2015

Digital Eye Modification A Countermeasure to Automated Face Recognition

Domenick Poster III

Digital Eye Modification

A Countermeasure to Automated Face Recognition

Domenick Poster III

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science in
Computer Science

Roy Nutter, Ph.D., Chair
Afzel Noore, Ph.D.
Katerina Goseva-Popstojanova, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, WV
2015

Keywords: Face recognition, Face detection, Face alteration, Eye modification

Abstract

Digital Eye Modification
A Countermeasure to Automated Face Recognition

Domenick Poster III

This thesis describes and assesses a series of subtle digital eye modification techniques and their impact on automated face detection and recognition. The techniques involve altering the relative positioning of a person's eyes in a photograph using a variety of horizontal and vertical movements local to the eye regions. Testing with Eigenfaces, Fisherfaces, and Circular Local Binary Pattern face recognition algorithms on a database of 40 subjects and over 4000 modified images shows these subtle geometric changes to the eyes can degrade automated face recognition accuracy by 40% or more. Certain modifications even lower the chance a face is detected at all by about 20%. The combined effect of particular eye modifications resulted in subjects being both detected and recognized less than 20% of time. These results indicate that nearly imperceptible modifications made to one or more key facial features may foil face recognition algorithms.

Acknowledgements

I want to thank my advisor Dr. Nutter for his guidance and feedback throughout my entire graduate career and for giving me the opportunity to work on his projects. I am also in great appreciation of Dr. Noore and Dr. Goseva-Popstojanova for taking time out of their busy schedules to advise my thesis work. In addition, I would like to extend a huge thank you to my colleague Jacob Wolen who greatly helped pave the way for my research.

Finally, I owe a debt of gratitude I can never repay to my parents for supporting me mentally, physically, and financially.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Our identities have long been associated with images of our faces, whether in the form of a portrait, a yearbook picture, a driver's license photograph, or a police mugshot. The task of actually matching a face to an identity has traditionally been the purview of humans. In the last couple decades, however, computers have been programmed to not only detect a human face, but also to learn whose face it is.

Face recognition technology has rapidly become the cornerstone of a diverse array of applications spanning from police surveillance to biometric authentication and social media. We live in a world where our visual appearance has been digitized, linked to our identity, and in some cases made publicly available, often with our own consent. Automated face recognition systems know what we look like and, if given a new, unseen image of a face, can reliably identify whose it is within certain constraints. Furthermore, these algorithms are constantly being made more robust to the demands of real-life scenarios.

1.1 Motivational Scenario

Online programs like Facebook automatically identify individuals using facial recognition. People also have access to photo-editing software allowing for morphing, embellishment, or estrangement of facial features. Yet, most literature on facial recognition seeks to examine the technology's robustness to common challenges such as age, pose, expression, and illumination as opposed to an algorithm's resilience to digital facial modification.

In this study, a new technique to thwart facial recognition of online photographs is introduced and assessed for its effectiveness. The countermeasure developed here addresses the scenario wherein an individual has pictures online associated with his or her identity but have digitally and perhaps subtly modified the facial features.

1.2 Key Terms

1.2.1 Face Detection and Recognition

Subverting facial recognition involves hiding or altering features which make a face distinguishable and unique from others. Face recognition systems have various points of vulnerability to attack. Being successfully identified from a photograph typically requires multiple steps. For the purpose of this thesis, the two main steps are 1) face detection and 2) face recognition.

Face Detection

Face detection is simply the process of finding a face in an image. It is not concerned with determining whose face it is, only that it is a human face.

Face Recognition

Face recognition, however, is the process of determining whether two or more faces are of the same individual. If a face cannot be automatically detected in a photograph, then automated recognition is often impossible.

1.2.2 Feature Extraction

Even though face detection precedes face recognition, they both share some underlying cords. Face detection and recognition algorithms generally rely on extracting a set of (hopefully) discriminating features from a face image [1]. This process is known as feature extraction. The features extracted from the image form the ”facial representation.” There are many different existing facial representations and more continue to be developed. A facial recognition algorithm can partly be categorized by the facial representation it uses. From the widely-accepted taxonomy established by Zhao et al, ”Three types of feature extraction methods can be distinguished: (1) generic methods based

on edges, lines, and curves; (2) feature-template-based methods that are used to detect facial features such as eyes; (3) structural matching methods that take into consideration geometrical constraints on the features" [2].

