Using Facial Recognition to gather Social Media Intelligence

Jack Neilson

April 27, 2018

Contents

1 Abstract

Social Media Intelligence is an emergent sector in Open Source Intelligence that examines gathering information via publicly available social network profiles. It is viewed by many to be an extremely important aspect of information gathering, with several intelligence agencies including the UK Ministry of Defence and the US Federal Bureau of Investigation recently investing in tools to gather and analyse Social Media Intelligence (Antonius and Rich, 2013).

A relatively unexplored method of gathering social media intelligence is by using facial recognition. By allowing a search using a person's face, a completely unknown person of interest could be identified and several important pieces of personal information gained.

Keywords: SOCMINT; Facial Recognition; OSINT; Social Media; Artificial Intelligence

2 Acknowledgements

Special thanks are due to Dr. John Isaacs of the Robert Gordon University, without whom this project would not have been possible.

Thanks are also owed to Dr. Libor Spacek for use of the Essex Computer Vision database, Dr. Mislav Grgic for use of the SCFace database, and Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller for the use of the LFW database, respectively.

3 Literature Review

3.1 Background

3.1.1 SOCMINT

Social media intelligence (SOCMINT) is an emergent field in intelligence gathering where data is gathered from social media profiles. Massive amounts of data are added to social media services every day (Omand et al., 2012), much of it personal, making social media sites a potentially valuable resource when gathering information about groups or individuals (Ruiz, 2018). Social networks have also been used as a means of communication between persons of interest to the security services, making mining intelligence from their profiles a high priority (Omand et al., 2012)(Kilburn and Krieger, 2014).

After the 2011 riots in London that were organised in large part on social media, Her Majesty's Inspectorate of Constabulary stated that the police services were "insufficiently equipped" to effectively use SOCMINT in their response (Antonius and Rich, 2013) which suggests that social media intelligence sources may be woefully underutilised (Omand et al., 2012). This is not to say that the value of SOCMINT is not realised however, as many intelligence agencies are investing in tools to effectively gather and analyse SOCMINT (Antonius and Rich, 2013) or are performing case studies in to potential uses (Klontz and Jain, 2013).

While traditional human intelligence (HUMINT) focuses on building rapport and a foundation of trust in order to extract information from people of interest (Russano et al., 2014), users of social networking websites are much more likely to divulge personal information due to a misplaced senes of privacy (Livingstone, 2008). This makes SOCMINT attractive when attempting to gather data with little investment. The amount of data available to gather is vast in comparison to HUMINT sources (Omand et al., 2012), making mass collection and analysis viable (Unknown, 2013). The nature of SOCMINT makes it easier to analyse than HUMINT, which relies on "tells" and small social cues (Russano et al., 2014).

3.1.2 Uses of SOCMINT

As previously stated, SOCMINT has seen some emergent use particularly in the security services. The Greek Ministry of Defence has developed a framework to identify individuals fitting certain psychiatric profiles from their social media accounts to allow for early identification of potential insider threats (Kandias and Stavrou, 2015). By identifying factors that multiple intelligence agencies agree make a person more likely to pose an insider threat or negatively influence society (See appendix A), they were

able to map usage habits (intensity, content, popularity) to these factors to draw conclusions about clusters of users. So far, the research has been helpful in insider threat prevention, delinquent behaviour prediction and forensic analysis support.

3.1.3 Facial Recognition

Facial recognition is a much more mature area of research than SOCMINT with many examples of industry usage. Facebook uses facial recognition to automate "tagging" photos with the identity of the persons pictured (Becker and Ortiz, 2008), and large companies are now releasing datasets such as YouTube Faces (Cui et al., 2013) in an effort to advance the field.

This is not to say that facial recognition is not without controversy however, as many privacy advocates have pointed out that accurate face regonition could infringe on their right to privacy (Ruiz, 2018). David Wood and Lucas Introne have posed that accurate facial recognition could lead to increased levels of surveillance, with no way to "opt out" (Introna and Wood, 2002)(Bowyer, 2004).

3.1.4 Uses of Facial Recognition

Facial recognition has many practical applications that are already being realised. As noted previously, Facebook uses facial recognition when "tagging" photos. This is presumably done to allow advertisers to more effectively target individual users - for example, a person identified in a photo with a barbeque may receive adverts for propane gas.

Facial recognition is also enjoying a heavy amount of attention from the security services due to it's use in identifying persons of interest. Case studies have been performed using images released to the public to ascertain how effective facial recognition is when looking for a specific person. In particular, Joshua Klontz and Anil Jain performed a case study using the images of the "Boston Bombers" against a set of test data (Klontz and Jain, 2013). Their approach was successful in recognising one of the perpetrators from a picture taken from his social media account (See appendix B).

3.1.5 Constrained vs Unconstrained

While facial recognition software has come a long way, achieving accuracy rates of up to 99% on small, consistent data sets (Best-Rowden et al., 2014), it is still in it's infancy when it comes to identifying people in "unconstrained" images. Images taken in the wild may have large variations in pose, facial occlusion and ambient lighting. This makes it difficult to identify facial features or markers (such as iris distinace, nasal distance, blemishes) which in turn has a negative impact on accuracy rates (Klare et al., 2015).

When looking at applications of face recognition software with unconstrained datasets, matches are typically achieved when the test image has similar pose, facial occlusion and lighting as the sample image (See appendix B).

3.2 Prior Knowledge Attacks

3.2.1 Social Engineering

Social engineering in the context of information security refers to the ability of a person to gain access to information or control systems through a user or administrator, rather than through any technical oversight (Bakhshi, 2008). It is a popular technique against "hardened" targets as technical prevention measures have proven to be ineffective (Krombholz et al., 2015) making the human users and administrators the weakest link in the proverbial chain. In addition, it is difficult to train users in defenses against social engineering attacks. People believe they would not fall for such a trick despite research showing that humans perform poorly when attempting to detect deception (Krombholz et al., 2015)(Bakhshi, 2008).

Social engineering using e-mail as a medium is ubiquitous (Bakhshi, 2008). As a generel strategy, the person wishing to gather information will send one or more people an e-mail in hopes of a response (whether that be in the form of visiting a website, replying with their username or password, opening a file etc). Recent studies in realistic environments have had success rates of up to 23% (Bakhshi, 2008) - this is especially worrying given that even 1 repsondant could compromise an entire company.

Examples of social engineering are not hard to find. In his book *The Art of Deception* Kevin Mitnick describes a young Stanley Rifkin using social engineering to make the biggest bank heist of all time, stealing over 10,000,000 USD (Mitnick and Simon, 2011). Social engineering attacks are certainly not difficult to perform - recently, a 13 your old child used social engineering to breach the private e-mail of the then-chief of the CIA, John Brennen (Timm, 2015)

3.2.2 Spearphishing

Spearphish attacks are a subsection of social engineering attacks wherein an attacker sends a malicous e-mail to a user that has been crafted using information that would make it seem authentic (Caputo et al., 2014). For example, a normal social engineering attack using e-mail might send boilerplate messages to all personnel in a department to attempt to gain access to a network account. By contrast, a spearphish attack may use the names of specific people in the department, the location of the department, a spoofed e-mail header to make it seem as if the message came from within the department, and so on.

Social engineering attacks have proven to be effective at accessing sensitive information even when significant effort has been expended to secure it. A recent spearphishing campaign was conducted against 500 US military cadets - over 80% clicked the link in the e-mail (Caputo et al., 2014). Spearphish attacks are even more effective than "blind" phishing because of the additional information included in the e-mail which lulls the user in to a false sense of security (particularly when the information could be wrongfully considered "sensitive", for example including a manager's name and phone number).

Spearphish attacks in particular are difficult to mitigate against. They target the human element in information security making technological defenses insufficient (Caputo et al., 2014). Education on spearphishing is not adequate to mitigate the potential threat either, as shown by William Pelgrin's exercise. He sent 10,000 New York State employees a phishing e-mail, and had a success rate of 15% (that is, 15% of users clicked the link in the e-mail then attempted to enter their passwords). The experiment was repeated after four months, with some success shown as the experiment had a success rate of 8% (Parmar, 2012). It must be remembered however that even a single successful spearphish attack could lead to a breach of information security.

3.3 Intelligence Gathering

3.3.1 SOCMINT

Social media intelligence (SOCMINT) refers to information gathered from social media profiles hosted on social networks. It is an emergent field in open-source intelligence (OSINT), which relies on gathering information users divulge about themselves in the public domain. It is characterised by the massive amount of data available (Omand et al., 2012) and the difficulty of analysis (Bartlett and Reynolds, 2015). An overview of subjects relating to social media intelligence gathering follows below.

3.3.2 HUMINT

Human intelligence (HUMINT) pertains to the gathering of intelligence from individual human subjects. Information may be divulged non-consensually e.g. in the case of interrogation (Evans et al., 2010), or consensually in the case of clandestine information gathering (Musco, 2017).

Non-consensual information gathering via interview or interrogation has only recently become a subject of study for the general public (Russano et al., 2014).

Consensual information gathering sits in a much more grey area. Presenting yourself as somebody else may not be illegal depending on the circumstances, however it poses

several moral questions. Clandestine intelligence gathering is still an extremely effective strategy, particularly when attempting to gather sensitive information which may be more heavily protected e.g. airgapped, firewalled (Musco, 2017). This makes it attractive during wartime or times of civil unrest (Charters, 2018)(Gioe, 2017)

Using a human intelligence approaches when gathering information has several downsides. It is a high risk strategy, as should a person be found out the consequences can be severe (Charters, 2018). The potential reward of sensitive information may be deemed to not be worth the risk. It goes without saying that HUMINT does not scale particularly well - it is a useful tool when attempting to extract information from a single person or small group, but it is much less useful when gathering information about larger groups. It relies on trust being built and may be ineffective when attempting to gather information from targets trained in tradecraft (Charters, 2018)(Musco, 2017)(Gioe, 2017).

3.3.3 Individual vs Group Data

Much of the research done on social media intelligence has focused on group trends or finding subsets of a population (Antonius and Rich, 2013)(Omand et al., 2012). Comparatively, fairly little research has been done on identifying a single person of interest from a social network service.

A practical example of where SOCMINT has been used in the real world is the framework created by Kandias and Stavrou, in which features of a social media profile such as number of friends, "friend hops" standard deviation, psychiatric profiling and usage intensity are used to predict groups of interest that may become radicalised (Kandias and Stavrou, 2015). While this is certainly very useful, it focuses more on identifying groups by some common factor rather than identifying a single person of interest for further analysis.

3.3.4 Quantity of Information

Having limited information is not a problem when gathering social media intelligence. Rather, the opposite is true - 250,000,000 photos are added to Facebook every day (Omand et al., 2012). The vast amount of data is what makes searching for social media so difficult (and nigh on impossible in real time).

While the total amount of data added to social networking websites is extremely large, the information about a single user may be fairly small especially if the user has been trained to release as little information as possible (Kandias and Stavrou, 2015). This may make targeting single persons of interest less valuable, however should information "leakage" occur the potential payoff is high. It should be remembered that scanning

social media profiles incurs little to no risk, something that cannot be said of human intelligence.

3.3.5 Accessibility of Data

As discussed previously the amount of data pushed to social networking websites is massive. This does not mean that all social networking data is readily accessible. Users may set privacy settings to disallow unauthorised parties from viewing their profile, or particular parts of it. This can be mitigated by having the service allow access, however acquiring access as a party of one may prove difficult.

Several social networking services also impose rate limits on their interfaces (for example, Facebook imposes a limit of 100 API calls per user of an application as well as limits on CPU time used). These limits obviously make searching through the entire data set of profiles unfeasible without allowances made on the side of the social network.

3.3.6 Uses

Usage of social media intelligence has been limited in large part due to the difficulty in analysing such large amounts of data for small snippets of useful information (Omand et al., 2012). There are, however, some interesting case studies available.

