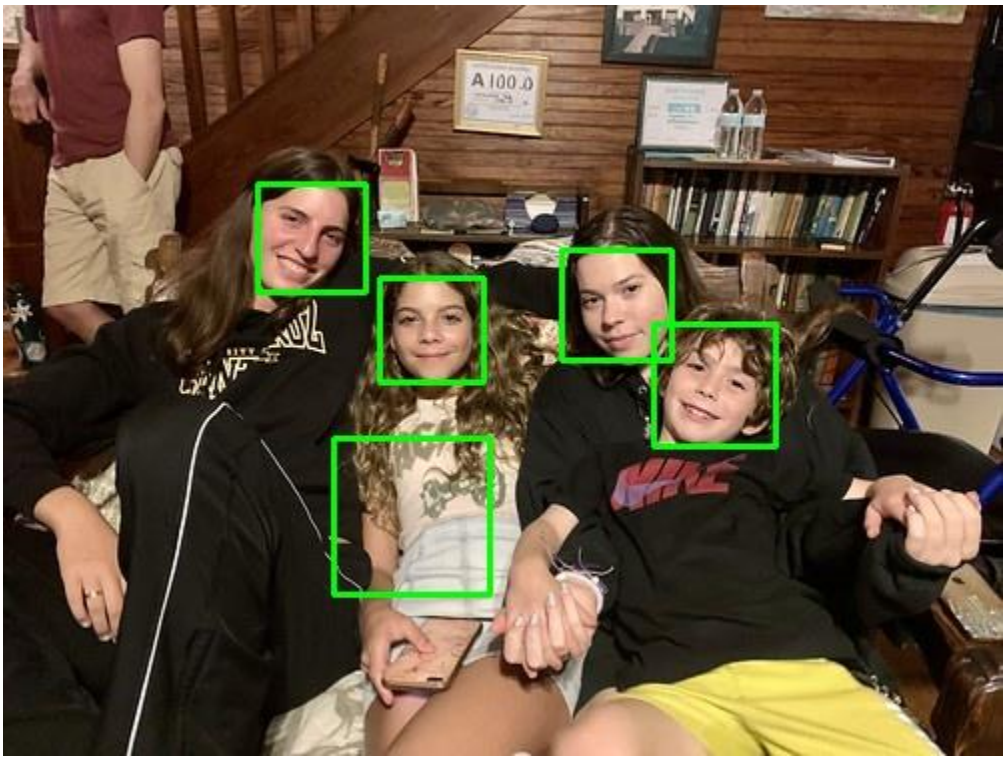

Figure 3. The Haar classifier with better parameters [8]

Although there was a false positive in the middle of the image, it produced a better result and could detect the other two faces compared to the original parameters.

The issue with choosing parameters was that each individual image had an optimal set of parameters to use, as each image had faces that were of differing distances from the camera. It is not feasible to manually determine the optimal parameters for a large dataset, so I looked to find patterns in improving the accuracy of the classifier. The most common result of poor parameters was a large amount of false negatives, where faces would not get recognized by the algorithm. Cases where the algorithm would produce false positives were rare, and they were rarely off by

more than a few faces. When running one set of parameters for a large dataset, many faces were not being detected properly for a large majority of the images.

To solve this issue, I decided to run the classifier multiple times, each with different parameters. In doing so, I would take the maximum value of the number of faces detected, as that generally led to the best result. This essentially changes the large amount of false negatives into a small amount of false positives. However, the amount of false positives is much lower, leading to an overall increase in the accuracy of the face detection algorithm. This method can be potentially run in parallel, as each set of parameters are processed independently of each other.

Another face detection algorithm that I attempted was with OpenCV's DNN module. The module was loaded with a pre-trained Caffe model for face detection. This method took more computational time, and produced roughly similar results as the Haar cascades classifier with one parameter set. One notable advantage of this method was that it fared slightly better on images where the face was not directly facing the camera. However, it struggled in some cases where it could not detect any faces on an image that contained several. Because of these reasons, I decided to continue attempting to improve the Haar cascade classifier.

III. RESULTS

I tested the maximum detection algorithm on a relatively small set of images that were gathered from the web scraper. These images had at least two front facing faces in each of them, all of varying ages and genders. Unfortunately, there was not a way to filter for subjects of a particular ethnicity, so the majority of images consisted of White individuals. This is perhaps due to the user base of Flickr being predominantly from North America.

On a set of ten images, we can see the differences between the number of faces detected from one set of parameters to another. This is represented in the chart below in Figure 4.