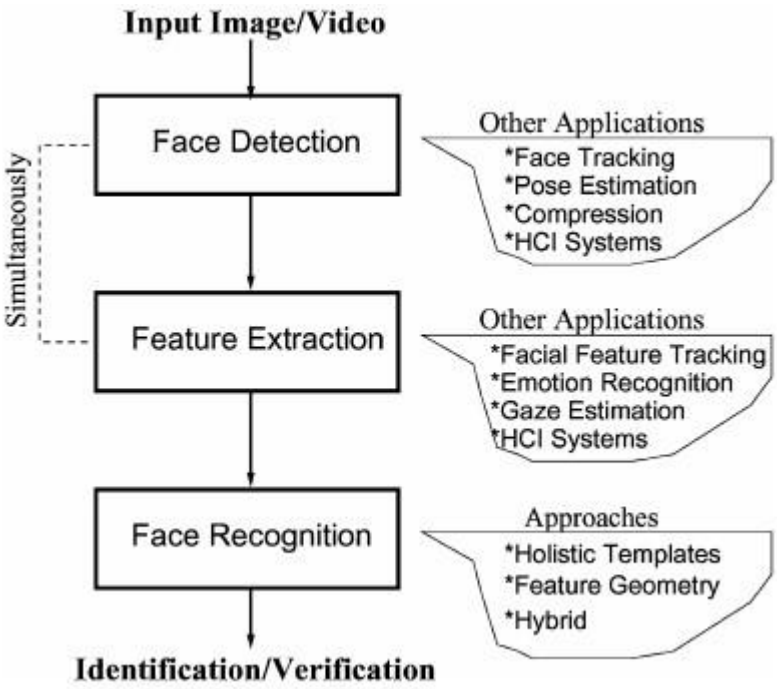


Figure 1.1: High-Level Overview of Face Recognition Systems [2]

When modifying an image, if the features used for face representation can be disturbed, then face detection and recognition should be affected. However, the wide variety of approaches make targeting any one type of feature set an unreliable tactic .

1.2.3 Classification

Classification is the final component of face detection and recognition. In the case of face detection, a classifier uses the feature set to determine whether an image contains a face, or for that matter, eyes, a nose, a mouth, and so on. For face recognition, the classifier attempts to match a face to other faces. Classification can be accomplished by employing a variety of statistical analysis methods, neural networks, or other machine learning techniques.

1.2.4 Approaches to Face Recognition

Since a wide variety of feature extraction and classification techniques can be combined, many different approaches to face recognition exist. To help describe and organize the algorithms, Zhao et al has created the following high-level categorization: holistic matching methods, feature-based (structural) matching methods, and hybrid methods [2]. A holistic method takes the entire face region as its input. Some algorithms which fall under this category include Eigenfaces, Fisherfaces, and Support Vector Machines (SVM). Alternatively, feature-based methods first detect facial features such as the eyes or nose, further perform feature extraction on the separate parts, then feed that information into a classifier. Some popular feature-based methods are pure Geomeotric Feature methods, Hidden Markov Models (HMM) and Convolutional Neural Networks (CNN). Hybrid approaches use some combination of holistic and local feature matching methods.

1.2.5 Measurements of Accuracy

Different metrics can be used to describe an algorithm's effectiveness. Certain measurements are more appropriate for certain tasks.

In face verification trials, in which the task is to determine if a given face image is of a specific person, an algorithm computes a score or confidence of the match. When the confidence is above a certain operating threshold, then it is deemed to be a positive match. Tracking the number of false positives and false negatives of the matches is a popular method [1]. Usually, false positives and false negatives are plotted across varying confidence thresholds, creating the receiver operating characteristic (ROC).

For closed-set identification, in which a face image is matched against a finite number of subjects, a straight-forward approach is to count successful matches versus unsuccessful matches. Another term for this is Rank-1 Accuracy. More formally, Rank-1 Accuracy is the percentage of times the algorithm's first choice is the correct choice. By extension, Rank-*n* accuracy is the percentage of times the correct choice is among the algorithm's top *n* picks. Recognition accuracy plotted over varying values of *n* gives rise to the cumulative match characteristic (CMC). Closed-set identification tests, especially with larger datasets, often use CMC curves as a primary metric of evaluation.

1.3 Digital Modification

Conceivably, a plethora of digital modifications can be applied to an image. Even so, modifications which are extremely obvious or detract from the human recognizability of the photo may not be desirable to share online. Instead, this study employs modifications that are more analogous to alterations produced by plastic surgery.

These modifications would focus on altering the relative spacing, orientation, size and symmetry of key facial features. This thesis is limited to studying the effect of modifying inter-pupil distances and eye symmetry on face recognition and face detection accuracy.

1.4 Problem Statement

The hypothesis is that by changing the geometric arrangement of the eyes, automated face detection and recognition accuracy will be negatively impacted.

1.5 Organization of Thesis

A review of literature on face detection and face recognition relevant to disguise and deception will follow in Chapter 2. Chapter 3 describes the process and tools used to create and test the digital modifications. Chapter 4 discusses the methodologies and results of the experiments. Chapter 5 concludes the findings, notes any threats to the validity of study, and offers recommendations on further research.

Chapter 2

Literature Review

Real-world face recognition scenarios, particularly regarding surveillance, are much more difficult than the highly controlled scenarios in which recognition algorithms are often benchmarked. Review of literature has shown little research examining the effects of deliberate digital modification as a countermeasure to face recognition. Hence, one must look at studies examining similar scenarios to provide some context and inspiration for this work.

2.1 Disguises

Singh et al identified and studied two major challenges to real world recognition scenarios - disguises and limited training data [3]. Disguises are a non-permanent modification made to mask one's identity from both face detection and recognition. If certain types of disguises are effective, then it may be possible to mimic their effect on previously captured photographs.