In April 2013, Edward Snowden leaked a deck of slides used by the NSA to brief people on the "PRISM" program (Unknown, 2013). These slides detailed how the NSA uses mass surveillance with cooperation from companies including Google, Facebook, and Apple, to conduct surveillance against persons of interest. As a government agency enabled by the oversight committee, the NSA has the power to require large social networking services like the ones listed to allow them access to their data.

As mentioned previously, Kandias and Stavrou have developed a framework for using social media intelligence for predicting and mitigating insider threats while working at the Information Security and Critical Infrastructure Protection Lab at the University of Athens (Kandias and Stavrou, 2015). The framework focuses on analysing the psychology of the subjects using their social media posting habits. It shows that seemingly inconsequential data such as posting frequency may still be useful when analysing social media intelligence. An example graph to find the amount of "klout" a social media user may have in the Nereus framework is referenced in Appendix C.

3.3.7 Challenges and Constraints

There are several challenges to consider when collecting and analysing data for social networking services. First is the issue of privacy. Many people see social media profiles

in a similar setting as a meeting between friends, and as such post things which they perhaps would not say or show in public (Livingstone, 2008). The chilling effect of public scrutiny, or even direct surveillance, does not seem to put a damper on this.

There are some that feel that police surveillance of social networks is too pervasive (Mateescu et al., 2015), and encroaches on their right to privacy. Despite the effectiveness of social media intelligence gathering, it may be harmful to law enforcement's public image if it continues to utilise SOCMINT.

As far as the law is concerned, information posted on social networking sites is considered to be in the public domain. This makes it legal for law enforcement to gather social media intelligence, provided they do it the same way an unprivileged user would. Several social networks have clauses against mass data collection in their Terms of Service, however it is theorised that this is to prevent competition from other companies when selling said information to advertisers.

3.4 Facial Recognition

3.4.1 Current Applications

Police and military uptake of face recognition technology is widespread. For instance, police in the United Kingdom recently used cameras to scan people's faces at the "Download" music festival (Gallagher, 2005). These images were then compared with a database of custody images across Europe, to identify known criminals for the purposes of crime prevention and outstanding warrant execution.

Facebook also uses face recognition on photos uploaded to its service (Facebook, 2018*c*). By comparing face mappings to the known face mappings of yourself and immediate friends, it can suggest "tags" of who is in the image which presumably makes the information contained within the image more valuable to advertisers.

3.4.2 Unconstrained Facial Recognition

Unconstraned facial recognition is a subset of facial recognition which focuses on identifying faces in images with variations in pose, lighting, and facial occlusion. It is significantly more difficult than facial recongition which imposes constraints on these conditions.

Unconstrained facial recognition is typically used in situations where a subject is unaware or unwilling. As already mentioned, it was used in a case study by Klontz and Jain where they compared the images released by the FBI of Tamerlan and Dzhokhar Tsarnaev (the "Boston Bombers"). They found that by comparing the released images taken from security cameras to images on social media in an unconstrained setting,

they could positively identify one of the brothers (Klontz and Jain, 2013) (see Appendix B).

3.5 Existing Solutions

3.5.1 pipl

`pipl.com` is a popular web service that allows users to search a database of people by name, e-mail, social media username, phone number, or location. It is aimed at de-anonymising people that use pseudonyms or that do not publicly associate identifying information with their e-mail or social media username by cross-referencing information from several sources. For example, a person may associate their e-mail but not their phone number with a username on a social media service, and then associate the same username with their phone number but not their e-mail address on a different social media service. With this method pipl can effectively collate identifying data about a large sample of people and make it searchable, which is evidently useful for many business applications (particularly within advertisement, as it allows for targeted adverts to be sent to the same person over several social media networks). It's list of clients include large companies within the social media and personal data sectors such as Twitter, equifax and Experian (pipl, 2018).

3.5.2 192

`192.com` is a similar service to pipl, however it allows for a more advaced search with more terms to restrict or search on. As well as using social media profiles it uses publicly available information (OSINT) such as the electoral register to enhance its data set. It claims to have 750 million records, far less than pipl's over 3 billion (although the records kept by 192 are much more detailed). The main feature of 192 is that it allows users to search for details about entities other than people, such as schools or businesses.

3.5.3 Facebook

Facebook itself offers fairly primitive search functionality, allowing users to search for posts, people, photos, videos, pages, places, groups, and events. While this may seem to be fairly extensive, it should be noted that the intended use of the search function is to allow userse to make more connections. The searches users can perform will therefore be tightly restricted - for example, when searching for people the search may only show those people that are indirectly connected through other friends, known connections, or people in close proximity (Facebook, 2018*b*). The Graph API that Facebook exposes allows users to search outside of these restrictions, however as discussed previously it

is difficult for a party of one to use the Graph API for large-scale searches due to rate limits and Facebook's privacy settings (Facebook, 2018*a*).

Even with these limitations, Facebook is still a very useful service for enumerating identifiable information about a person of interest provided the user has some basic information such as the person of interest's full name.

3.5.4 Gaps in Functionality

The services listed above are extremely useful when attempting to gather more information about a person of interest. Given some basic identifying information such as username, given name, e-mail address etc. they can be used to effectively build a profile of searchable person. Where they fall short however is when a person of interest needs to be identified from a collection - the amount of data returned when searching for location or place of residence alone is far too large to sift through. Even Facebook's search falls short when trying to identify a single person from a potentially large collection, as the lack of customisable search restrictions means that the result of a search is unlikely to be able to be processed in a reasonable amount of time.

With the exception of Facebook, the services listed aggregate social media profiles in order to hold as much meaningful information about individuals as is feasible. This necessitates that some information must be omitted when collating from several social media networks. One of the pitfalls of these services is that several of the omitted items can be very useful when restricting searches - for example, a profile picture may prove useful for identification if a user has a reference image of a person of interest.

A potentially large gap in functionality is that lack of "soft" comparisons. The services above perform well when doing direct comparisons, such as having a specific name or a name matching a regular expression. They do not offer search functionality allowing for some margin of error such as a name matching a regular expression or having a Hebbian distance of 1, or having a face harvested from a profile picture be similar to a test image.

4 Methodology

4.1 Overview

The software implemented will be in the form of a Python command line program, which allows users to specify an input image and data set to check it against. Additional identifying information such as name, date of birth etc. should also be utilised if given. The data sets the software references must come from social networking websites (or in the case of this proof of concept, a data set which is an accurate representation).

As its output, the software must show the five most similar faces in the data set to allow for human verification. They should be presented with information that was gained from their social media profiles such as age, gender, name etc., as well as the similarity to the test image. In the case that the software is being ran headlessly, the results should be available in a file format which includes an image of the match and a representation of their social media profile.

4.2 Functional Requirements

4.2.1 Face Recognition

The software must implement a method of facial recognition, to rank a set of faces by similarity to a given test image. The test face must be extracted from the input image. The top 5 matches should be outputted, along with relevant identifying information.

4.2.2 Social Media Integration

The software must have the ability to use social networking websites as it's data set when comparing a test face. It would be desirable for the software to have the capability to use other identifying information alongside the test image to narrow the search space. Due to limited time and resources, test accounts may be used as a proof of concept.

4.2.3 Headless

The software should be able to function without a GUI, since the requirement of being able to use extremely large data sets may require deployment to a large server cluster or to a cloud.

4.2.4 Multithreaded

Where applicable, the software should make full use of parallelisation to increase its speed and resource utilisation. This is particularly important when working with large data sets.

4.3 Non-functional Requirements

4.3.1 Accuracy

The software should have a *relatively* high success rate when identifying a person from a social media collection. Note that typical facial reocgnition performance in unconstrained environments achieve a maximum success rate of 30%.

4.3.2 Usefulness

The information gained by using the software should help when crafting prior-knowledge attacks, increasing their effectiveness.

4.3.3 Cross-Platform

The software must be cross-platform to allow for easy deployment to servers or a cloud.

4.3.4 Imprecision Tolerance

The software should attempt a best guess when using an input image with less than ideal pose, facial occlusion or lighting. Inferior results when given a test image like this are acceptable to a degree.

4.3.5 Usability

The software is being developed for use by information security professionals in the form of a command line tool. This is ubiquitous in the industry, so no more than a cursory glance over the usage information should be needed to begin using the tool.

4.3.6 Facial Recognition Libraries

Due to time constrants, reimplementing facial recognition capabilities is not feasible. Therefore, the software should make use of already existing facial recognition libraries, in particular dlib or OpenCV.

4.3.7 Language

The software should be written in Python to allow for easy use of face recognition programs and rapid prototyping.

4.3.8 Time Spent

The software must be able to complete the above task in the same time or less than a human.

4.3.9 Documentation

The software should have at a minimum a man page and usage information. More documentation would be beneficial.

4.3.10 Testing

The software will be tested using a comination of publicly available face recognition datasets including Labeled Faces in the Wild, the Essex Facial Recognition database and the SCFace database.

5 Implementation

5.1 Face Recognition

The application has been developed using the face recognition module created by A. Geitgey:

`https://github.com/ageitgey/face\_recognition/`.

It is an easy to use facial recognition library build on top of dlib. It exposes a python API, making it ideal for this project. It also boasts an accuracy rate of up to 99.38% on the LFW data set.

Alternate libraries (such as dlib) that were investigated were found to either have a much higher barrier to entry, or a much lower accuracy.

5.2 Test Server

5.2.1 nginx

To test the application, nginx is used to serve static content as well as as a reverse proxy. Nginx was chosen over Apache or another native Python webserver due to its capability at handling massive concurrent requests, its flexibility, and its speed at server static content. Since the application makes several connections simultaneusly and requests large amounts of images, nginx was ideal.

5.2.2 cherrypy

To serve a dynamic description of where profiles and their respective images are located, a cherrypy server was implemented which expands a glob from the test directory (which contains the test data sets) then returns a json representation of them. As noted earlier, nginx acts as a reverse proxy to this service

5.3 Portable Module

The application has been developed in a modular fashion. To this end, a run file has been provided alongside a python module. The python module includes the logic for retrieving profiles from a web service and setting up multiple threads to compare face mappings, whereas the run file comes with some pre-processing and default arguments. Ideally, the module should be re-usable for other web services.

5.4 Threading

The application has been developed to make use of multiple threads. The main thread in *facegather.py* controls these threads and their results using locks, thread-safe queues, semaphores, and the thread.join() method.

The producer / consumer consumer model was useful when developing the application. In this context, the producers retrieve profiles from the social networking service and the consumers compare the retrieved profiles to the test data given and store the result.

5.5 Command Line Interactivity

Rather than use a graphical user interface (GUI), the application makes use of command line arguments (see Appendix D). There are several reasons for this; it cuts down on needless computational cost by eliminating the overhead of a GUI, it allows the OS the application is running on to be "headless" (i.e. run with only a command line or remote connection for user interactivity) again cutting down on overhead, and it allows for easier integration in to scripts (for example, appending logs to a log file using the POSIX >> operator).

One potential issue when using an application without a GUI is presenting the results. The application persists its results in a JSON format for use by other applications or for direct consumption by the end user.

5.6 Social Network Interactivity

Since this project has been done on a fairly small scale in a short time frame, it was not possible to gain access to a large social media platform to perform testing (ignoring the obvious ethical issues with such an approach). Instead, the test environment has been set up in such a way as to mirror the way in which a social media API may be presented - with an array of profiles, each one having a link to some related resources.

5.7 Challenges

5.7.1 Loading Images from Memory

Since the application retrieves images remotely over the internet, it makes sense to manipulate the images directly from memory. This avoids filesystem reads and writes, which are several magnitudes slower than memory reads and writes. The time save is massive, especially when repeated over thousands of images.

Unfortunately the face recognition library does not support loading images from memory, instead providing a method (load_image_file(uri)) to load images from the local file system. Upon inspection this method loads an image file from disk in to a Python Imaging Library (PIL) Image object, then casts it to a numpy array to allow for encoding operations. To mirror this method for use with images loaded to memory, the application uses the PIL Image.open() method on the raw response stream from the social network web server to create an Image object, then casts it to a numpy array. This achieves the same result as loading the image from the local file system significantly more efficiently.

```
# Load an image in to memory from local or remote sources
def load_images(uri, remote=True):
    if remote:
        print('Retrieving image from ' + uri)
        resp = requests.get(uri, stream=True)
        resp.raw.decode_content = True
        img = Image.open(resp.raw)
        return fr.face_encodings(numpy.array(img))
    else:
        print('Retrieving image from ' + uri)
        img_array = fr.load_image_file(uri)
    return fr.face_encodings(img_array)
```

Above: A function with an optional keyword parameter to load an image from either a local or remote source.

