PCA-Based Face Detection using FOSCAM IP Camera and Facebook

A Fall 2013 Computer Science 105 Final Project Report

Yasha S. Iravantchi

S.B. Candidate, Engineering Sciences

Harvard University

School of Engineering and Applied Sciences

Cambridge, MA 02138

December 11, 2013

Abstract

Off-the-shelf security cameras do not employ ”smart” features. The FOSCAM FI8910W (purchased for $50) features orientation control via Web, motion-detect image capture, FTP upload of images, and IR for night-vision. However, Motion-detection can be triggered very easily and at lowest sensitivity, motion detection is mostly triggered by environment changes. Typical image captures data sets were about 900 images/day and about 20M images/day if set to video. In cases of emergency, such as burglary, this is too much data for a human to manually process within a reasonable time scale. The goal of this CS105 Final Project was to make this camera ”smart” and to enable certain security features found on more robust surveillance systems. Specifically, in this project, I integrated PCA-based facial recognition capabilities, with Facebook as a face training database, to the FOSCAM IP camera.

Keywords: Cyber Privacy, PCA, Facial Recognition, FOSCAM IP, Facebook API

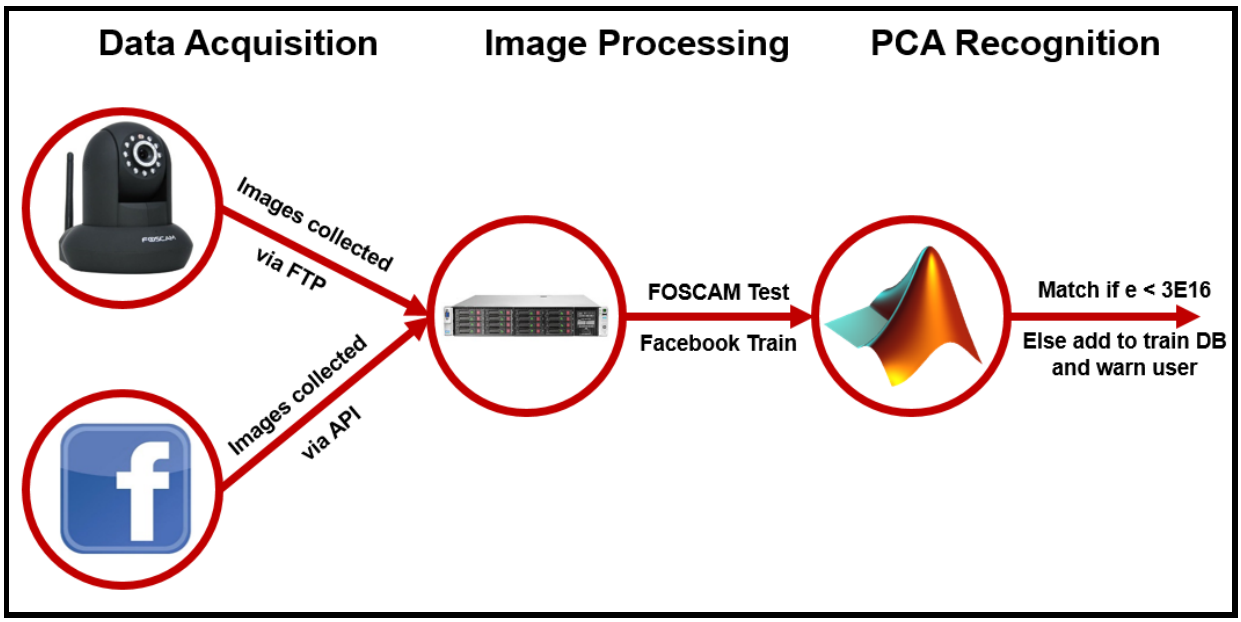


Fig. 1. Graphical representation of the implemented, PCA-based system

CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. Introduction

A. Problem Statement

Off-the-shelf security cameras do not employ smart features. The FOSCAM FI8910W (purchased for $50) features orientation control via Web, motion-detect image capture, FTP upload of images, and IR for night-vision. However, Motion-detection can be triggered very easily and at lowest sensitivity, motion detection mostly triggered by environment changes. Typical image captures data sets were about 900 images/day, 20M images/day if set to video. In case of emergencies, such as burglary, this is too much data for a human to manually process. The goal of this CS105 Final Project was to make this camera smart and to enable certain security features found on more robust surveillance systems, such as facial recognition.

B. Motivation

As stated earlier, the primary motivation was to enable a security camera to produce an output that is immediately useful to a human. Providing tens of thousands of images is not useful for trivial applications, such as when someone is looking for who stole a cookie from the cookie jar, nor is it immediately useful when someone needs a name and a face in emergency circumstances. As so, an immediate motivation towards using Facebook stems from the fact that theft is often done by someone you know, especially in the case of identity theft.[1] As more and more of our social lives are represented online, the best place to start mining for imageswould be Facebook.

C. State of the Art

Matching faces to names is not a new problem: the NSA and other three-letter acronym agencies are well aware of the computational expense of face detection, let alone face recognition and classification. However, it does not seem, at least publicly, that these agencies are mining social networks as a means of matching faces to names. In the case of the 2013 Boston Bombing, the Tsarnaev brothers were first identified using crowd-dynamic software by walking in the

opposite direction of the crowd.[2] Their identities were not immediately known until the FBI went public with their images, using crowdsourcing as a means of putting an identity to a face, not facial recognition.[2] It seems that often times, facial recognition software is used as a means of confirmation or comparison rather than a means of discovery. For example, the 9/11 commission reviewed security tapes after the fact to confirm the identity of the terrorist hijackers. [3]

Currently, the most sophisticated public use of matching faces to names is actually Facebook itself.[4] When a photo is uploaded to Facebook, the page will nudge a user to tag people in the image, having already pre-detected some faces in the image. Upon clicking a face, Facebook will provide a best guess on who that person is, with the user either confirming by clicking on that selection or manually entering another selection.

