Nathan Cheshire

nvc29

Dr. Eric Hansen

CSE4633-01

Artificial Intelligence

11/20/20

**Facial Detection and Recognition in Python using the Facial Recognition Library with variable tolerance**

Facial recognition; a marvelous and surreal application of machine learning and linear algebra, optimized by neural networks, and utilized by copious security applications. There exist numerous ways to detect a face in a picture, one of the most primitive forms perhaps is an Eigenface approach. This approach works based on eigenvectors which are derived from a covariance matrix of the probability distribution over the high-dimensional vector space of the image of which we are trying to find faces in; simple right? For this project, however, instead of reinventing the wheel, I decided to use the *facial recognition* library for Python. This was installed via the pip command. This library relies on the *Dlib* C++ library that in term uses machine learning algorithms. So although this project is highly abstracted since I am using a python library, it is still utilizing the wonders and reaping the benefits of machine learning.

The *facial recognition* library in Python offered an extensive variety of tools to select from. The prominent features include simply finding faces in an image, manipulating faces, recognizing specific people by name (identifying) in images, and even real-time facial recognition on video data. The main abilities of this library I made use of include detecting any face in an image and training a neural network on multiple sets of faces for each person to then be fed images that were not trained on and attempt to find and recognize faces given a set tolerance in the range (0,1).

The results are semi-difficult to quantify and similarly error is reasonably difficult to quantify as well. I will my best, however, and attempt to provide some form of error analysis using MAE, Mean Absolute Error, in one cases. Other cases I will make educated and informed guesses at what went wrong, if anything, and how it could be improved with larger data sets, more GPU processing power as my machine does not have a great GPU to extensively train a neural network on, or enhanced algorithms.

To begin the project I knew I had to use python as it has become the prominent language for machine learning as Dr. Hansen pointed out in class. The topic of facial recognition always interested me and I thought it would be an interesting final project to tackle as opposed to digit or character recognition which is substantially easier to accomplish. The facial recognition library that exists in python includes an extensive amount of high-level functionality that provided plenty of support to complete this project.

Thankfully Python is a very well documented language, and I was able to swiftly read about the library’s functions and capabilities and write scripts to accomplish both parts of my project. Part 1 I decided would be to attempt to recognize specific faces that the neural network was trained on within a given dataset, a directory full of labeled faces. The second part, the semi-quantifiable part, would be to determine the MAE of found faces versus actual number of faces for multiple datasets. This library is excellent for detecting faces but sometimes a face is too blurry or impaired by something in front of it. Resultantly, the neural network misses a face in this instant. The Neural Network also cannot detect faces that are rotated more than ± 5 degrees on the z-axis in Euclidian space. With this in mind, I will conduct quantifiable runs on datasets that consists of only upright images as it will obviously not perform well on rotated/invalid images.

Following from the official python documentation on the facial recognition library, the prominent functions I will be utilizing are compare\_faces, face\_distance, face\_encodings, face\_locations, and load\_image­­­\_file. Starting with load\_image\_file, the function converts an image into a numpy array so that it may be processed and passed through the neural network. Face\_locations returns a list of tuples of found face locations and the corners that bound the face to a rectangle. Utilizing face\_encodings, the program will convert faces in an image to a facial encoding of 128 dimensions. Face\_distance allows a Euclidean distance for each comparison face in a list of face\_encodings to be found. This allows us to filter based on tolerance level. Lastly, compare\_faces compares a list of face encodings to a potential match; this is where the true facial recognition happens after training our neural network on pre-existing data.

To restate the parts I decided to implement to narrow down the broad topic of “facial\_recognition,” I chose to write a facial recognition program where you train the neural network on directories full of tagged individuals and then feed it a directory of unknown images. The neural network will then attempt to recognize faces from the unknown directory that it was previously trained on. I also implemented a feature to loop through multiple tolerance values. A higher tolerance means you will have more matches for known faces, but some of them may not be accurate. A user may also toggle the option to draw a rectangle around an unknown face to show the user that the program did indeed detect the presence of an unidentifiable face.