5.7.2 Maximum Loaded Images

A potential problem when using the producer / consumer model is that producers may produce too much data, filling all available memory and leading to a crash. To solve this a maximum loaded images parameter can be supplied when running the application, which will cap the number of face encodings loaded into memory at that number. This has been implemented using a semaphore initialised with the maximum number of face encodings to be loaded. Each producer thread calls acquire() before loading a set of encodings in to memory, and each consumer thread calls release() when the encoding has been compared and the memory can be freed.

```
# Producer thread
     self.waiting_counter.acquire()
          if profile_queue.empty():
              return
          profile = profile_queue.get()

# Consumer thread
```

```
            img = img_queue.get()
            ---
            Comparison happens here
            ---
        self.counter.release()
```

Above: The producer thread calls acquire() on the semaphore before loading mappings in to memory, and the consumer thread calls release() on the semaphore after the memory used for mappings has been freed.

5.7.3 Multiple Threads for Producing and Consuming

As it stands, the application has been developed with mulitple threads in mind. The user can define how many threads should be used for downloading images and producing face mappings via the –cthreads argument, and how many threads should be used to compare face mappings via the –pthreads argument. This allows for much higher processor utilisation than a single threaded application, as all CPU cores can be utilised.

While the application processes images orders of magnitude faster than it would had it been single-threaded, it may still be too inefficient for use on particularly large data sets. It is also difficult to scale due to its singleton nature. In the future it may be worth refactoring to make use of a discrete GPU, or to make use of specialised libraries for big data processing such as MapReduce.

5.7.4 Thread Object Access Control

To control object access between threads two Queue objects are used. One contains a JSON representation of all of the profiles in the search spaces and is only accessed by the producer threads, and the other is used by producer threads to send face mappings of profiles to consumer threads. These objects are guaranteed to be threadsafe, meaning that both the put() and get() operations are atomic. To control access to the result object that multiple consumer threads may try to access, a Lock() is passed in the consumer constructor. Before a consumer accesses the result object it must first call lock.acquire() which will block if another thread has acquired the lock first. After a consumer has finished writing to the result object it releases the lock by calling lock.release(), allowing other consumer threads access.

```
            self.result_lock.acquire()
                self.result.add([distance, img[1]])
                self.result_lock.release()
```

Above: The result object is not thread-safe, so must have a lock around it.

5.7.5 Result Retrieval

Having a "top 5" result as was initially planned would require many comparison and list traversal operations. It would also make having multiple consumers pointless, as only one thread would be able to do comparisons at a time. Rather than having the top 5 matches, the result returned is now all faces that match the test face within a given threshold (by default, face distance < 0.6).

5.7.6 Command Line Arguments

The application can take many command line arguments to customise its use. Having the user fill out and remember every one of these arguments would make the application unintuituve and difficult to use. Therefore, the decision was made to make heavy use of optional keyword arguments both when running the program (e.g. –cthreads=1) and in functions within the application. This allows the user to specify some, all, or none of the optional parameters and still have the application function correctly.

```
def search(test_face_location,
           uri,
           no_producer_threads=None,
           no_consumer_threads=None,
           max_loaded=None,
           threshold=None,
           name=None):
```

Above: The search function uses several keyword arguments with default arguments, then checks and loads with defaults if not supplied before continuing.

5.7.7 Inter-Thread Communication

The only inter-thread communication required by the application is for the consumer threads to be signalled when there are no more face mappings to compare. To this end, the main thread which spawns both the producer and consumer threads blocks by calling producer.join() on all producer threads. One all producer threads have returned, the main thread pushes a sentinel object on to the end of the face mapping queue. It then blocks by calling consumer.join() on all consumer threads. Once a consumer thread pops the sentinel object, it adds another sentinel to the end of the queue (for further consumer threads to consume) then returns. This guarantees that there is no unconsumed data, and that the consumer threads do not terminate prematurely.

```
# Main thread
for processor in processor_list:
    processor.join()
img_queue.put(sentinels.NOTHING)

for recogniser in recogniser_list:
    recogniser.join()

# Consumer thread
if img == sentinels.NOTHING:
    img_queue.put(sentinels.NOTHING)
    return
```

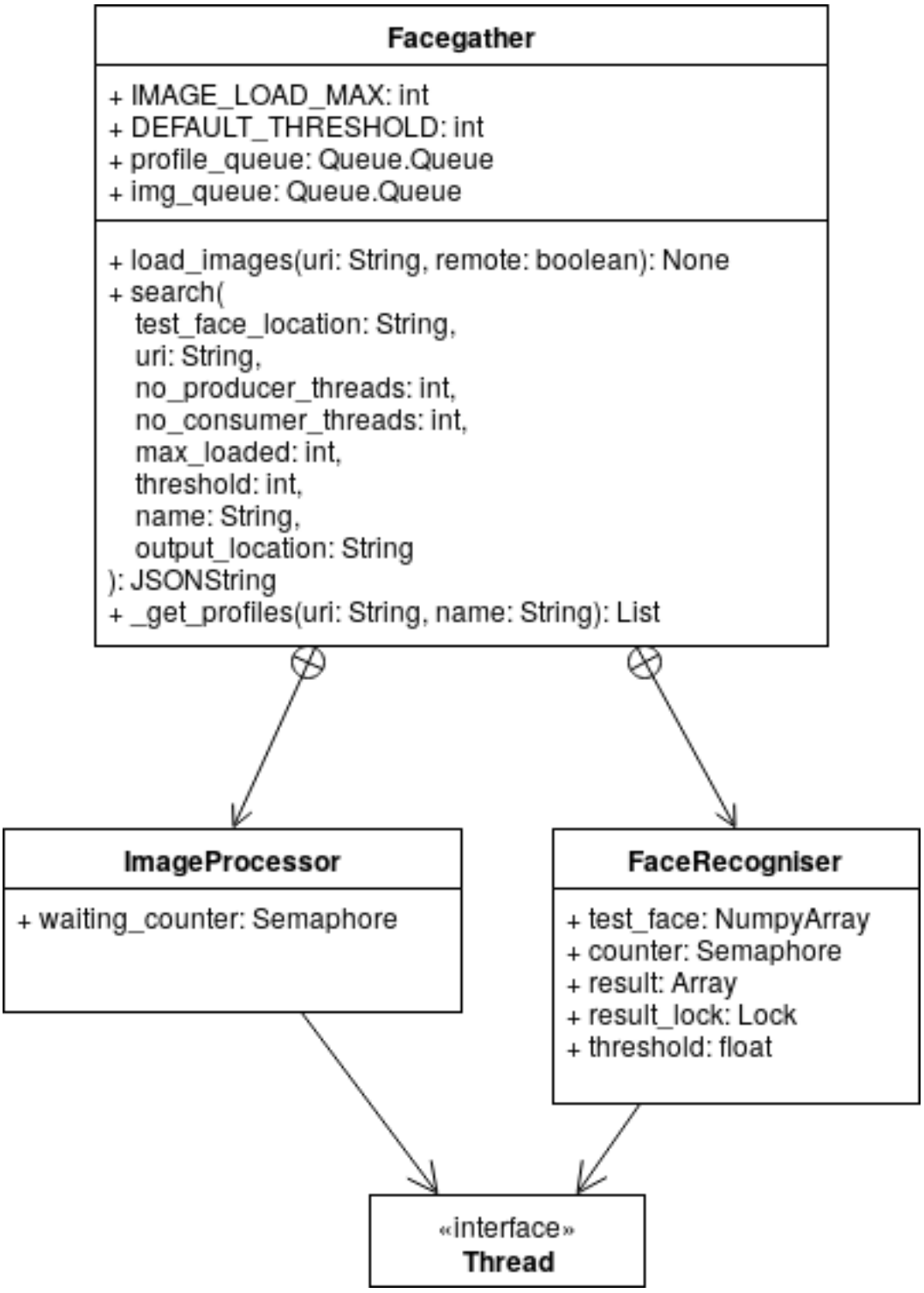
Above: The main thread blocks until all producer threads are finished, then pushes a sentinel object to the end of the queue, then blocks until all consumer threads are finished.

5.7.8 Loading Greyscale Images

The import function for loading images from memory described above failed when applied to greyscale images that had been converted from the JPG to the PNG format. All JPG images had been stored as RGB images even when they contained no colour, so when they were converted the compression algorithm saved space by converting them to single-channel images. To get around this problem, a special case was added to convert images in this format to RGB before attempting to generate facial mappings from them.

```
    except:
        # Catch line in case PNG file is encoded as greyscale
        return fr.face_encodings(numpy.array(img.convert('RGB')))
```

5.7.9 UML Diagram



6 Testing and Results

6.1 Aims

The aim of the application is to allow a user to query a dataset and find the social media profile of a person in a test image. The data sets used for testing are varied so as to allow for testing across a trained data set, testing across an untrained data set, and testing across an abnormal data set (in this case, the IR images taken from the SCFace database). The testing should reflect this, and as such should provide insight into the following areas:

- Even when comparing over multiple data sets that have been generated using different methods, a reasonable accuracy should still be maintained.
- Images that have been compressed with different algorithms should still match, given that they are of the same person.
- Success rates across abnormal data sets - for example, the SCFace IR images - should be tested to gather insight in to how effectively or otherwise these images could be used in facial recognition when gathering social media intelligence.
- The restrictions on search terms should work as expected, and exclude profiles which do not meet the search criteria.
- Testing done across data sets which have not been used in training should have a reasonable accuracy.

6.2 Testing Strategy

6.2.1 Comparing Images from Different Data Sets

To compare images from different data sets five images were selected from the total pool of images. One was taken from each of the seperate categories in the Essex data set, and one was taken from the LFW data set. The number of correct matches was then compared against the number of expected correct matches, and the number of false positives.

6.2.2 Comparing Images with Different Compression Algorithms

A data set that uses a different compression algorithm than JPEG could not be located, so instead a new data set was generated from the LFW data set by compressing the images using the Portable Network Graphic (PNG) algorithm. A comparison was then

made between the accuracy of an image compared with the same compression algorithm, and an image compared using two different compression algorithms.

6.2.3 Comparing Images Taken in the Infra-Red Spectrum

Images taken from the SCFace data set were compared to the IR images contained within the database, and the accuracy rate noted.

6.2.4 Restricting by Identifiable Information

A comparison was done in the accuracy and speed of searches on the LFW data set when restricting the search by first and full name.

6.3 Expected Results

6.3.1 Comparing Images from Different Data Sets

The program should maintain a high accuracy rate even across data sets it has not been trained on.

6.3.2 Comparing Images with Different Compression Algorithms

The PNG compression algorithm that was used is a lossless algorithm. No detail should be lost when compressing the data set, and so the expected accuracy when comparing photos with different compression algorithms is the same.

6.3.3 Comparing Images Taken in the Infra-Red Spectrum

Ideally, photos taken in infra-red should be matched at a similar accuracy as those taken in RGB. This may not be the case due to difficulty identifying facial descriptors.

6.3.4 Restricting by Identifiable Information

Searches against a restricted data set give less opportunities for false positives, and increase the ratio of matching images to non-matching images. Therefore, it is expected that the accuracy and speed of the search should be superior to an unrestricted search.