D. Potential Applications

Potential applications of such a system exist both in the commercial and government realms. The commercial applications that immediately come to mind are private home and office security systems where theft can be a common occurrence. Government application could include a security system that surveys public areas that are potential targets, such as airports, parks, and high-profile office buildings.

II. Design Goals

The main goal of this system was to first isolate faces detected in images captured by the FOSCAM IP Camera then report a name by comparing that isolated face to a database of faces collected from Facebook. I set quantitative goals for each step of the process so that I can objectively evaluate the project's success.

A. *Face detect from FOSCAM IP-sourced Image*

The first criteria was the length of time that my system took from receiving an image from an FTP server to producing a detected face that is cropped and resized from the image. I determined that this should be accomplished in less than 10 seconds per photo, regardless of the number of faces in the photo.

B. *Harvesting Data from Facebook for Training Database*

The second criteria was the length of time that my system took to harvest a single user's Facebook profile of all images they own. Granted, this step is heavily dependent on factors such as internet bandwitdth, but I determined that a fair amount of time would be less than 60 seconds on average per each profile. This does not include running the Facebook photos through my face detector and isolating the faces from the image.

C. *Matching FOSCAM-sourced Face to Facebook-sourced Face*

The last criteria was the length of time that my system took to match the harvested faces from the FOSCAM IP camera to the harvested faces from Facebook photos. I determined that this task should be accomplished in less than 15 seconds per face given a training database of less than 1000 faces. Please see the "Significant Challenges and Pitfalls" section for greater detail as to why the database was restricted to 1000 faces.

TABLE I
THE FOLLOWING TABLE SHOWS A LIST OF CRITERIA WHICH MEETING THE TARGET WOULD DEEM THIS PROJECT A SUCCESS.

Criteria	**Target**
Time to detect and harvest face	$\leq$ 10 seconds per image
Time to download Facebook photos	$\leq$ 60 seconds per profile
Time to match harvested face to Facebook face	$\leq$ 15 seconds per face

III. Design Overview

The project detailed in this paper intends to develop a security system built around the FOSCAM IP camera with the ability to perform facial recognition and classification. The project intends on using mainstream resources accessible to the average person: not everyone has access to a fast cloud based computing environment such as FAS Orchestra. Execution of this project falls under three main sections: acquisition, face detection, face recognition and classification.

A. Data Acquisition

Under acquisition, we will use the FOSCAM IP camera under motion-detection mode, where it will capture an image under perceived motion detection. An offsite FTP depository will be set up so that the FOSCAM IP camera can dump images with complete EXIF data onto the server. After we acquire these images, a script will be run to preprocess the images in a convenient format, such as timestamps for file names that match the EXIF data.

In conjunction with the images we receive from the camera, we use the Facebook API to pull images of my friends on Facebook to populate a training database in which we can attempt to make matches to from the faces we gather from the security camera. This is achieved by using a piece of open-source software called Photograbber.

Photograbber is an open-source (GPLv3) Facebook photo downloading utility written in Python.[5] It allows for a very straightforward downloading of high-resolution photos organized by user and album and uses the Facebook API to download photos directly to the users PC. However, Photograbber has some drawbacks, such as the limitations of the Facebook API (see Significant Challenges and Pitfalls). Using the latest GitHub version, I modified the Python code to output the last person who could be harvested before the Facebook API limit kicked in so that I could increment the IP by one and start Photograbber again.

B. Image Processing

We will then use the MatLab Vision Toolbox to detect faces in images for both the FOSCAM-sourced and Facebook-sourced images. We should note that the Vision Toolbox does not recognize or classify faces, only detects them and provides a bounding box. We should also note that the face detection methods are very coarse and lead to many false positives.

We first load the images into MatLab and use the Image Processing Toolbox to utilize the GPU in resizing all of the images gathered down to 1MB for speed and computational resource purposes. We then use the Vision Toolbox to detect faces in each image and provide a bounding box around each image. Upon detection of faces, we determine the max and min of the face sizes via the size of the bounding box (in pixels) and we discard the faces that are less than 75% the size of the largest face. This is done for two reasons: the first being that typically MatLab's Vision Toolbox will pick up many false faces in the texture of the background, the second being that faces that are smaller than our cutoff are often in the background, resulting in an image that may not be of much use.

We then crop and clean up the faces we have extracted by adjusting contrast and then resizing the image to 180x200 using bicubic anti-aliasing. Lastly, we write out the image to either the test or training database depending on the source of the image. For images coming from the FOSCAM camera, we write out the image to the test database and preserve the filename's timestamp. For images coming from the Facebook API, we write out the image to the training database and change the filename to the name of the user who uploaded the photo. In cases where there are multiple faces in an image, the filename will have a number appended to it corresponding to the order (from left to right) in which the face appeared in the image.

C. Face Recognition and Matching

1) OpenCV Face Recognition:

My first approach at implementing a face identifier was to use the OpenCV library function, FaceRecognizer. OpenCV (Open Computer Vision) is an open source library with functions specifically geared towards computer vision applications.[6] The FaceRecognizer function requires a training database, a test image, and a threshold. At first, it seemed like using this function would allow for a quick implementation of facial identification. However, using OpenCV with MatLab requires a nontrivial setup.[7] Additionally, I found the function to be particularly rigid, only providing a binary recognition scheme rather than a more complex scheme that provides an output such as ”Match”, ”Match with Uncertainty”, ”Not a Match with Uncertainty”, ”Not a Match”.