Secondly, in order to test how efficient this library was with recognizing faces based on their orientation, position, coloring, shadowing, and how much of the face is visible for example if the subject is wearing a mask (topical) or if they are wearing some sort of head ware that hides part of the face, I decided to attempt to calculate MAE. From the data, I produced a chart showing how many faces the program saw and how many faces were really there. I do believe many improvements could be made to this library for detecting obvious faces for low resolution images, images with poor lighting using edge detection, or even faces that are rotated more than 45 degrees along the z-axis, the axis that would run from head to foot.

Some notable and immediate conclusions from testing datasets of group photos are that this library has a difficult time detecting faces that are wearing masks. It almost always works if the majority of the face is facing forward and exposed. Only in extreme cases such as if the pixel dimensionality is less than 100x100 does it begin to suffer and not detect what a human would classify as a face. Below 50x50 pixels, it is not capable of recognizing a face whereas a human would recognize the possibility, likely hood, or high probability of it being a face.

Another major conclusion was that turned faces, faces without any or both eyes visible for example, were typically not recognized. In figure one, the library was unable to recognize any faces.



**fig. 1**

The group-photosdataset I calculated MAE on consisted of around 35 group photos. From the collected data, the MAE can be viewed in the graph below. The main point of performing these calculations, however, was to provide insight into how the underlying neural network recognizes faces. The prominent things I learned from conducting this first simple test was that faces should be greater than 100x100 pixels, should face relatively forward, and should be mostly uninhibited by any clothing or garments. The face should also be completely in frame and not cut off such as from the eyes and above or the nose and down.

**fig. 2**

As seen, the MAE value is 5.675. This indicates that on average for the chosen data set, the library missed approximately 6 faces that I, a relatively complex genetic algorithm (I would think), classified as detectable from each image. To be fair, however, this dataset included a large number of group photos, for some of which the algorithm detected no faces since their pixel dimensions were less than 50x50 pixels.

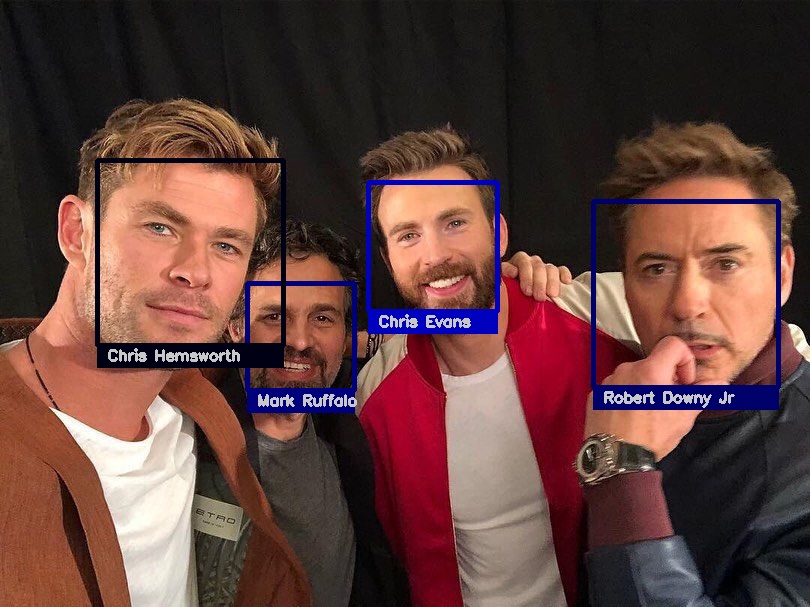
Moving on with confirmation that the library will recognize reasonable faces, I gathered a dataset of images of known people such as me, my father, and celebrities that meet the requirements to be identified by the library as a face. I then trained the neural network on most of the data and ran unknown images through the neural network and varied the tolerance level to obtain different results. I made sure that the images I chose for the dataset to train on, as well as the unknown images, consisted of well defined, mid to high pixel density, forward facing, and properly oriented faces so that the targeted goal for this part could be achieved.