6.4 Comparing Images from Different Data Sets

Test Number	Accuracy	False Positives	Time Elapsed (Hours)
1	1.0	0	2:06:59
2	1.0	0	2:07:27
3	0.59	13	2:07:50
4	1.0	0	2:06:18
5	1.0	0	2:08:02
Avg	0.92	2.6	2:07:19

6.4.1 Test 1

Test image: essex_cswww/faces94/male/cjsake/cjsake.1.jpg

Image	Distance	False Positive
cjsake.1.jpg	0	No
cjsake.2.jpg	0.12415972564754801	No
cjsake.3.jpg	0.1202683265955043	No
cjsake.4.jpg	0.12056058735503877	No
cjsake.5.jpg	0.15399061544936016	No
cjsake.6.jpg	0.15170091461313734	No
cjsake.7.jpg	0.15761164790252086	No
cjsake.8.jpg	0.18675358283846008	No
cjsake.9.jpg	0.22679627515383244	No
cjsake.10.jpg	0.16498243163116574	No
cjsake.11.jpg	0.2083636000224947	No
cjsake.12.jpg	0.19579060568070747	No
cjsake.13.jpg	0.21036201522486805	No
cjsake.14.jpg	0.16785907543854067	No
cjsake.15.jpg	0.2078185538842606	No
cjsake.16.jpg	0.19772310008051744	No
cjsake.17.jpg	0.18166327218341696	No
cjsake.18.jpg	0.20414891693929105	No
cjsake.19.jpg	0.19693488266700446	No
cjsake.20.jpg	0.1671873443722762	No

6.4.2 Test 2

Test image: essex_cswww/faces95/adhast/adhast.1.jpg

Image	Distance	False Positive
adhast.1.jpg	0	No
adhast.2.jpg	0.2312083988155676	No
adhast.3.jpg	0.25972150326024057	No
adhast.4.jpg	0.21590055493708946	No
adhast.5.jpg	0.24049261224827648	No
adhast.6.jpg	0.23465681660325552	No
adhast.7.jpg	0.24771615139343836	No
adhast.8.jpg	0.2569527085240263	No
adhast.9.jpg	0.2835389426928616	No
adhast.10.jpg	0.2774791080863853	No
adhast.11.jpg	0.2532400849084287	No
adhast.12.jpg	0.28278790526661146	No
adhast.13.jpg	0.2937914096756732	No
adhast.14.jpg	0.2970183349284969	No
adhast.15.jpg	0.29927374807082874	No
adhast.16.jpg	0.2783214797648933	No
adhast.17.jpg	0.2736083238480712	No
adhast.18.jpg	0.2851054856115722	No
adhast.19.jpg	0.2891104254644535	No
adhast.20.jpg	0.2816904095878937	No

6.4.3 Test 3

Test image: essex_cswww/faces96/cjhewi/cjhewi.1.jpg

Image	Distance	False Positive
cjewi.1.jpg	0	No
cjewi.2.jpg	0.19289455983673667	No
cjewi.3.jpg	0.24900325719699287	No
cjewi.4.jpg	0.3291783082081157	No
cjewi.5.jpg	0.31046947257682805	No
cjewi.6.jpg	0.375778847049278	No
cjewi.7.jpg	0.49121008818147793	No
cjewi.8.jpg	0.24928926125340656	No
cjewi.9.jpg	0.2805170839075122	No
cjewi.10.jpg	0.2917877509617489	No
cjewi.12.jpg	0.33383609910398243	No
cjewi.13.jpg	0.3107498407520015	No
cjewi.14.jpg	0.3226220896718056	No
cjewi.15.jpg	0.3315345047726276	No
cjewi.16.jpg	0.29075881781505053	No
cjewi.17.jpg	0.3323513335348239	No
cjewi.18.jpg	0.3216638643477499	No
cjewi.19.jpg	0.310250334339258	No
cjewi.20.jpg	0.338702834726292	No
namull.1.jpg	0.48963088468953936	Yes
namull.6.jpg	0.4842381188636226	Yes
namull.7.jpg	0.4662936114262804	Yes
namull.8.jpg	0.4836512854747962	Yes
namull.9.jpg	0.4696436628368218	Yes
namull.10.jpg	0.4940518896453423	Yes
namull.11.jpg	0.48309837097908737	Yes
namull.12.jpg	0.48312326468475714	Yes
namull.13.jpg	0.486709853926479	Yes
namull.14.jpg	0.49657762085731155	Yes
namull.16.jpg	0.47954763770455816	Yes
namull.17.jpg	0.49958672970235724	Yes
namull.19.jpg	0.49557076522667326	Yes

6.4.4 Test 4

Test image: essex_cswww/grimace/glen/glen_exp.16.jpg

Image	Distance	False Positive
glen_exp.1.jpg	0.34313537619742157	No
glen_exp.2.jpg	0.32513380486202487	No
glen_exp.3.jpg	0.328383372717064	No
glen_exp.4.jpg	0.3491545736970797	No
glen_exp.5.jpg	0.30237431205513626	No
glen_exp.6.jpg	0.29009140244087817	No
glen_exp.7.jpg	0.24115004735784007	No
glen_exp.8.jpg	0.32010274711449116	No
glen_exp.9.jpg	0.24679108112701786	No
glen_exp.10.jpg	0.25755122131334396	No
glen_exp.11.jpg	0.25527497911678315	No
glen_exp.12.jpg	0.2613227017464422	No
glen_exp.13.jpg	0.15115098114591524	No
glen_exp.14.jpg	0.14626106627309007	No
glen_exp.15.jpg	0.12721964956014153	No
glen_exp.16.jpg	0	No
glen_exp.17.jpg	0.1282296546247514	No
glen_exp.18.jpg	0.15230068444806952	No
glen_exp.19.jpg	0.1574033445984365	No
glen_exp.20.jpg	0.171147992496858	No

6.4.5 Test 5

Test image: lfw/Aaron_Peirsol/Aaron_Peirsol_0001.jpg

Image	Distance	False Positive
Aaron_Peirsol_0001.jpg	0	No
Aaron_Peirsol_0002.jpg	0.359368799618006	No
Aaron_Peirsol_0003.jpg	0.3356236792942648	No
Aaron_Peirsol_0004.jpg	0.41244297976804756	No

6.5 Comparing Images with Different Compression Algorithms

Test Number	Accuracy	False Positives	Time Elapsed (Hours)
1	1.0	0	1:23:20
2	0.25	3	1:22:42
3	0.72	1	1:24:10
4	0.5	2	1:22:46
5	0.82	0	1:23:16
Avg	0.658	1.2	1:23:15

6.5.1 Test 1

Test image: lfw/Aaron_Peirsol/Aaron_Peirsol_0001.jpg

Image	Distance	False Positive
Aaron_Peirsol_0001.png	0.012334246377964656	No
Aaron_Peirsol_0002.png	0.3603868116326761	No
Aaron_Peirsol_0003.png	0.3358269006565544	No
Aaron_Peirsol_0004.png	0.4213942834670981	No

6.5.2 Test 2

Test image: lfw/Doc_Rivers/Doc_Rivers_0001.jpg

Image	Distance	False Positive
Doc_Rivers_0001.png	0.008126612697683498	No
Glenn_Rivers_0001.png	0.4441926835082163	Yes
Maurice_Cheeks_0001.png	0.4755522108329577	Yes
Thabo_Mbeki_0001.png	0.48904588661316073	Yes

6.5.3 Test 3

Test image: lfw/George_HW_Bush_0003.jpg

Image	Distance	False Positive
George_HW_Bush_0001.png	0.43779864167037513	No
George_HW_Bush_0003.png	0.05526240671390998	No
George_HW_Bush_0005.png	0.49556510308006735	No
George_HW_Bush_0006.png	0.49678314589520917	No
George_HW_Bush_0007.png	0.4661770164726044	No
George_HW_Bush_0009.png	0.4343336388504174	No
George_HW_Bush_0011.png	0.41749203800312495	No
George_HW_Bush_0012.png	0.4958857255192459	No
George_HW_Bush_0013.png	0.4970243387404675	No
George_W_Bush_0287.png	0.44049867749481153	Yes

6.5.4 Test 4

Test image: lfw/Joseph_Blatter_0002.jpg

Image	Distance	False Positive
Joseph_Blatter_0001.png	0.47858057359342215	No
Joseph_Blatter_0002.png	0.0336530518133179	No
Sepp_Blatter_0002.png	0.4862694459206191	Yes
Sepp_Blatter_0004.png	0.42559962697675985	Yes

6.5.5 Test 5

Test image: lfw/Tony_Blair_0002.jpg

Image	Distance	False Positive
Tony_Blair_0002.png	0.4771680101285018	No
Tony_Blair_0003.png	0.480985145188809	No
Tony_Blair_0004.png	0.4596974293243147	No
Tony_Blair_0006.png	0.036247002247086095	No
Tony_Blair_0007.png	0.3749530714147472	No
Tony_Blair_0009.png	0.44356856961314667	No
Tony_Blair_0012.png	0.39056265389937056	No
Tony_Blair_0013.png	0.36810087134361	No
Tony_Blair_0015.png	0.47290939486890193	No
Tony_Blair_0016.png	0.4799575413945816	No
Tony_Blair_0017.png	0.3791103058656651	No

Image	Distance	False Positive
Tony_Blair_0019.png	0.47370780408374297	No
Tony_Blair_0020.png	0.4030814376470261	No
Tony_Blair_0022.png	0.3484769600414663	No
Tony_Blair_0024.png	0.4749464761064012	No
Tony_Blair_0025.png	0.37668636275839784	No
Tony_Blair_0026.png	0.4540464799599294	No
Tony_Blair_0027.png	0.4175971449726495	No
Tony_Blair_0028.png	0.45196053604334974	No
Tony_Blair_0029.png	0.4040722828269198	No
Tony_Blair_0030.png	0.4040920514555924	No
Tony_Blair_0032.png	0.4403225959394582	No
Tony_Blair_0033.png	0.47767228365950215	No
Tony_Blair_0035.png	0.4457419703771305	No
Tony_Blair_0036.png	0.45523765642848557	No
Tony_Blair_0037.png	0.47821689320577304	No
Tony_Blair_0039.png	0.46474096062460735	No
Tony_Blair_0040.png	0.45155503656989276	No
Tony_Blair_0042.png	0.4625118053790461	No
Tony_Blair_0041.png	0.47343379679317094	No
Tony_Blair_0043.png	0.46666912378683195	No
Tony_Blair_0046.png	0.4029704245820186	No
Tony_Blair_0047.png	0.42564273287725796	No
Tony_Blair_0048.png	0.47068237137426056	No
Tony_Blair_0049.png	0.42269784320192466	No
Tony_Blair_0050.png	0.498966415423654	No
Tony_Blair_0052.png	0.48711129745936227	No
Tony_Blair_0051.png	0.4863045001385002	No
Tony_Blair_0053.png	0.4878306595340014	No
Tony_Blair_0054.png	0.4547578270562901	No
Tony_Blair_0055.png	0.45477935130613995	No
Tony_Blair_0056.png	0.46440980003212756	No
Tony_Blair_0057.png	0.4044213964481114	No
Tony_Blair_0058.png	0.44105103047407657	No
Tony_Blair_0059.png	0.4572087448490241	No
Tony_Blair_0060.png	0.48874981459517125	No
Tony_Blair_0061.png	0.4259232633107541	No
Tony_Blair_0062.png	0.4312476463279641	No
Tony_Blair_0063.png	0.4539069717684182	No
Tony_Blair_0064.png	0.42503974397901295	No
Tony_Blair_0065.png	0.4034136646600798	No
Tony_Blair_0067.png	0.36800919552842437	No