2) PCA-based Face Recognition:

Being unsatisfied with the OpenCV Face Recognizer, I set out to implement my own PCA-Based face recognizer. Developing a computational model of face recognition is difficult as faces are complex and multidimensional. The aim is to develop a computational model of face recognition that is fast, simple, and accurate in constrained environments (home/office). The PCA approach transforms face images into a small set of characteristic images called eigenfaces.[8] Eigenfaces are principle components of the initial training set of faces and the principle component of a test face. Eigenfaces are calculated by first finding the covariance matrix of the image, then finding the eigenvectors of the covariance matrix. Recognition is performed by projecting a new image onto the ”face space”. The recognition process is done through the following set of steps:

1) *Initialization:* Acquire training set of face images and calculate the eigenfaces (define face

space).

2) *Encounter:* Calculate a set of weights based on the input image and the M eigenfaces by projecting the input image onto each of the eigenfaces.
3) *Determine:* Is the image a face at all? Check if the image is close to the "face space"
4) *Classify:* If it is a face, check the weight pattern if it is a known person or unknown person. This is typically done by finding the eigenface with the lowest Euclidean distance.
5) *Incorporate:* If the same unknown face is seen multiple time, learn and incorporate to known faces.

Typically, this distance to the "face space" is known as the Euclidean distance. When we select a test image, we look for an image in our training database with the lowest Euclidean distance. To provide some "real-world" meaning of the Euclidean distance, I set thresholds for what constitute a match and with how much certainty.

TABLE II
THE FOLLOWING TABLE SHOWS MATCHING CATEGORIES USING THE EUCLIDEAN DISTANCE

Category	**Euclidean profile**
Match	$< 3 * 10^{16}$
Match with Uncertainty	$4 * 10^{16} \geq 3 * 10^{16}$
Not a Match with Uncertainty	$5 * 10^{16} \geq 4 * 10^{16}$
Not a Match	$> 5 * 10^{16}$

These Euclidean profiles were developed heuristically. After running the algorithm many times, I noticed that Euclidean distances typically less than $3 * 10^{16}$ were definite matches, distances typically greater than $5 * 10^{16}$ were definite non-matches, and distances in-between were a toss-up. Rather than being overly optimistic and providing the lowest Euclidean distance match as an output for each photo, or overly pessimistic and rejecting all but perfect matches, I decided that a gradient output would be best as to not mislead the user into thinking that these are guaranteed results.

When the face recognition algorithm is run, the image in the database that the test image was matched to is pulled up and shown next to the test image. Since the training image had its

filename set to the name of a Facebook friend, we can quickly parse the filename and produce an output that includes the name of the person, the match certainty, and its Euclidean distance.

D. Equipment and Setup

1) Image Acquisition Hardware: FOSCAM FI8910W:

Image capture was done by a FOSCAM FI9810W IP camera. The camera was configured with the web-interface enabled and FTP upload to an off-site server. Sensitivity of the motion-detect feature was set to a minimum (1) and the IR ”Night-vision” setting was enabled. All other settings remained stock, including image contrast (5), image brightness (5), and image resolutiuon (640x480).

2) Image Acquisition Hardware: HP Proliant DL365 G1:

The Facebook API was called using open-source cross-platform software (Photograbber) on an HP Proliant DL365 G1 with a gigabit Ethernet connection. The server was maintained in Maxwell Dworkin B121 with a public, static IP that was incremented by one upon a temporary IP ban instated by Facebook’s Graph API. The server was configured with 2x AMD Dual-core Opteron CPU, 8GB of DDR2 ECC Registered RAM, and 4x 250GB 15k SCSI drives configured in RAID 0. The process of harvesting 112 Facebook profiles consumed 32.3 GB of hard disk space, 5% CPU utilization, 20% RAM utilization, and 4.4% Network utilization.

3) Image Processing Hardware: Lenovo ThinkPad W520:

The data was processed using MatLab 2013b 64-bit with the Image Processing and GPU Array toolboxes. The workstation used was a Lenovo ThinkPad W520 running Windows 7 64-

bit with Service Pack 1. The workstation was configured with a Quad-core Intel Core i7, 8GB of DDR3 RAM, a nVidia Quadro 2000m, and a 128GB SATA SSD drive. The process of face detection and face matching averaged to 20% CPU utilization, 30% RAM utilization, and 10% GPU utilization. Network and disk utilization was trivial.

E. Significant Challenges and Pitfalls

1) Unideal Images from FOSCAM IP Camera:

Unfortunately, the FOSCAM IP Camera's maximum resolution for capturing images was 640x480. As a result, a typical face extracted from such an image was 40x40 in size. These faces typically exhibited major artifacting when resized in order to do a comparison with the training database. In addition, the faces also often had shadows or had the subject looking away from the camera, which resulted in uncertain matches. I briefly looked into using homographies in order to reorient the face and to develop a composite face in which I could then compare, but it seemed that without a facial feature detector, this method would not work. I ended up using heuristic methods to change the Euclidean profiles to improve accuracy of matches.

2) Photograbber and the Facebook API:

Facebook has become increasingly restrictive with the use of their APIs in addition to taking measures that prevent overt web scraping. As so, they have begun enabling various API rate limits that severely limit the ability of someone to mine their databases.[9] Although they have not published the exact threshold limit to institute an IP ban for image requests through their API, it seems that the magic number is around 400 for tagged photos per request and about 1200 per hour. In order to overcome this limitation of the API, I had to manually increment my IP address by one once hitting that limit. Later, I noticed that Facebook does not impose these API

limits when tags are not requested. If I simply download all the photos for each friend I have on Facebook, I lose the possibility of making an accurate match without tags, I gain a larger database that improves the chances of a potential match.

As a result, I decided that it would be best to look for tagged photos in the ”Profile Picture” album, the reasoning being is that most people have a picture of themselves in their profile picture and a tag more-or-less should be a reasonable way to decimate the vast number of pictures I would be downloading. For the sake of brevity and the possibility that either my API token would finally be banned or I would run out of available IPs, I downloaded tagged profile pictures until I reached 1000 images containing a face.