Instructions and documentation exist in the static\_recog.py file on how to train the neural network on your own data and pass in your datasets for training and recognition. The results I obtained from hardly any training data proved surprising and satisfying. The pictures I trained off of are in lfw-now and include 6 pictures of me and 3 pictures of my father. The program was effectively able to identify the difference between me and my father and also did not falsely recognize me or my father in a couple bogus images I threw in.

**fig. 3 [Boy Scouts with my father 5 years apart]**



Following the usage of the facial detection script on me and my father, I decided it would be fun to try it out on four different celebrities that look somewhat similar enough that might trip a program. I chose a few actors from the Avengers cast: Chris Evans, Mark Ruffalo, Chris Hemswork, and Robert Downy Jr. The neural network was fed 2 images of each actor and the output for a tolerance level of 0.5 and 0.6 can be found in the project directory. The main difference is that for a tolerance of 0.6, all faces were identified but one face in a random group photo was marked as being Nathan Cheshire (me). Dropping to a tolerance of 0.5 resolved this error but also had the effect of not properly identifying Robert Downy Jr. in the group photo. This could be solved by simply adding more training data for myself and the actors.



Given that my computer does not have an amazing graphics card to train a neural network on, I did not even attempt to train on the dataset of over 5000 celebrities I found on the internet. This is an option, however, for people who would like to utilize these scripts and attempt facial recognition on their own unknown images. All datasets that I used and their outputs are still present in the root directory which I submitted alongside the paper and presentation.

An interesting thing to point out, since this library requires the face to be forward facing, mostly in view and mostly un-impaired, and not rotated, it will also detect faces on pictures/paintings in the background. This is due to the fact that there is no three dimensional analysis and the neural network merely looks at pixel data in two dimensional space and attempts to find a face and identify it as a face it was trained on. This can be seen in figure 4. Also note the difficulty in detecting faces wearing masks. The algorithm seems to suffer by about 50%. To reiterate a point made earlier, this library should support a way to recognize faces that are rotated and not almost perfectly oriented vertically.



**fig. 4**

In conclusion, although this final project was substantially shorter than my midterm project on making a pathfinding visualizer in combination with pac-man, I learned quite a bit about facial recognition in python and some of the challenges of it. I would have loved to try and take advantage of the fact that it struggles with detecting faces that have masks on to attempt to make some sort of a program to detect weather or not someone was wearing a mask.

Facial recognition is a very interesting application of neural networks and machine learning and the challenges such as utilizing depth sensing via LIDAR or infrared sensors or detecting low resolution faces prove to be strenuous yet intriguing. Choosing this as my project has made me appreciate how much work goes into the simple *Windows Hello* I use every day on my Lenovo. It has also made me appreciate the complex algorithms and machine learning utilized by Apple for their FaceID system. Hopefully, other will find these scripts easy and fun to use and will perform their own facial recognition tests on their favorite cousin, celebrity, or teacher.

Works Cited

Ageitgey. “Ageitgey/face\_recognition.” *GitHub*, github.com/ageitgey/face\_recognition.

“Face-Recognition.” *PyPI*, pypi.org/project/face-recognition/.

“Eigenface.” *Wikipedia*, Wikimedia Foundation, 1 Oct. 2020, en.wikipedia.org/wiki/Eigenface.

Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

“Convolutional Neural Network.” *Wikipedia*, Wikimedia Foundation, 1 Nov. 2020, en.wikipedia.org/wiki/Convolutional\_neural\_network.

“Machine Learning.” *Wikipedia*, Wikimedia Foundation, 30 Oct. 2020, en.wikipedia.org/wiki/Machine\_learning.