Image	Distance	False Positive
Tony_Blair_0068.png	0.49371680987282746	No
Tony_Blair_0070.png	0.499827996283155	No
Tony_Blair_0071.png	0.47263084324812427	No
Tony_Blair_0072.png	0.4770663160672075	No
Tony_Blair_0073.png	0.49287495683175353	No
Tony_Blair_0078.png	0.42120037854851483	No
Tony_Blair_0080.png	0.4271619535881904	No
Tony_Blair_0081.png	0.45576621662738687	No
Tony_Blair_0082.png	0.4291947084716399	No
Tony_Blair_0083.png	0.47053749832287933	No
Tony_Blair_0084.png	0.49055603003745324	No
Tony_Blair_0085.png	0.48118130886360233	No
Tony_Blair_0086.png	0.39077778686829673	No
Tony_Blair_0087.png	0.47799662896914624	No
Tony_Blair_0088.png	0.45594586876542087	No
Tony_Blair_0089.png	0.3956468551870259	No
Tony_Blair_0091.png	0.4874395113110329	No
Tony_Blair_0092.png	0.4505891128182158	No
Tony_Blair_0093.png	0.47806026796247597	No
Tony_Blair_0096.png	0.4638839966233622	No
Tony_Blair_0097.png	0.426256451794294	No
Tony_Blair_0099.png	0.4111159768122537	No
Tony_Blair_0101.png	0.4141999990896525	No
Tony_Blair_0102.png	0.4700670088189572	No
Tony_Blair_0103.png	0.46185539009939025	No
Tony_Blair_0104.png	0.44897356154630347	No
Tony_Blair_0106.png	0.47399290771782543	No
Tony_Blair_0107.png	0.49320329875468166	No
Tony_Blair_0109.png	0.497298251309488	No
Tony_Blair_0111.png	0.4658191818926708	No
Tony_Blair_0112.png	0.45226067648954343	No
Tony_Blair_0114.png	0.3888446086340705	No
Tony_Blair_0113.png	0.483735856230772	No
Tony_Blair_0116.png	0.4922189710321329	No
Tony_Blair_0117.png	0.4881579137913724	No
Tony_Blair_0118.png	0.4767777939761526	No
Tony_Blair_0119.png	0.4888145552284681	No
Tony_Blair_0122.png	0.4048638517205401	No
Tony_Blair_0121.png	0.43206610453180183	No
Tony_Blair_0123.png	0.4848490145683085	No
Tony_Blair_0124.png	0.4858565492585873	No

Image	Distance	False Positive
Tony_Blair_0126.png	0.49118102706622646	No
Tony_Blair_0125.png	0.44974738645157736	No
Tony_Blair_0129.png	0.46810588485536203	No
Tony_Blair_0130.png	0.4753108653778671	No
Tony_Blair_0132.png	0.4995327443552532	No
Tony_Blair_0134.png	0.4707471675268031	No
Tony_Blair_0133.png	0.46476424681366074	No
Tony_Blair_0136.png	0.4114815889521775	No
Tony_Blair_0138.png	0.45758562676217934	No
Tony_Blair_0139.png	0.4721173528401837	No
Tony_Blair_0140.png	0.47340721200992797	No
Tony_Blair_0141.png	0.39354676586036913	No
Tony_Blair_0142.png	0.38277308586730685	No
Tony_Blair_0143.png	0.44663782899819965	No

6.6 Comparing Images Taken in the Infra-Red Spectrum

Test Number	Accuracy	False Positives	Time Elapsed (Hours)
1	1.0	0	0:01:01
2	0.33	2	0:00:55
3	0.5	1	0:01:56
4	1	0	0:00:57
5	1	0	0:00:58
Avg	0.766	0.6	0:01:09

6.6.1 Test 1

Test image: SCface_database/surveillance_cameras_IR_cam8/001_cam8.jpg

Image	Distance	False Positive
001_cam8.jpg	0	No

6.6.2 Test 2

Test image: SCface_database/surveillance_cameras_IR_cam8/012_cam8.jpg

Image	Distance	False Positive
012_cam8.jpg	0	No
012_cam8.jpg	0.48812956542079605	Yes
012_cam8.jpg	0.4626028092713898	Yes

6.6.3 Test 3

Test image: SCface_database/surveillance_cameras_IR_cam8/018_cam8.jpg

Image	Distance	False Positive
018_cam8.jpg	0	No
106_cam8.jpg	0.4797243721214274	Yes

6.6.4 Test 4

Test image: SCface_database/surveillance_cameras_IR_cam8/050_cam8.jpg

Image	Distance	False Positive
050_cam8.jpg	0	No

6.6.5 Test 5

Test image: SCface_database/surveillance_cameras_IR_cam8/123_cam8.jpg

Image	Distance	False Positive
123_cam8.jpg	0	No

6.7 Restricting by Identifiable Information

Test Number	Accuracy	False Positives	Time Elapsed (Hours)
1	0.72	1	0:00:28
2	1	0	0:00:35
3	1	0	0:00:23
4	0.95	0	0:00:35
5	1	0	0:00:11
Avg	0.934	0.2	0:00:26

6.7.1 Test 1

Test image: lfw/George_HW_Bush_0003.jpg

Image	Distance	False Positive
George_HW_Bush_0001.jpg	0.44468805399677747	No
George_HW_Bush_0003.jpg	0	No
George_HW_Bush_0005.jpg	0.49416427819847825	No
George_HW_Bush_0006.jpg	0.497188652920442	No
George_HW_Bush_0007.jpg	0.46536163951784293	No
George_HW_Bush_0009.jpg	0.43531340164790916	No
George_HW_Bush_0011.jpg	0.41985097908792973	No
George_HW_Bush_0012.jpg	0.4960292649139455	No
George_HW_Bush_0013.jpg	0.49746813708113324	No
George_W_Bush_0287.jpg	0.4454600672429446	Yes

6.7.2 Test 2

Test image: essex_cswww/faces94/male/cjsake/cjsake.1.jpg

Image	Distance	False Positive
cjsake.1.jpg	0	No
cjsake.2.jpg	0.12415972564754801	No
cjsake.3.jpg	0.1202683265955043	No
cjsake.4.jpg	0.12056058735503877	No
cjsake.5.jpg	0.15399061544936016	No
cjsake.6.jpg	0.15170091461313734	No
cjsake.7.jpg	0.15761164790252086	No
cjsake.8.jpg	0.18675358283846008	No
cjsake.9.jpg	0.22679627515383244	No
cjsake.10.jpg	0.16498243163116574	No
cjsake.11.jpg	0.2083636000224947	No
cjsake.12.jpg	0.19579060568070747	No
cjsake.13.jpg	0.21036201522486805	No
cjsake.14.jpg	0.16785907543854067	No
cjsake.15.jpg	0.2078185538842606	No
cjsake.16.jpg	0.19772310008051744	No
cjsake.17.jpg	0.18166327218341696	No
cjsake.18.jpg	0.20414891693929105	No
cjsake.19.jpg	0.19693488266700446	No
cjsake.20.jpg	0.1671873443722762	No

6.7.3 Test 3

Test image: essex_cswww/faces95/adhast/adhast.1.jpg

Image	Distance	False Positive
adhast.1.jpg	0	No
adhast.2.jpg	0.2312083988155676	No
adhast.3.jpg	0.25972150326024057	No
adhast.4.jpg	0.21590055493708946	No
adhast.5.jpg	0.24049261224827648	No
adhast.6.jpg	0.23465681660325552	No
adhast.7.jpg	0.24771615139343836	No
adhast.8.jpg	0.2569527085240263	No
adhast.9.jpg	0.2835389426928616	No
adhast.10.jpg	0.2774791080863853	No
adhast.11.jpg	0.2532400849084287	No
adhast.12.jpg	0.28278790526661146	No
adhast.13.jpg	0.2937914096756732	No
adhast.14.jpg	0.2970183349284969	No
adhast.15.jpg	0.29927374807082874	No
adhast.16.jpg	0.2783214797648933	No
adhast.17.jpg	0.2736083238480712	No
adhast.18.jpg	0.2851054856115722	No
adhast.19.jpg	0.2891104254644535	No
adhast.20.jpg	0.2816904095878937	No

6.7.4 Test 4

Test image: essex_cswww/faces96/cjhewi/cjhewi.1.jpg

Image	Distance	False Positive
cjhewi.1.jpg	0	No
cjhewi.2.jpg	0.19289455983673667	No
cjhewi.3.jpg	0.24900325719699287	No
cjhewi.4.jpg	0.3291783082081157	No
cjhewi.5.jpg	0.31046947257682805	No
cjhewi.6.jpg	0.375778847049278	No
cjhewi.7.jpg	0.49121008818147793	No
cjhewi.8.jpg	0.24928926125340656	No
cjhewi.9.jpg	0.2805170839075122	No
cjhewi.10.jpg	0.2917877509617489	No
cjhewi.12.jpg	0.33383609910398243	No
cjhewi.13.jpg	0.3107498407520015	No
cjhewi.14.jpg	0.3226220896718056	No
cjhewi.15.jpg	0.3315345047726276	No
cjhewi.16.jpg	0.29075881781505053	No
cjhewi.17.jpg	0.3323513335348239	No
cjhewi.18.jpg	0.3216638643477499	No
cjhewi.19.jpg	0.310250334339258	No
cjhewi.20.jpg	0.338702834726292	No

6.7.5 Test 5

Test image: essex_cswww/grimace/glen/glen_exp.16.jpg

Image	Distance	False Positive
glen_exp.1.jpg	0.34313537619742157	No
glen_exp.2.jpg	0.32513380486202487	No
glen_exp.3.jpg	0.328383372717064	No
glen_exp.4.jpg	0.3491545736970797	No
glen_exp.5.jpg	0.30237431205513626	No
glen_exp.6.jpg	0.29009140244087817	No
glen_exp.7.jpg	0.24115004735784007	No
glen_exp.8.jpg	0.32010274711449116	No
glen_exp.9.jpg	0.24679108112701786	No
glen_exp.10.jpg	0.25755122131334396	No
glen_exp.11.jpg	0.25527497911678315	No
glen_exp.12.jpg	0.2613227017464422	No
glen_exp.13.jpg	0.15115098114591524	No
glen_exp.14.jpg	0.14626106627309007	No
glen_exp.15.jpg	0.12721964956014153	No
glen_exp.16.jpg	0	No
glen_exp.17.jpg	0.1282296546247514	No
glen_exp.18.jpg	0.15230068444806952	No
glen_exp.19.jpg	0.1574033445984365	No
glen_exp.20.jpg	0.171147992496858	No

7 Evaluation

7.1 Design

The overall design of the implementation was sound. In particular the producer-consumer model suited the project well as it allows for a thread-based implementation with multiple queues, allowing the computational cost of searching the data set to be spread across multiple CPU cores.

While the design is fairly modular, as it stands the JSON data containing the list of profiles must be in a particular format and must be sent as a single response). This would be inadequate if the application were to be tested against an actual social media service, as the profile data that can be retrieved from their APIs are in differing formats and are typically sent in chunks. For example, Facebook sends its profiles in chunks of 50 (Facebook, 2018*a*). Ideally this problem would be solved by refactoring the code to include a layer between the calling program (run.py) and the python module (facegather.py) in which formats can be defined, and the module called repeatedly on chunks of results. Alternatively, a proxy server could be placed in between the computer running the search and the social media service which caches API calls to the provider and returns them as a single response in the expected format.

The design allows for a multi-threaded architecture, however to see truly optimal performance a design based around GPU computing should be considered. By utilising thousands of cores rather than 4 it is theorised that the most time consuming part of the search, the facial descriptor comparisons, could be reduced by an order of magnitude.

7.2 Implementation

Although there were several challenges to a successful implementation of the design, the program developed achieves all of the functional and non-functional requirements set out in the specification.

There are some gaps in functionality which are desirable but could not be added due to limitations in time. For example, while the search can be restricted by identifying information other than a person of interest's face, the only information currently searchable on is their name. While this feature is obviously useful, more search restrictions could be implemented such as location, place of work, home address etc. to narrow down the search space. This would be important if the program were to be used in a national security or industry setting, as otherwise the data set would be far too large to search in a timely manner.

The implementation uses a command-line interface to allow users to search services with

a test image, specifying search restrictions, threads for producing and consuming data, maximum image load and other parameters. While this is adequate for expert users and is desirable to allow for inter-program communication (for example, search commands being generated by one program and being piped into facegather), non-expert users may struggle to use the program effectively without a GUI.

7.3 Results

7.3.1 Comparing Images from Different Data Sets

The implementation performed admirably when comparing images against multiple data sets, some of which it had not been trained on. It had an average accuracy of 0.92, only marginally less than dlib's advertised performance when searching the Labeled Faces in the Wild data set. The only result that kept it getting a 100% accuracy was test 3, which contained several false positives. All false positives had a distance greater than 0.45, whereas all correct matches had a distance less than 0.35 suggesting that accuracy could be improved by giving a more precise thresholding value. It is unlikely to be a problem with the algorithm itself, as test 4 used a much more difficult data set and was searched with 100% accuracy.