3) Facebook's Ever-changing Photo Naming Conventions:

In the past, images harvested from Facebook typically had a naming scheme such that followed something along the lines of ”<album_id>_<user_id>_<photo_id>_n.jpg”. The name of the file could be parsed so that the user ID could be extracted, allowing a developer to send a request to the Graph API and receive the name of the person who owns the photo. That functionality has been removed. Although Facebook has maintained the same filename convention on the surface, it seems that the ID strings no longer return any API response. Additionally, it is not guaranteed that photos from the same album will have the same album ID or user ID even if there is only one album owner/uploader, hinting that Facebook must have an internal database in order to keep track of these IDs. Thus, after harvesting Facebook photos, I had to rename each photo using a Windows batch command along the following name convention: ”<First Name> <Last Name> _(#).jpg”.

F. Participants

Participants were those who entered the image area captured by the FOSCAM IP Camera. During the study, three males aged 22-27 and 1 female aged 20 were captured. All participants

were informed that there is a security camera in the room.

IV. Evaluation and Results

I am proud to state that the the system implemented in this final project has met or surpassed all expectations.

A. *Face detect from FOSCAM IP-sourced Image*

With regards to the time to face detection from the FOSCAM IP-sourced image, on average the system required 2.3 seconds for each photo input regardless of the number of faces in the image. As it turns out, the way Mathworks has implemented their Vision Toolbox makes it especially sensitive to the image size rather than the number of faces in the image. By comparison, a 8MP image taken from an iPhone 5S camera with only my face in it required 90.3 seconds to complete. Interestingly, with two faces in an image taken with the same camera took 74 seconds to complete, which leads me to believe that the speed is correlated to the size of each face rather than the size of the image as a whole.

B. *Harvesting Data from Facebook for Training Database*

Additionally, the average time to harvest Facebook Images was about 30 seconds per Facebook profile. This step had a very high variance, as some Facebook friends had small album sizes (a few friends had exactly 0 photos) and others had album sizes that exceeded 5GB. In future implementations of this system, I may modify Photograbber further in order to omit all but profile photos during the harvesting stage rather than later in the pipeline.

C. *Matching FOSCAM-sourced Face to Facebook-sourced Face*

Lastly, the average time to face match a test image to the Facebook training database was between 3.4 seconds. It seems that in increasing or decreasing the training database size in a

range from 1 image to 1000 images, the change in the time required to make a match is trivial. It seems that MatLab loads the database into RAM rather than using swap, so the speeds still remain fast at least for an experiment of this size. I imagine the time to face match will increase once the MatLab runs out of memory and is required to use swap space.

Additionally, it is important to note that out of the 20 test images of 4 distinct persons, 16 resulted in a ”Match” and 4 resulted in a ”Match with Uncertainty”. Effectively speaking, PCA provided a strong match 80% of the time and an acceptable match 100% of the time. Going through the matches by hand, all 20 photos were matched to the correct person.

TABLE III
THE FOLLOWING TABLE SHOWS A LIST OF CRITERIA TO WHICH THE SYSTEM HAS BEEN EVALUATED FOR SUCCESS.

Criteria	**Target**	**Result**	**Target Met**
Time to detect and harvest face	$\leq$ 10 seconds per image	2.3 seconds per image (average)	Yes
Time to download Facebook photos	$\leq$ 60 seconds per profile	30.0 seconds per profile (average)	Yes
Time to match harvested face to Facebook face	$\leq$ 15 seconds per face	3.4 seconds per image (average)	Yes

V. PRIVACY IMPLICATIONS

A. *Threat Model*

1) Risk:

Without the use of face recognition software, a user must sift through thousands and thousands of images by hand to search for a face to which he can then recognize. Given the fact that this would take a significant amount of time by hand, but not very long via computer, the risk is lowered by using such a system.

2) Exploit Cost:

Without the use of a face recognition system, it may take hours to find a face and then additional time to find the correct identity belonging to that face. In the case of the Tsarnaev brothers, had they been on a flight out of the country the day of the bombing, they would have not been captured. Thus, the exploit cost is medium-high depending on the application that the recognition is being used for.

3) Mitigation Cost:

Since the face recognition system is agnostic in the sense that it records all movements and performs facial recognition on all detected faces in its field of view, it may record some private inter-personal actions that the user may have not wanted recorded. However, it is easy to remove such images assuming there is proper authentication with the FTP server. Thus, the mitigation cost is low.

4) Penalty:

Since the face recognition system is only a reporting system and not an offensive security system (such as one equipped with a turret gun), all the recognition system can do is alert the user that someone, whether matched or unmatched, has entered the field of view of the camera. It is still up to the user to pursue a penalty. Thus, the penalty is low.

5) Efficacy of Solution:

Although there is a low risk and mitigation cost, the exploit cost is relatively high. With a high

exploit cost, the low penalty hampers the total security of the system such that it may not be a sufficient deterrent to prevent an intrusion. Thus, the total efficacy of the solution is medium. A high efficacy solution may come in the form of an automated response in the presence of known targets.

B. Legality

It seems that the root question of legality stems from whether it is legal to record with a camera within your home or public places. In the case of recording within your home, it is legal to record within your home without the consent of the person being recorded. [10] However, it is important to note that audio recording is not included in this definition of legality, only video recording. Since we are only capturing images in locations where there is no ”expectation of privacy” (the participants were notified of the camera), the system implemented as-is operates with full legality. Additionally, it is legal to record public places outside of your home as long as audio is not being recorded in the process. [10]

Another question of legality comes from whether it is legal to use images from Facebook for purposes of mining and creating a training database for facial recognition. The following quote comes from Facebook’s Terms of Services:

> ”When you publish content or information using the Public setting, it means that you are allowing everyone, including people off of Facebook, to access and use that information, and to associate it with you (i.e., your name and profile picture).”[11]

Thus, if images are set to Public, it does not violate their Terms of Services. Such images include profile pictures, so the system implemented as-is does not violate that point in their Terms.