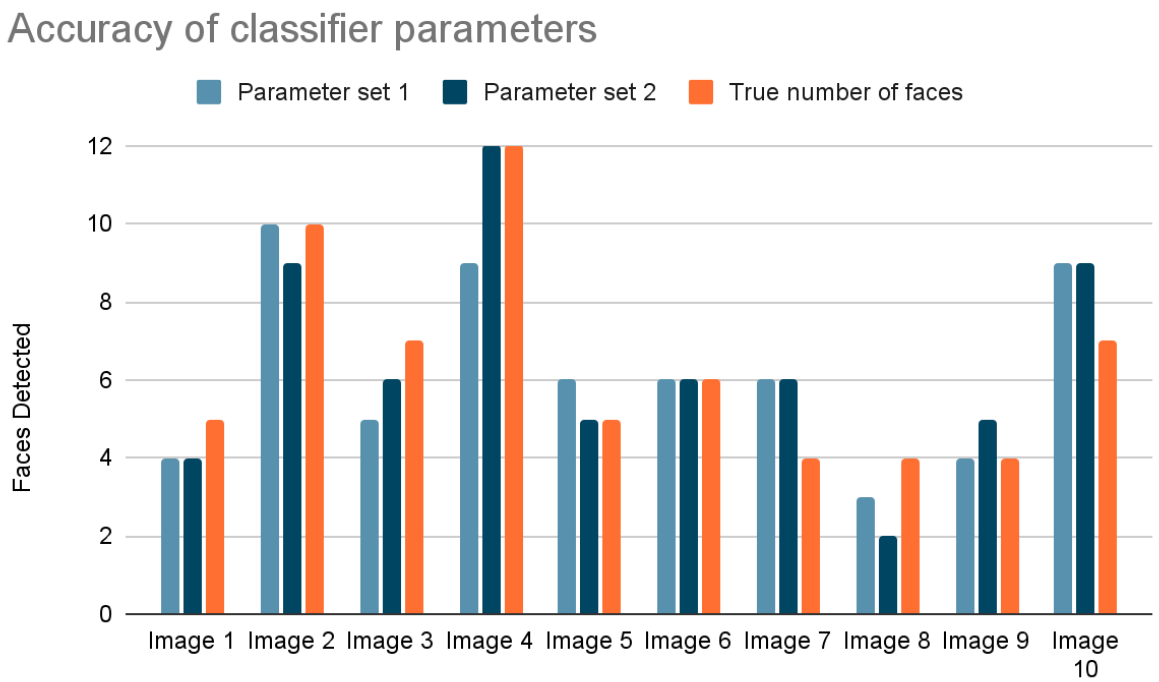

Figure 4. Comparison between two sets of parameters

Although the parameters are very similar, they cause different results in the amount of faces detected depending on the image. Both parameters have images where they are stronger than the other in determining an accurate count of faces in the image. By taking the maximum of the number of faces detected by both algorithms, we eliminate cases where one parameter set fails to detect large portions of faces. This introduces some false positives, as seen on images 5 and 9, however these cases are usually minor compared to the false negative rate. In some scenarios such as images 1, 3, and 8 both parameter sets undercounted the number of faces. This could potentially be fixed through the use of more than two parameter sets. The maximum of the two sets of parameters can be seen in the chart in Figure 5 below.