Of particular note is the fact that there was only a single false negative, again in test 3. This is an excellent result, as the intended use of the application is to restrict the search space as much as possible before handing the results to either a secondary program or a human operator. False negatives are therefore much more problematic than false positives, hence why the thresholding value was set to be quite forgiving.

7.3.2 Comparing Images with Different Compression Algorithms

The implementation performed less well when comparing images with different compression algorithms. The accuracy of the search dropped to 0.658, mainly due to the high number of false positives. It should be noted that, even with the substantial drop in accuracy, recall was still fairly high.

The main reason behind the drop in accuracy is most likely due to the changes in the image when compressing greyscale images. The library responsible for facial recognition has been trained mostly on RGB images, so it may have difficulty identifying facial descriptors in greyscale images particularly when faced with a drop in quality.

7.3.3 Comparing Images Taken in the Infra-Red Spectrum

The implementation performed slightly better when comparing images in the infra-red spectrum, achieving an accuracy of 0.766. This is less impressive than the first set

of tests however, as there are far fewer potential false positives in the data set - 129 compared to several thousands for the first and second tests. Again, the algorithm may suffer when comparing greyscale images. Compounding this is the fact that the images were taken in the infra-red spectrum. The images taken from this camera show less distinction between facial features, for example the contrast between where the cheek bones end is much less noticable.

7.3.4 Restricting by Identifiable Information

The results obtained by this set of tests are the most promising in the entire project. The implementation achieved an accuracy of 0.934, the highest in any of the sets of tests. This is to be expected, as by supplying a name to restrict the search by the number of false negatives is reduced from several thousand to at most several hundred. Most impressive is the time saved by restricting the search - the average time taken to search was improved from over 2 hours to just 26 seconds, a time save of over 27,000%.

An interesting note is that the false positive in test 1, where the test image provided was of George H.W. Bush, was a picture of George W. Bush - the subject's son. If the application behaves similarly in a real-life applications this could be a useful unintended side-effect as gathering information on family members (sons, fathers etc.) would be valuable.

8 Conclusion

8.1 Overview

Overall, the project was succesful in applying machine vision techniques to recognise the faces of persons of interest to gather social media intelligence. The application developed allows users to search a social media service for an image of a person of interest, optionally restricting by name. The underlying facial recognition library is based on dlib trained using the Labeled Faces in the Wild data set.

The emergence of social media intelligence has been helped by several controversies recently. With the new exposure, users of social media services are much more aware of how much information they may potentially be exposing about themselves. Even with this newfound knowledge there are some basic datum which every user must make public - their name and profile photo being examples. This makes the tool developed attractive to intelligence services who may be gathering information about large groups of people, but may lack the tools to single out a person of interest from a population.

8.2 Findings

The application was tested using the Labeled Faces in the Wild, Essex University Computer Vision and SCFace databases. It was tested against datasets it was not trained against, data sets using different compression algorithms, data sets captured in the infra-red spectrum, and data sets that had been restricted using identifiable information (in this case, the person of interest's name).

The results are encouraging overall. The application had accuracy rates of over 90% in tests covering multiple datasets, which closely mirror images representative of typical social media profile photos. Importantly, there was only 1 false negative giving a recall of over 95%. The most realistic text, wherein the search space was restricted by name, had an accuracy of 93.4%.

The tests including different compression algorithms and infra-red images were less succesful, although they do offer areas for improvement. These searches test non-essential functionality of the program, and can be considered important for continued improvement but niche in a real-word setting.

8.3 Future Work

There are several improvements that could be made over the current implementation. While facial recognition technology improves, so too will this application. It can be considered a wrapper for whichever face recogntion implementation is performing best

at the current time. The main improvement that could be made is to have GPUs compare face mappings in batches, rather than the current implementation comparing them one-by-one over multiple CPU cores.

The application certainly warrants further research. The initial findings are promising, but an improved implementation tested against an actual social media service (or a data set truly representative of one) could prove groundbreaking for intelligence gathering communities.

References

Antonius, N. and Rich, L. (2013), 'Discovering collection and analysis techniques for social media to improve public safety', **3**, 42.

Bakhshi, T. (2008), 'A practical assessment of social engineering vulnerablities'.

Bartlett, J. and Reynolds, L. (2015), the state of the art 2015 a literature review of social media intelligence capabilities for counter- terrorism, Technical report, Centre for Analysis of Social Media.

Becker, B. C. and Ortiz, E. G. (2008), Evaluation of face recognition techniques for application to facebook, *in* 'Automatic Face & Gesture Recognition, 2008. FG'08. 8th IEEE International Conference on', IEEE, pp. 1–6.

Best-Rowden, L., Han, H., Otto, C., Klare, B. F. and Jain, A. K. (2014), 'Unconstrained face recognition: Identifying a person of interest from a media collection', *IEEE Transactions on Information Forensics and Security* **9**(12), 2144–2157.

Bowyer, K. W. (2004), 'Face recognition technology: security versus privacy', *IEEE Technology and Society Magazine* **23**(1), 9–19.

Caputo, D. D., Pfleeger, S. L., Freeman, J. D. and Johnson, M. E. (2014), 'Going spear phishing: Exploring embedded training and awareness', *IEEE Security & Privacy* **12**(1), 28–38.

Charters, D. A. (2018), 'Professionalizing clandestine military intelligence in northern ireland: creating the special reconnaissance unit', *Intelligence and National Security* **33**(1), 130–138.

Cui, Z., Li, W., Xu, D., Shan, S. and Chen, X. (2013), Fusing robust face region descriptors via multiple metric learning for face recognition in the wild, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 3554–3561.

Evans, J. R., Meissner, C. A., Brandon, S. E., Russano, M. B. and Kleinman, S. M. (2010), 'Criminal versus humint interrogations: The importance of psychological science to improving interrogative practice', *The Journal of Psychiatry & Law* **38**(1-2), 215–249.
URL: *https://doi.org/10.1177/009318531003800110*

Facebook (2018*a*), 'Facebook Graph API Supporting Documents', `https://developers.facebook.com/docs/graph-api/`. Online; accessed 13th March 2018.

Facebook (2018*b*), 'Facebook People Search Page', `https://www.facebook.com/people-search.php`. Online; accessed 13th March 2018.

Facebook (2018*c*), 'How does Facebook suggest tags?', `https://www.facebook.com/help/122175507864081?helpref=faq_content`. Online; accessed 2nd March 2018.

Gallagher, P. (2005), 'Download festival: Facial recognition technology used at event could be coming to festivals nationwide'.

Gioe, D. V. (2017), the more things change: Humint in the cyber age, *in* 'The Palgrave Handbook of Security, Risk and Intelligence', Springer, pp. 213–227.

Introna, L. and Wood, D. (2002), 'Picturing algorithmic surveillance: The politics of facial recognition systems', *Surveillance & Society* **2**(2/3).

Kandias, M. and Stavrou, V. (2015), 'Personal traits analysis as a means to predict insiders'.

Kilburn, M. and Krieger, L. (2014), 'Policing in an information age: The prevalence of state and local law enforcement agencies utilising the world wide web to connect with the community', *International Journal of Police Science & Management* **16**(3), 221–227.
URL: *https://doi.org/10.1350/ijps.2014.16.3.341*

Klare, B. F., Klein, B., Taborsky, E., Blanton, A., Cheney, J., Allen, K., Grother, P., Mah, A. and Jain, A. K. (2015), Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 1931–1939.

Klontz, J. C. and Jain, A. K. (2013), 'A case study on unconstrained facial recognition using the boston marathon bombings suspects', *Michigan State University, Tech. Rep* **119**(120), 1.

Krombholz, K., Hobel, H., Huber, M. and Weippl, E. (2015), 'Advanced social engineering attacks', *Journal of Information Security and applications* **22**, 113–122.

Livingstone, S. (2008), 'Taking risky opportunities in youthful content creation: teenagers' use of social networking sites for intimacy, privacy and self-expression', *New Media & Society* **10**(3), 393–411.
URL: *https://doi.org/10.1177/1461444808089415*

Mateescu, A., Brunton, D., Rosenblat, A., Patton, D., Gold, Z. and Boyd, D. (2015), 'Social media surveillance and law enforcement', *Data Civil Rights, October* **27**, 2015–1027.

Mitnick, K. D. and Simon, W. L. (2011), *The art of deception: Controlling the human element of security*, John Wiley & Sons.

Musco, S. (2017), 'The art of meddling: a theoretical, strategic and historical analysis of non-official covers for clandestine humint', *Defense & Security Analysis* **33**(4), 380–394.

Omand, S. D., Bartlett, J. and Miller, C. (2012), 'Introducing social media intelligence (socmint)', *Intelligence and National Security* **27**(6), 801–823.
URL: *https://doi.org/10.1080/02684527.2012.716965*

Parmar, B. (2012), 'Protecting against spear-phishing', **2012**, 811.

pipl (2018), 'pipl home page', `https://pipl.com/`. Online; accessed 13th March 2018.

Ruiz, J. (2018), Gchq and mass surveillance, Technical report, Open Rights Group.

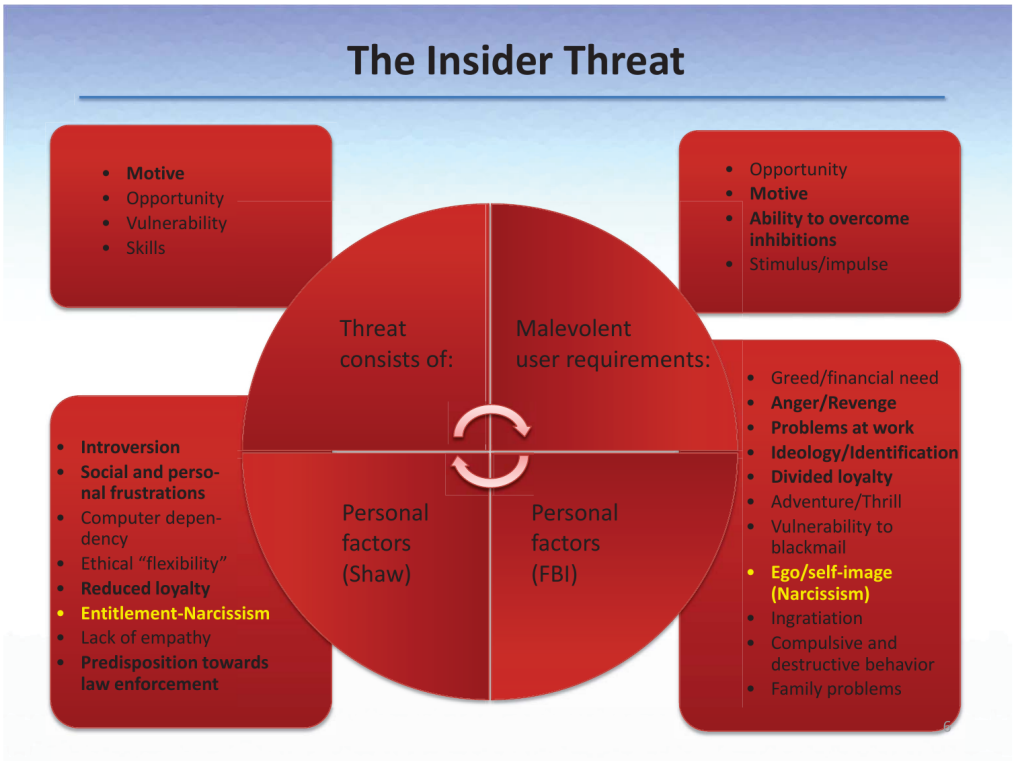
Russano, M. B., Narchet, F. M., Kleinman, S. M. and Meissner, C. A. (2014), 'Structured interviews of experienced humint interrogators', *Applied cognitive psychology* **28**(6), 847–859.