VI. Conclusion

In conclusion, there are three main takeaways from this experiment. Firstly, the FOSCAM IP Cameras resolution is acceptable, but could be better. Unfortunately, the total resolution of image was 640x480 while a typical face extracted was 40x40. Additionally, as mentioned previously, Facebook can provide a great database but it comes with many caveats: only about 400 tagged photos can be downloaded API request and it seems like around 1200 tagged photos can be downloaded per hour before a temporary IP ban is issued. Additionally, although I switched to using strictly profile pictures, many of them do not actually contain faces and even tagged photos may not contain faces. Lastly, PCA proved to be an incredibly powerful and fast for facial recognition. Even in unideal situations, such as when poor resolution, age progression, or facial orientation is a factor, PCA continued to provide accurate matches.

VII. Future Work

In the future, Id like to implement a web interface that visualizes who and when someone enters the field of view of the FOSCAM IP Camera. Additionally, I would like to set up a MMS-based notification system that sends a message to my cell phone with the name of the person entering the field of view with the image captured attached.

VIII. Q&A

The following section is dedicated to answering some questions that have risen as a result of this project.

A. How do you substantiate using the e-value you chose? Can you provide some statistical support beyond "what seemed right"?

Simply said, I don't. There is a long history of computer scientists throwing out magic number thresholds that seem to work best for their data sets. [8] [**?**] I could, in practice, develop a SVM-based machine learning algorithm that learns a threshold in which to define what is a mach and what is not a match, but I did not determine that to be within the scope of this project.

B. You stated that you used Facebook profile pictures as the training database for your face detection algorithm. While there are certainly exceptions, most of these pictures are front-facing pictures of peoples faces, with good lighting and resolution. How does your algorithm fare on a training database that is much less robust? Would the power of PCA-based face detection still be strong enough to accurately identify faces from a much dirtier dataset?

Interestingly, I think PCA would have done better with a database that is less robust. Typically, because most pictures are front-facing, I only have a very small subset of pictures containing the subject looking away. Therefore, most of the time I have to compare an unideal picture to an ideal picture. There may be many situations in which an unideal image may match better.

C. Have you experimented with photo editing software to modify the lighting on pictures after the fact to see if that improves identifying the people?

I do most of my photo editing within MatLab, it would defeat the purpose of this project to use Photoshop and manually edit each photo by hand. That being said, I do fix the contrast and the lighting to the best of my abilities within MatLab. The fact that PCA still makes accurate matches considering the quality of the test images speaks more to the ability of PCA to compare ”faceness” of faces rather than direct image comparisons.

IX. Acknowledgements

I would like to thank the CS105 Teaching Staff and the Harvard School of Engineering and Applied Sciences Undergraduate Teaching Labs for provisioning their guidance, knowledge, experience, and resources for this project.

PCA-Based Face Detection using FOSCAM IP Camera and Facebook

Yasha Iravantchi

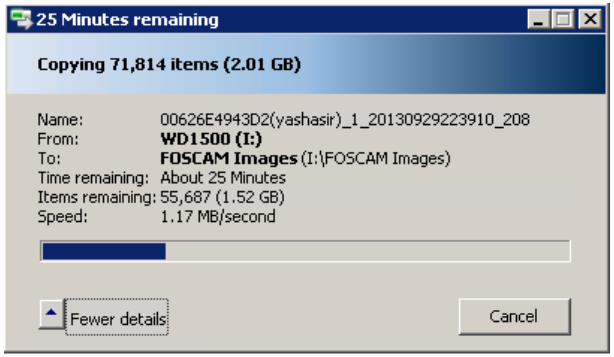
CS 105 Final Project

December 3, 2013

Problem Statement

- Off-the-shelf security cameras do not employ “smart” features
- Features include:
  - Orientation control via Web
  - Motion-detect image capture
  - FTP upload of images
  - IR for night-vision
- Motion-detection can be triggered very easily
- At lowest sensitivity, motion detection mostly triggered by environment changes
- ~900 images/day, 20M images/day if set to video
- In case of emergency situations, such as robbery, this is too much data for a human to manually process
- How do we make this camera “smart”?