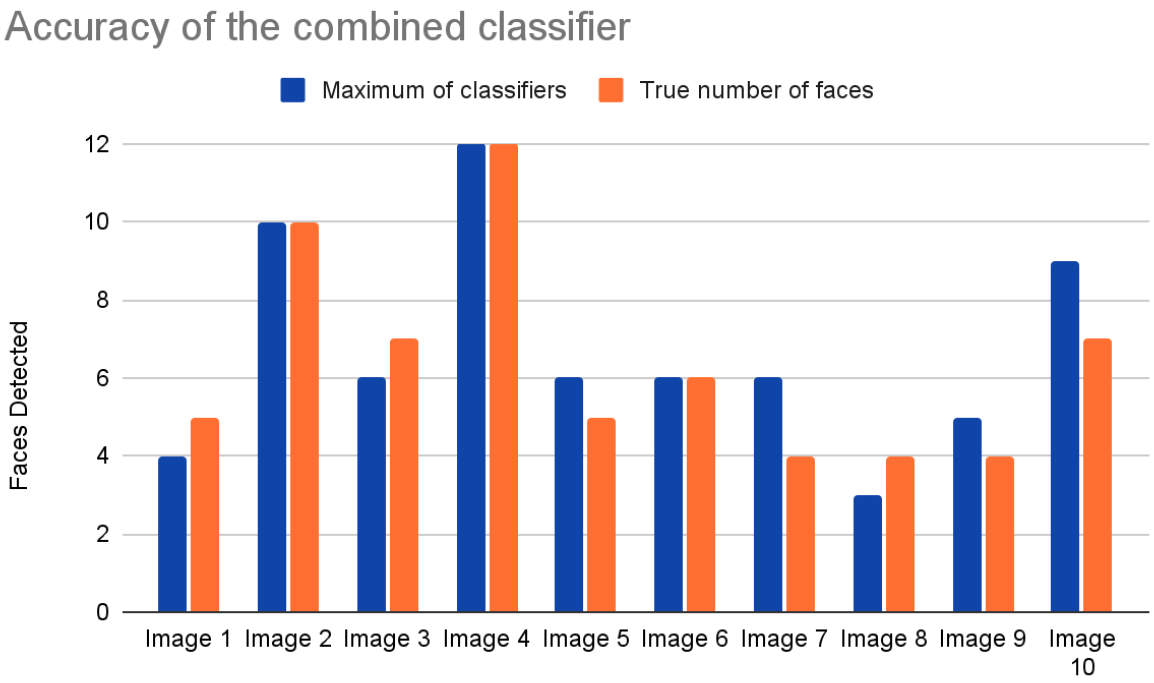

Figure 5. Using the maximum number of faces detected

From this chart, we see that the number of faces detected is generally slightly higher than the true number of faces. However, we can see that the largest difference between the reported value

and the true value decreases to just two. In comparison, on image 4, the second parameter set was off by not detecting four faces.

Applying this method to a larger set of images, the improvements are much more notable. As the number of images increases, more variations on the parameters are required to accurately classify all the images. When applied to fifty images, the maximum of the two classifiers with different parameter sets was not enough to accurately classify each image accurately. After increasing the parameter sets to five, the classifier was able to reach results similar to the ten image set.

Other methods such as taking the arithmetic mean do not work as well, as the Haar cascade classifier will err to the side of having a low false positive rate, making the average noticeably lower than the actual value. By taking the mean, the classifier consistently underreported the number of faces detected and was usually less accurate than the classifier with the most sensitive parameters. The same issue arises with the median and mode, where only a few parameters allow the classifier to detect previously undetected faces.

One potential issue in this method is the focus on the count of the number of faces. For instance, a classifier may claim that there are the correct number of faces in the image, but may be classifying the wrong parts of the image as a face. This could lead to a more inaccurate classifier that coincidentally gets the right number of faces. I believe this issue can only be solved through the usage of images with data on the area of the faces. From those areas, we can compare if our classifier's face areas are similar to the data. For this project, I was not able to test the accuracy of the face detection areas because I did not have images with facial location data.

IV. CONCLUSIONS AND FURTHER DEVELOPMENT

From this project, I was able to produce a method of increasing the accuracy of facial detection algorithms applied to a large image set. By taking advantage of the Haar cascade classifier's low false negative rate and very fast computational time, we can utilize multiple runs of the classifier with different parameters and take the maximum number of faces detected.

Further development could attempt to fix the issue of false positives in the combined classifier. In some cases, only a few parameters produced false positives, while the rest produced accurate results. Unfortunately, this is the result of taking the maximum number of faces counted. A potential fix for this would be to utilize some machine learning methods such as random forests or boosting could improve the false positive rate of the combined classifiers, similar to how specific features are selected for the classifier.

LIST OF REFERENCES

[1] Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. Ieee, 2001.

[2] Castrillón, Modesto, et al. "A comparison of face and facial feature detectors based on the Viola–Jones general object detection framework." *Machine Vision and Applications* 22.3 (2011): 481-494.

[3] Padilla, Rafael, C. F. F. Costa Filho, and M. G. F. Costa. "Evaluation of haar cascade classifiers designed for face detection." *World Academy of Science, Engineering and Technology* 64 (2012): 362-365.

[4] Wang, Yi-Qing. "An analysis of the Viola-Jones face detection algorithm." *Image Processing On Line* 4 (2014): 128-148.

[5] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[6] Ghenescu, Veta, et al. "Face detection and recognition based on general purpose dnn object detector." *2018 International Symposium on Electronics and Telecommunications (ISETC)*. IEEE, 2018.

[7] Prasad, Nitin, et al. "Frontal and non-frontal face detection using deep neural networks (DNN)." *International Journal of Research in Industrial Engineering* 10.1 (2021): 9-21.

[8] Figure 3,4: by Adam Baker, Flickr CC-BY-NC-2.0

https://www.flickr.com/photos/atbaker/51485301758/

APPENDIX

SAMPLE CODE

```
img = cv2.imread(image.path)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
faces1 = cascade.detectMultiScale(gray,scaleFactor = 1.05,minNeighbors=5,minSize=(10,10))
faces2 = cascade.detectMultiScale(gray,scaleFactor = 1.15,minNeighbors=4,minSize=(10,10))
faces3 = cascade.detectMultiScale(gray,scaleFactor = 1.25,minNeighbors=3,minSize=(10,10))
faces4 = cascade.detectMultiScale(gray,scaleFactor = 1.15,minNeighbors=2,minSize=(10,10))
faces5 = cascade.detectMultiScale(gray,scaleFactor = 1.10,minNeighbors=3,minSize=(10,10))
max_faces = max(len(faces1),len(faces2),len(faces3),len(faces4),len(faces5))
```

Figure 6. An example of using the maximum of five different parameters

VITA

May 2020 - B.S. in Computer Science, University of Mississippi