Timm, T. (2015), 'The cia director was hacked by a 13-year-old, but he still wants your data', Published in the Guardian newspaper.

Unknown (2013), 'Prism slides', Leaked to several newspapers by Edward Snowden in late 2013.

Appendices

A Threat Graph



Graph of insider threat factors (Kandias and Stavrou, 2015).

B "Boston Bomber" Identification

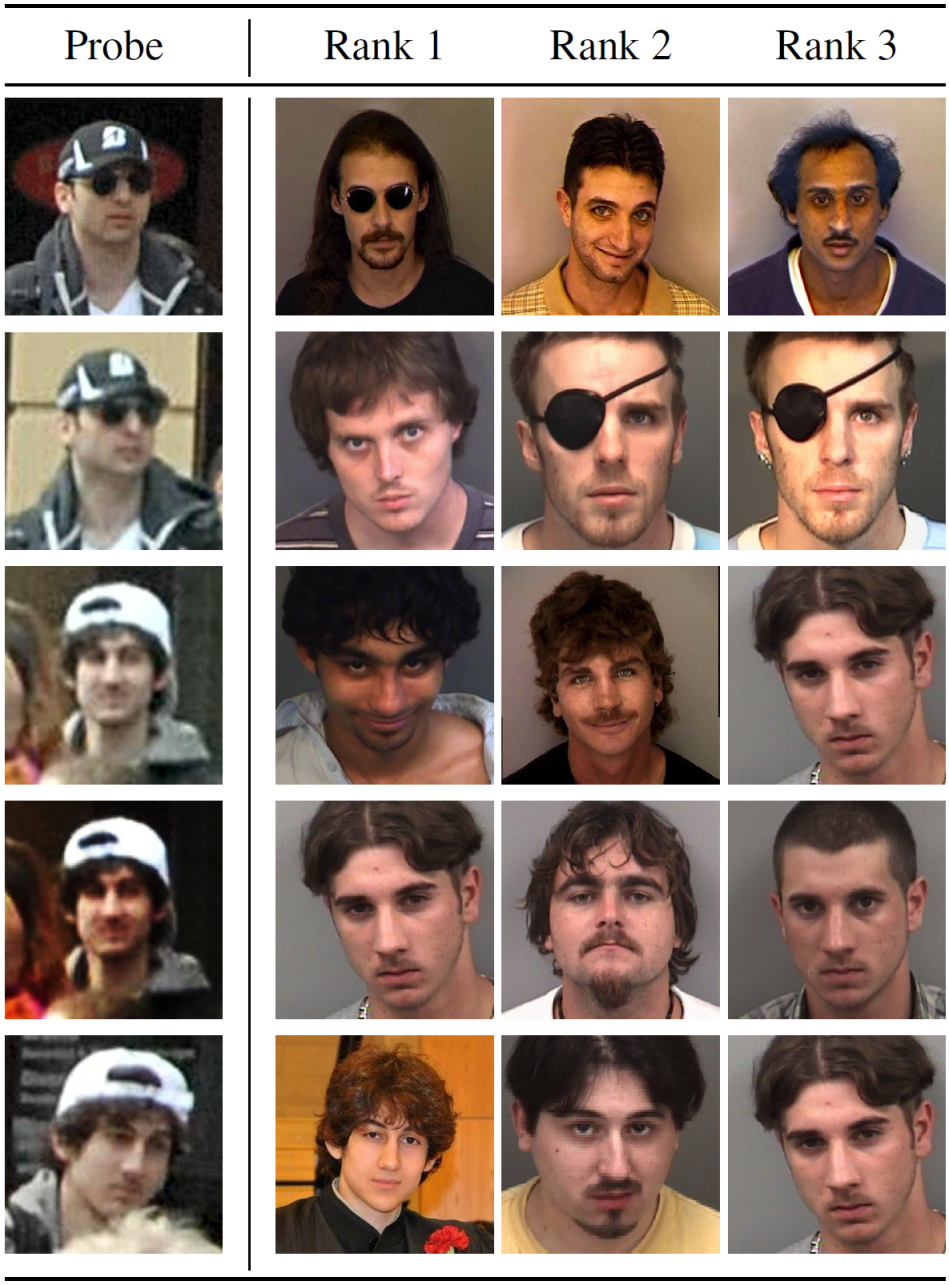


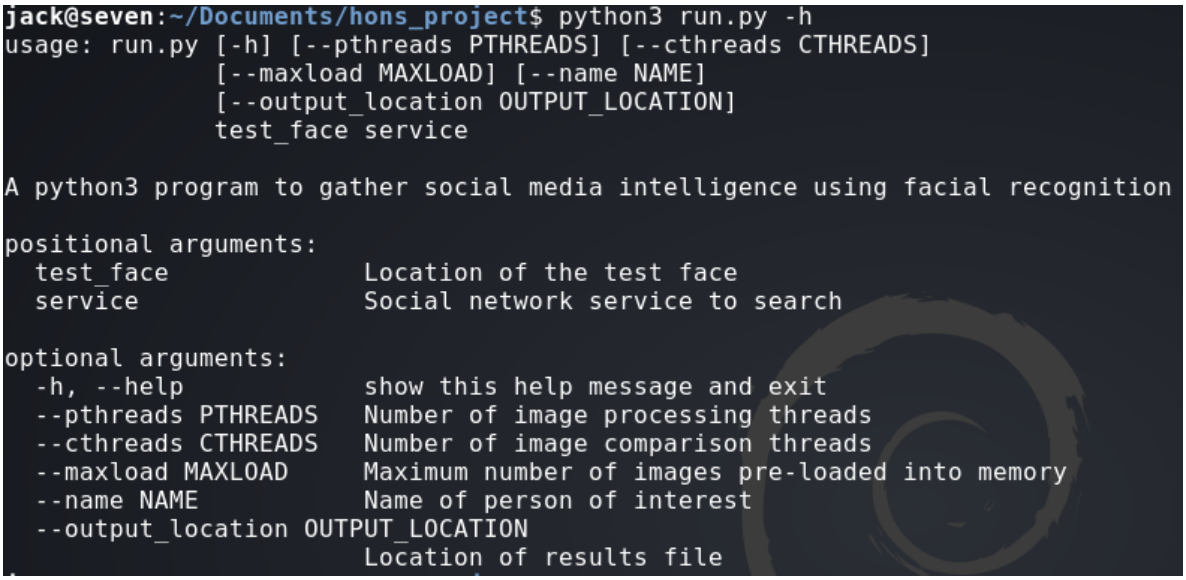
Table of potential matches, note the correct identification from the picture taken from social media with similar pose and lighting (Klontz and Jain, 2013).

C Connection Graph

Category	Influence valuation	Klout score	Usage valuation
Loners	0 - 90	3.55 - 11.07	0 - 500
Individuals	90 - 283	11.07 - 26.0	500 - 4.500
Known users	283 - 1.011	26.0 - 50.0	4.500 - 21.000
Mass Media & Personas	1.011 - 3.604	50.0- 81.99	21.000 - 56.9000

Graph of social media connections and klout score in the Nereus framework(Kandias and Stavrou, 2015).

D Command Line Arguments



A help page showing the command line arguments that can be supplied.

E GitHub Commit log

Add commit log

F Full Source Code

F.1 facegather.py

```
from PIL import Image
from io import BytesIO
import requests
import face_recognition as fr
import threading
import queue
import numpy
import sentinels
import time
import json

IMAGE_LOAD_MAX = 200
DEFAULT_THRESHOLD = 0.5
profile_queue = queue.Queue()
img_queue = queue.Queue()


# Load an image in to memory from local or remote sources
def load_images(uri, remote=True):
    if remote:
        resp = requests.get(uri, stream=True)
        try:
            img = Image.open(BytesIO(resp.content))
            return fr.face_encodings(numpy.array(img))
        except:
            # Catch line in case PNG file is encoded as greyscale
            return fr.face_encodings(numpy.array(img.convert('RGB')))
    else:
        img_array = fr.load_image_file(uri)
    return fr.face_encodings(img_array)


# Search a data set at the given URI for the best matches to the test face
def search(test_face_location,
           uri,
           no_producer_threads=None,
           no_consumer_threads=None,
           max_loaded=None,
           threshold=None,
           name=None,
           output_location=None):
```

```
# Load default values in case called with "None"
if no_producer_threads is None:
    no_producer_threads = 1
if no_consumer_threads is None:
    no_consumer_threads = 1
if max_loaded is None:
    max_loaded = IMAGE_LOAD_MAX
if threshold is None:
    threshold = DEFAULT_THRESHOLD

result = []
result_lock = threading.Lock()

profile_queue_counter = threading.Semaphore(int(max_loaded))
print('Getting profile information...')
if name is None:
    print(uri)
    profiles = _get_profiles(uri)
else:
    print(uri + '?name=' + name)
    profiles = _get_profiles(uri + '?name=' + name)
for profile in profiles:
    profile_queue.put([profile['image_location'], profile])
print('Done')

print('Processing images...')
processor_list = []
for i in range(0, int(no_producer_threads)):
    processor = ImageProcessor(profile_queue_counter)
    processor.start()
    processor_list.append(processor)

recogniser_list = []
for i in range(0, int(no_consumer_threads)):
    recogniser = FaceRecogniser(test_face_location,
        profile_queue_counter, result, result_lock, threshold)
    recogniser.start()
    recogniser_list.append(recogniser)

for processor in processor_list:
    processor.join()
img_queue.put(sentinels.NOTHING)

for recogniser in recogniser_list:
    recogniser.join()
```

```
    print('Done')
    if output_location is None:
        return result
    else:
        output_file = open(output_location + '/facegather_' +
            str(time.time()) + '.txt', 'w')
        output_file.write(json.dumps(result))
        return result


def _get_profiles(uri, name=None):
    if name is not None:
        uri += '?name='
        uri += name
    resp = requests.get(uri)
    return resp.json()


# Class based thread to take a profile from the queue, download the profile
    photo, and pre-process it to be ready for
# facial recognition by 1 or more FaceRecogniser threads
class ImageProcessor(threading.Thread):
    def __init__(self, counter):
        threading.Thread.__init__(self)
        self.waiting_counter = counter

    # Take a URI from the queue, download image from that location then add
        face mappings and profile uri to the queue
    # to be consumed by a FaceRecogniser
    def run(self):
        while True:
            self.waiting_counter.acquire()
            if profile_queue.empty():
                return
            profile = profile_queue.get()

            img_array = load_images(profile[0])
            for img in img_array:
                img_queue.put([img, profile[1]])


# Class based thread to take images loaded in to memory and rank them in
    terms of similarity to the test image.
class FaceRecogniser(threading.Thread):
    finished_flag = False
```

```
    def __init__(self, test_face_location, waiting_counter, result,
        result_lock, threshold):
        threading.Thread.__init__(self)
        self.test_face =
            fr.face_encodings(fr.load_image_file(test_face_location))[0]
        self.counter = waiting_counter
        self.result = result
        self.result_lock = result_lock
        self.threshold = threshold

    # Take a face mapping from the queue, then test for similarity.
    def run(self):
        while True:
            img = img_queue.get()
            if img == sentinels.NOTHING:
                img_queue.put(sentinels.NOTHING)
                return

            if fr.face_distance([self.test_face], img[0]) < self.threshold:
                distance = fr.face_distance([self.test_face], img[0])
                img[1]['distance'] = distance[0]
                self.result_lock.acquire()
                self.result.append(img[1])
                self.result_lock.release()
            self.counter.release()
```