FOSCAM FI8910W IP Camera
$50USD



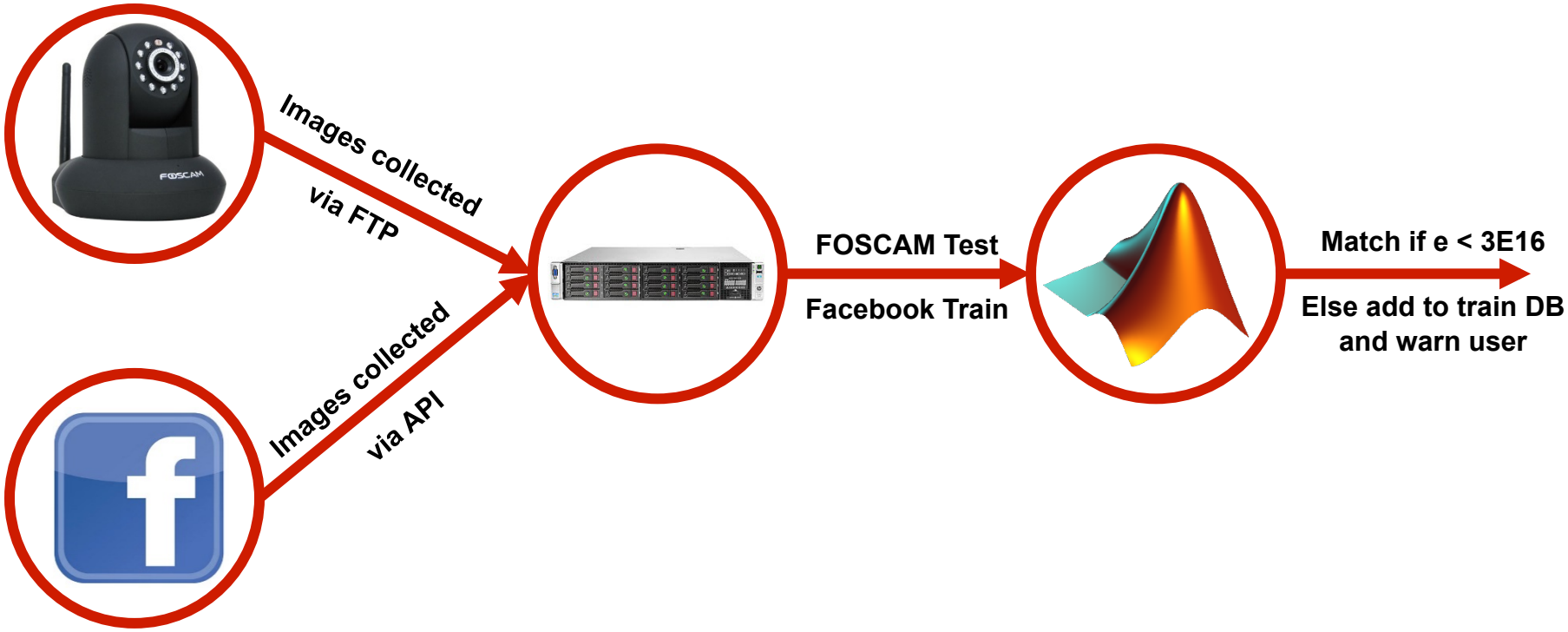
→ FOSCAM IP Cameras do not constitute a security system alone

Implemented Solution

Data Acquisition

Image Processing

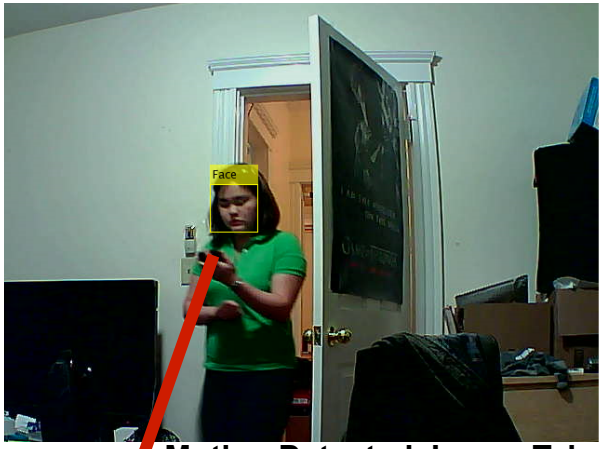
PCA Recognition



→ Facebook provides a large data set of faces with names for face recognition

Conclusions

- FOSCAM IP Cameras' resolution is decent, but could be better
  - Total resolution of image: 640x480
  - Typical face size in pixels: 40x40
- Facebook provides a great database with caveats:
  - Only 400 photo downloads per API request
  - Unknown number of total requests before IP/Token Ban, seems like ~1200/hr
  - Lots of profile pictures that don't contain faces
- PCA is incredibly powerful and fast for facial recognition



Motion Detected, Image Taken

Face Detected In Image

Face Matched to Facebook

→ Facebook+FOSCAM makes for a very powerful (and creepy) security system

Demo

- Face Detect when poor resolution is a factor
  - Subject has shadows on face and is far from camera
- Face Detect when age is a factor
  - Subject is 10 years older in test image compared to training data
- Face Detect when orientation is a factor
  - Subject is looking away from camera

→ Input data is often not ideal…and even when ideal there are issues…

Q&A Slides

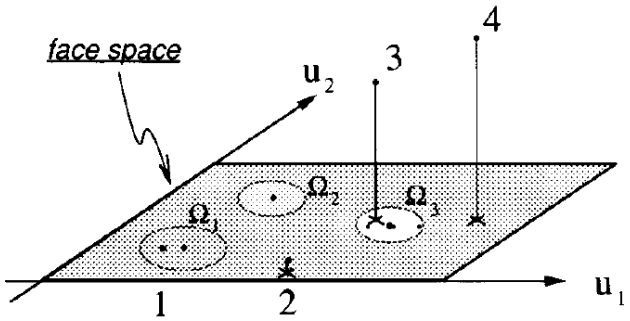
Yasha Iravantchi

CS 105 Final Project

December 3, 2013

Introduction to Eigenfaces

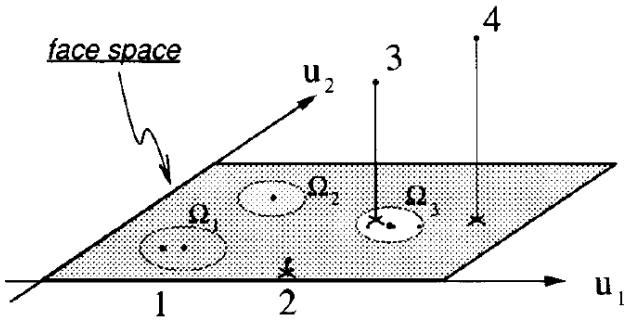
- Developing a computational model of face recognition is difficult.
- Faces are complex, multidimensional
- Aim is to develop a computational model of face recognition that is fast, simple, and accurate in constrained environments (home/office)
- Approach is to transform face image into small set of characteristic images. "eigenfaces"
- Eigenfaces are principle components of the initial training set of faces.
- Recognition is performed by projecting a new image onto the "face space"



→A face can be detected from the local minima of the face map.

Using Eigenfaces for Recognition

- **Recognition Process**
  - **Initialization: Acquire training set of face images and calculate the eigenfaces (define face space)**
  - **Encounter: Calculate a set of weights based on the input image and the M eigenfaces by projecting the input image onto each of the eigenfaces**
  - **Determine: Is the image a face at all? Check if the image is close to the "face space"**
  - **Classify: If it is a face, check the weight pattern if it is a known person or unknown person**
  - **Incorporate: If the same unknown face is seen multiple time, learn and incorporate to known faces**



→ Eigenfaces is basically PCA

Calculating Eigenfaces

- **Find the covariance matrix**

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T \quad (1)$$

- **A is the matrix of vector images with mean zero**
- **C is the covariance**
- **Calculate eigenfaces from the covariance matrix**

(a)

(b)

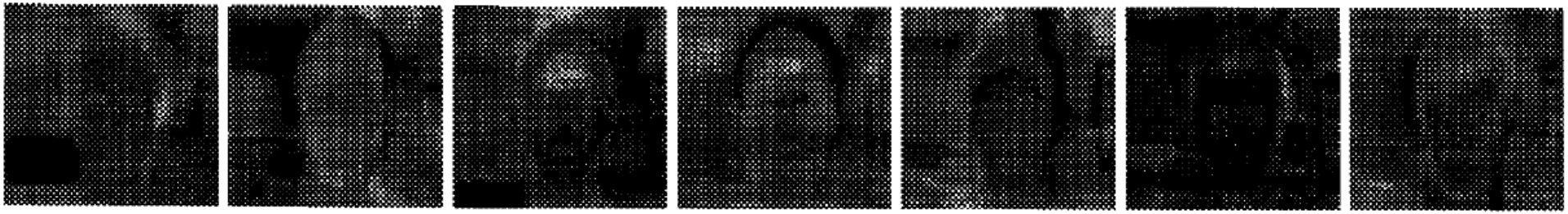


Figure 2: Seven of the eigenfaces calculated from the images of Figure 1, without the background removed.

→ We create eigenfaces based on where the principle components are

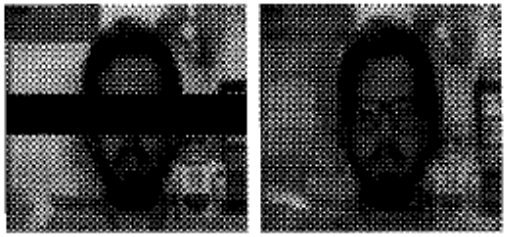
Projecting Onto Face Space

- Faces when projected onto the face space remain faces
- Nonfaces when projected onto the face space look like junk

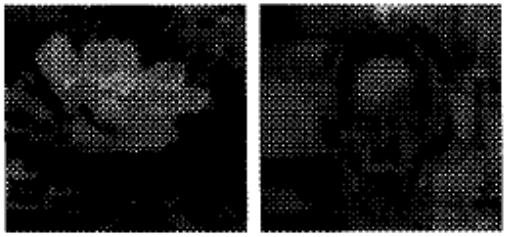
$$\Phi_f = \sum_{i=1}^{i=M'} \omega_k \mathbf{u}_k$$

Projection Weight Eigenface

(a)



(b)



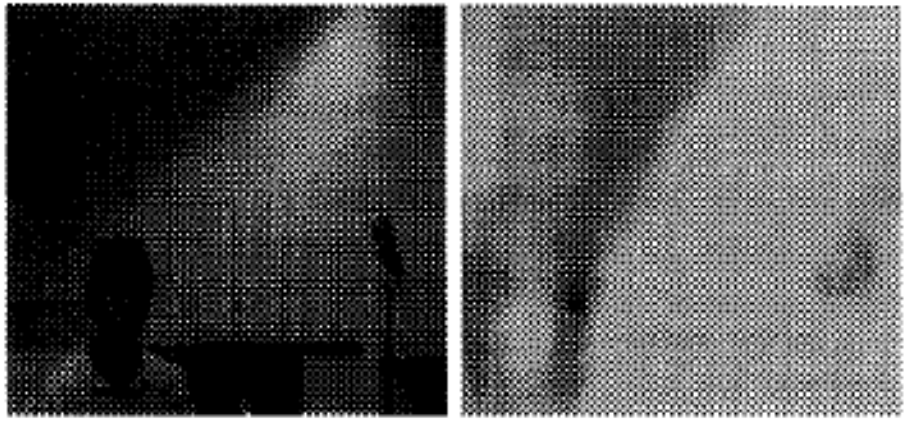
(c)

Figure 3: Three images and their projections onto the face space defined by the eigenfaces of Figure 2.

→ What matters is the "faceness of the face"

Detecting Faces

- Face map is calculated by detecting distance from face-space.
- Distance from face space is converted into 0-255
- Low values translate into dark areas
- Dark areas indicate the presence of a face