F.2 run.py

```
from facegather import facegather
import face_recognition
import argparse

parser = argparse.ArgumentParser(
    description='A python3 program to gather social media intelligence using
        facial recognition')
parser.add_argument("test_face", help='Location of the test face')
parser.add_argument("service", help='Social network service to search')

parser.add_argument("--pthreads", help='Number of image processing threads')
parser.add_argument("--cthreads", help='Number of image comparison threads')
parser.add_argument("--maxload", help='Maximum number of images pre-loaded
    into memory')
parser.add_argument("--name", help='Name of person of interest')
```

```
parser.add_argument("--output_location", help='Location of results file')

args = parser.parse_args()

uri = ''
if args.service == 'demo':
    uri = 'http://localhost:8081/static'

    test_face = facegather.load_images(uri +
        '/lfw/Aaron_Peirsol/Aaron_Peirsol_0001.jpg')[0]
    test_face2 = facegather.load_images(uri +
        '/lfw/Aaron_Peirsol/Aaron_Peirsol_0002.jpg')[0]
    result = face_recognition.face_distance([test_face], test_face2)
    print(result)

    test_face = facegather.load_images(
        '/home/jack/Documents/hons_project/test/lfw/Aaron_Peirsol/Aaron_Peirsol_0001.jpg',
        remote=False)[0]
    test_face2 = facegather.load_images(
        '/home/jack/Documents/hons_project/test/lfw/Aaron_Peirsol/Aaron_Peirsol_0002.jpg',
        remote=False)[0]
    result = face_recognition.face_distance([test_face], test_face2)
    print(result)

elif args.service == 'test':
    uri = 'http://localhost:8081/'
    result = ''
    if args.name is not None:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name,
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name
```

```
            )
    else:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
            )
    print(result)

elif args.service == 'test_multiple_db':
    uri = 'http://localhost:8081/test_multiple_datasets/'
    result = ''
    if args.name is not None:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name,
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name
            )
    else:
        if args.output_location is not None:
```

```
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,https://www.facebook.com/
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
            )
    print(result)

elif args.service == 'test_image_compression':
    uri = 'http://localhost:8081/test_image_compression/'
    result = ''
    if args.name is not None:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name,
                output_location=args.output_location
            )https://www.facebook.com/
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name
            )
    else:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
```

```
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
            )
    print(result)

elif args.service == 'test_IR_RGB':
    uri = 'http://localhost:8081/test_IR_RGB/'
    result = ''
    if args.name is not None:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name,
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name
            )
    else:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
```

```
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
            )
    print(result)

elif args.service == 'test_IR_images':
    uri = 'http://localhost:8081/test_IR_images/'
    result = ''
    if args.name is not None:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name,
                output_location=args.output_location
            )
        else:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                name=args.name
            )
    else:
        if args.output_location is not None:
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
                output_location=args.output_location
            )
        else:
```

```
            result = facegather.search(
                args.test_face,
                uri,
                no_producer_threads=args.pthreads,
                no_consumer_threads=args.cthreads,
                max_loaded=args.maxload,
            )
    print(result)
```

F.3 server.py

```
import cherrypy as cp
import glob


class EnumerateFiles:
    @cp.expose
    def index(self, **kwargs):
        directory = 'test/**'
        if 'name' in kwargs:
            directory = 'test/**/' + kwargs['name']

        sb = '['
        for location in glob.glob(directory + '/*.json', recursive=True):
            if not location[-9:] == '_png.json':
                f = open(location, 'r')
                sb += f.read()
                sb += ','https://www.facebook.com/
                f.close()
        if sb.endswith('['):
            return '[]'
        else:
            return sb[:-1] + ']'


class TestMultipleDB:
    @cp.expose
    def index(self, **kwargs):
        directories = ['test/lfw/', 'test/essex_cswww/']
        sb = '['
        for directory in directories:
            if 'name' in kwargs:
                directory = directory + kwargs['name'] + '/'
```

```
            for location in glob.glob(directory + '**/*.json',
                recursive=True):
                f = open(location, 'r')
                sb += f.read()
                sb += ','
                f.close()
        sb = sb[:-1] + ']'
        return sb


class TestImageCompression:
    @cp.expose
    def index(self):
        sb = '['
        for location in glob.glob('test/lfw/**/*_png.json', recursive=True):
            f = open(location, 'r')
            sb += f.read()
            sb += ','
            f.close()
        sb = sb[:-1] + ']'
        return sb


class TestIRImages:
    @cp.expose
    def index(self):
        sb = '['
        for location in
            glob.glob('test/SCface/SCface_database/surveillance_cameras_IR_cam8/*.json'):
            f = open(location, 'r')
            sb += f.read()
            sb += ','
            f.close()
        sb = sb[:-1] + ']'
        return sb


if __name__ == '__main__':
    cp.config.update({
        'server.socket_port': 8080,
        'tools.proxy.on': True,
        'tools.proxy.base': 'localhost'
    })

    home = EnumerateFiles()
```

```
    cp.tree.mount(EnumerateFiles(), '/', None)
    cp.tree.mount(TestMultipleDB(), '/test_multiple_datasets', None)
    cp.tree.mount(TestImageCompression(), '/test_image_compression', None)
    cp.tree.mount(TestIRImages(), '/test_IR_images', None)

    cp.engine.start()
    cp.engine.block()
```

F.4 compression_algorithm_test.sh

```
#!/bin/sh
python3 run.py ./test/lfw/Aaron_Peirsol/Aaron_Peirsol_0001.jpg
    test_image_compression --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/compression_algorithm_test;
python3 run.py ./test/lfw/Doc_Rivers/Doc_Rivers_0001.jpg
    test_image_compression --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/compression_algorithm_test;
python3 run.py ./test/lfw/George_HW_Bush/George_HW_Bush_0003.jpg
    test_image_compression --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/compression_algorithm_test;
python3 run.py ./test/lfw/Joseph_Blatter/Joseph_Blatter_0002.jpg
    test_image_compression --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/compression_algorithm_test;
python3 run.py ./test/lfw/Tony_Blair/Tony_Blair_0006.jpg
    test_image_compression --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/compression_algorithm_test;
```

F.5 identifying_information_restriction_test.sh

```
#!/bin/sh
python3 run.py ./test/lfw/George_HW_Bush/George_HW_Bush_0003.jpg test
    --pthreads=2 --cthreads=2 --maxload=200 --name="George*"
    --output_location=./test/results/identifying_information_restriction_test;
python3 run.py ./test/essex_cswww/faces94/male/cjsake/cjsake.1.jpg test
    --pthreads=2 --cthreads=2 --maxload=200 --name="cj*"
    --output_location=./test/results/identifying_information_restriction_test;
python3 run.py ./test/essex_cswww/faces95/adhast/adhast.1.jpg test
    --pthreads=2 --cthreads=2 --maxload=200 --name="ad*"
    --output_location=./test/results/identifying_information_restriction_test;
python3 run.py ./test/essex_cswww/faces96/cjhewi/cjhewi.1.jpg test
    --pthreads=2 --cthreads=2 --maxload=200 --name="cj*"
```

```
    --output_location=./test/results/identifying_information_restriction_test;
python3 run.py ./test/essex_cswww/grimace/glen/glen_exp.16.jpg test
    --pthreads=2 --cthreads=2 --maxload=200 --name="glen*"
    --output_location=./test/results/identifying_information_restriction_test;
```

F.6 IR_image_test.sh

```
#!/bin/sh
python3 run.py
    ./test/SCface/SCface_database/surveillance_cameras_IR_cam8/001_cam8.jpg
    test_IR_images --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/IR_image_test;
python3 run.py
    ./test/SCface/SCface_database/surveillance_cameras_IR_cam8/012_cam8.jpg
    test_IR_images --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/IR_image_test;
python3 run.py
    ./test/SCface/SCface_database/surveillance_cameras_IR_cam8/018_cam8.jpg
    test_IR_images --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/IR_image_test;
python3 run.py
    ./test/SCface/SCface_database/surveillance_cameras_IR_cam8/050_cam8.jpg
    test_IR_images --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/IR_image_test;
python3 run.py
    ./test/SCface/SCface_database/surveillance_cameras_IR_cam8/123_cam8.jpg
    test_IR_images --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/IR_image_test;
```

F.7 multiple_dataset_test.sh

```
#!/bin/sh
python3 run.py ./test/essex_cswww/faces94/male/cjsake/cjsake.1.jpg
    test_multiple_db --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/multiple_dataset_test;
python3 run.py ./test/essex_cswww/faces95/adhast/adhast.1.jpg
    test_multiple_db --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/multiple_dataset_test;
python3 run.py ./test/essex_cswww/faces96/cjhewi/cjhewi.1.jpg
    test_multiple_db --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/multiple_dataset_test;
```

```
python3 run.py ./test/essex_cswww/grimace/glen/glen_exp.16.jpg
    test_multiple_db --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/multiple_dataset_test;
python3 run.py ./test/lfw/Aaron_Peirsol/Aaron_Peirsol_0001.jpg
    test_multiple_db --pthreads=2 --cthreads=2 --maxload=200
    --output_location=./test/results/multiple_dataset_test;
```

G generate_test_data.py

```
import glob


def generate_data(root_dir):
    for location in glob.glob(root_dir + '/lfw/**/*.jpg', recursive=True):
        file = open(location[:-3] + 'json', 'w')
        towrite = '{"name":"%s", "image_location":"%s"}' %
            (location.split('/')[-2],
            'http://localhost:8081/static'+location[1:])
        file.write(towrite)
        file.close()

    # for location in glob.glob(root_dir + '/lfw/**/*.jpg', recursive=True):
    #    print(location)
    #    subprocess.run(["convert", location, "-colorspace", "sRGB", "-type",
        "truecolor", location[:-3] + "png"])

    for location in glob.glob(root_dir + '/lfw/**/*.png', recursive=True):
        file = open(location[:-4] + '_png.json', 'w')
        towrite = '{"name":"%s", "image_location":"%s"}' %
            (location.split('/')[-2],
            'http://localhost:8081/static'+location[1:])
        file.write(towrite)
        file.close()

    for location in glob.glob(root_dir + '/essex_cswww/**/*.jpg',
        recursive=True):
        file = open(location[:-3] + 'json', 'w')
        towrite = '{"name":"%s", "image_location":"%s"}' %
            (location.split('/')[-2],
            'http://localhost:8081/static'+location[1:])
        file.write(towrite)
        file.close()
```

```
    # SCface db has several formats, need to be specific
    for location in glob.glob(root_dir +
        '/SCface/SCface_database/mugshot_frontal_cropped_all/*.JPG',
        recursive=True):
        file = open(location[:-12] + '.json', 'w')
        towrite = '{"name":"%s", "image_location":"%s"}' %
            (location.split('/')[-1][:3],
            'http://localhost:8081/static'+location[1:])
        file.write(towrite)
        file.close()

    for location in glob.glob(root_dir +
        '/SCface/SCface_database/mugshot_frontal_original_all/*.jpg',
        recursive=True):
        file = open(location[:-12] + '.json', 'w')
        towrite = '{"name":"%s", "image_location":"%s"}' %
            (location.split('/')[-1][:3],
            'http://localhost:8081/static'+location[1:])
        file.write(towrite)
        file.close()

    for location in glob.glob(root_dir +
        '/SCface/SCface_database/mugshot_rotation_all/*.jpg', recursive=True):
        name = location.split('/')[-1][:-4]
        towrite = '{"name":"%s", "image_location":"%s"}' % (name,
            'http://localhost:8081/static' + location[1:])
        file = open(location[:-3] + 'json', 'w')
        file.write(towrite)
        file.close()

    for location in glob.glob(root_dir +
        '/SCface/SCface_database/surveillance_cameras_*/**/*.jpg',
        recursive=True):
        name = location.split('/')[-1][:-4]
        towrite = '{"name":"%s", "image_location":"%s"}' % (name,
            'http://localhost:8081/static' + location[1:])
        file = open(location[:-3] + 'json', 'w')
        file.write(towrite)
        file.close()


generate_data('.')
```

H Example nginx configuration file

```
upstream app_server {
   server localhost:8080;
}

gzip_http_version 1.0;
gzip_proxied    any;
gzip_min_length    500;
gzip_disable    "MSIE [1-6]\.";
gzip_types    text/plain text/xml text/css
        text/javascript
        application/json;

server {
   listen 8081;
   location ^~ /static {
      alias /home/jack/Documents/hons_project/test/;
   }

   location / {
      proxy_pass http://app_server;
      proxy_redirect off;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Host $server_name;
   }
}
```

this