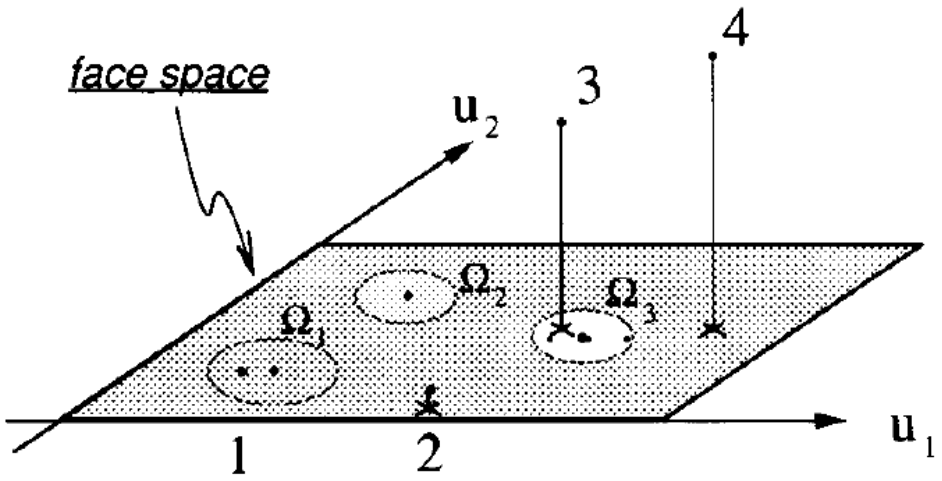
(a) (b)

Figure 4: (a) Original image. (b) Face map, where low values (dark areas) indicate the presence of a face.

→ We can detect faces based on their distance to the face-space

Revisiting Face Space

- Thus there are four possibilities for an input image and its pattern vector:
- (1) near face space and near a face class;
- (2) near face space but not near a known face class;
- (3) distant from face space and near a face class;
- (4) distant from face space and not near a known face class.
- Figure shows these four options for the simple example of two eigenfaces.



→ We can also categorize via face-spaces

Real Time Recognition

- Humans move in normal life
- Check for motion via thresholding
- Head should be slow moving and contiguous
- Head should be smaller blob
- Body is larger blob
- Figure 7 shows an image with the head located, along with the path of the head in the preceding sequence of frames.

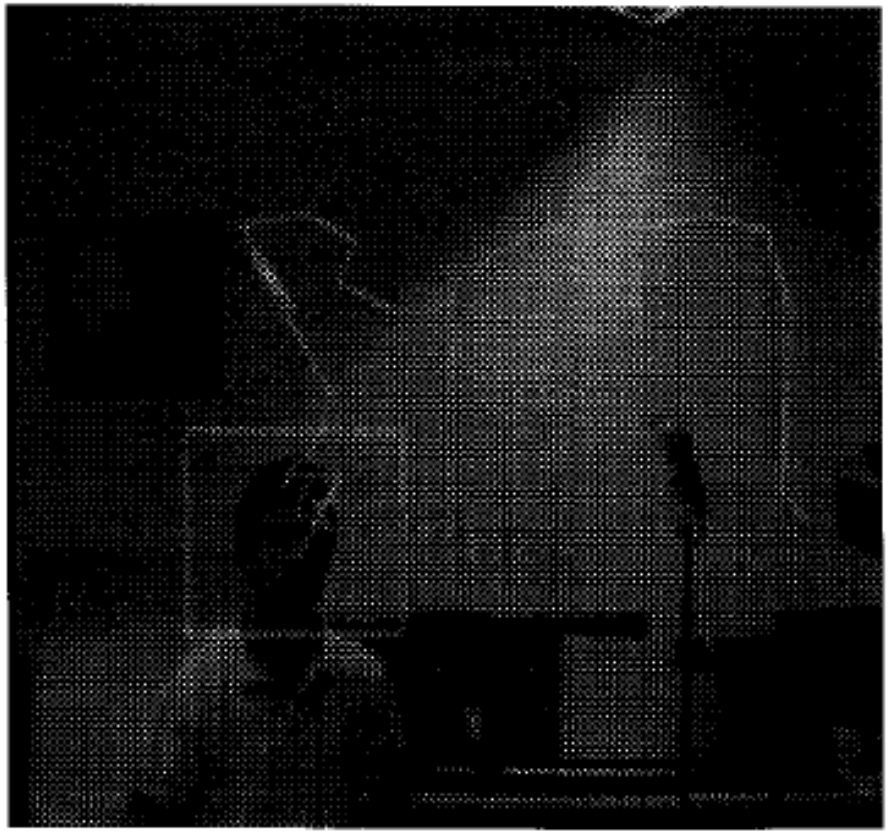


Figure 7: The head has been located — the image in the box is sent to the face recognition process. Also shown is the path of the head tracked over several previous frames.

→ Because eigenfaces is fast, it can do face-detection in real time

REFERENCES

[1] “Fact sheets.” Internet, ID Theft Center, ”http://www.idtheftcenter.org/Fact-Sheets/fs-115.html”.

[2] “Justice department directing full resources to investigate boston marathon bombings.” Internet, Huffington Post, ”http://www.huffingtonpost.com/2013/04/15/justice-department-boston-marathon-bombings]”.

[3] “9/11 commission: Findings.” Internet, Wikipedia, ”http://en.wikipedia.org/wiki/9/11CommissionReportFindings”.

[4] “Facebook may add your profile photo to facial recognition database.” Internet, NBC News, ”http://www.nbcnews.com/technology/facebook-may-add-your-profile-photo-facial-recognition-database-8C11030921”.

[5] “Photograbber.” Internet, Photograbber, ”http://photograbber.org”.

[6] “Opencv.” Internet, OpenCV, ”http://opencv.org”.

[7] “Matlab and opencv.” Internet, Mathworks, ”http://www.mathworks.com/discovery/matlab-opencv.html”.

[8] “Eigenfaces for recognition.” Internet, Pentland and Turk, MIT Media Lab, ”http://www.face-rec.org/algorithms/PCA/jcn.pdf”.

[9] “Facebook api rate limiting.” Internet, Facebook, ”https://developers.facebook.com/docs/reference/ads-api/api-rate-limiting/”.

[10] “Hidden camera laws explained.” Internet, Brickhouse Security, ”http://www.brickhousesecurity.com/category/hidden+cameras/hidden+spy+cam

[11] “Facebook statement of rights and responsibility.” Internet, Facebook, ”https://www.facebook.com/legal/terms/”.