Singh et al developed a novel approach to address this scenario using a 2d log polar Gabor transform in concert with a dynamic neural network. It is compared against Principal Component Analysis (PCA), Geometric Features, Local Features, Independent Gabor Features, and Local Binary Pattern (LBP) algorithms. The databases used were the AR database (a database containing 3000 images of 116 people) and the National Geographic database (originally containing only 46 images of a single individual) modified to include 15 more individuals each with 10 variations of synthetically generated disguises including glasses, hats, facial hair, varied hairstyles, and makeup. Of all the single disguises, Singh et al found alterations involv-

ing glasses or facial hair to be the most detrimental to accuracy. Testing with the AR dataset where the individual is wearing dark glasses yielded a best-case Rank 1 identification accuracy of 71.7%. PCA's performance was a dismal 28.6%. With the synthetic database, best-case performance increased to 85.2% with glasses while the worst algorithm achieved 70.9% accuracy. Unsurprisingly, a combination of disguises presented the greatest challenge - 71.2% best-case and 19.7% worst-case [3]. A major limitation of this study is the limited size and synthetic nature of the databases.

Makeup as a potential form of disguise is analyzed separately by Eckert et al [4]. In the study, a variety of makeup such as shadow, blush, eyeliner, and lipstick has been applied to the skin, mouth and eyes. The database contained 339 images with 50 reference photos and was manually assembled from makeup tutorial videos. Each image was categorized as either slight, intermediate, or heavy makeup. A Local Binary Pattern algorithm was used to match isolated features such as eyes or mouth and also faces as a whole. The images without makeup were used as the gallery.

Eckert et al found slight and heavy eye makeup to decrease Rank 1 accuracy to about 52% and 45% respectively from a baseline of 65% [4]. However, a spike in accuracy was observed with intermediate level makeup for all features. Eckert et al surmised this was because intermediate makeup "enhances characteristic features and contours, which leads to better distinguishable eye shapes" whereas heavy makeup has an "estranging" effect. Once again, recognition performed on the whole face with multiple modifications resulted in the lowest accuracy of 40%.

One interesting finding was an increase in face recognition accuracy when images with intermediate makeup were used as the gallery. As Eckert et al explain, "intermediate makeup increases both interclass and intraclass variation but the increase is higher for interclass variation therefore the impact is positive" [4]. In effect, the intermediate makeup photo when used as the reference acted as a "bridge" between photos with no makeup and photos with heavy makeup. Therefore, photos digitally modified for the purpose of camouflaging one's identity could actually backfire if that camouflaged photograph is correctly identified through other means and incorporated into a training set.

While wearing glasses or makeup may not necessarily be done to deceive, the act of wearing masks is a far more deliberate tactic. In 2013, Kose and Dugelay studied the vulnerability of face recognition to these physical masks. They found face recognition algorithms, especially 3D recognition algorithms,

to be vulnerable to spoofing mask attacks [5]. For the problem defined in this thesis, these masks are not a realistic strategy for privacy preservation as they cannot be applied to a photograph after it's been taken. In addition, producing a realistic mask is not a trivial process.

Also worth noting is once a photograph has been taken the disguises may be difficult to convincingly overlay onto the face, as some of the synthetically generated databases demonstrated. A great deal of time, effort, and skill would be required to realistically recreate facial hair or add apparel, not to mention the need for commercial-grade photo editing software.

2.2 Plastic Surgery

There is one additional emerging domain within face recognition research which could prove insightful: plastic surgery. As far as could be determined, this area of research was first identified by Singh et al [6] in 2009.

Singh et al created a database from before and after photos found on plastic surgery websites [6]. Using several different feature-based, appearance-based, and texture-based algorithms, they found before and after matching accuracy to be very low (38.8% in the best case). This is partly due to the inherently challenging nature of the manually curated database. The photographs used in the database do not control for pose, expression, makeup, hairstyle, or illumination. This preliminary study concluded certain facial features played an important role in face recognition, particularly nose, chin, and eyes. What they termed 'global surgery' or a full-face lift had a particularly negative effect on accuracy. Depending on the algorithm, Rank 1 accuracy decreased to anywhere from 2.8% to 10.6% for subjects who underwent global surgery [6].

In 2010, Singh et al expanded on the preliminary study with a more thorough investigation involving an augmented database [7]. Unfortunately they had to employ a separate, non-surgery database to establish a baseline accuracy. Nevertheless, they observed similar results to their preliminary study. They found "ear surgery has the lowest effect on the performance" while "nose, chin, eyelids, cheek, lips, and forehead play an important role in face recognition." Accuracies ranged from 18% to 61% depending on the severity of surgery and the algorithm employed [7].

The next step was taken by De Marisco et al in 2011 [8]. De Marisco et al first analyzed the contribution of different facial regions to recognition

performance by hiding regions and noting the change in accuracy. They found that among all the isolated regions, the eye region is the most helpful to recognition.

By modeling the relative importance of each region, De Marisco et al then developed two integrative regions of interest (ROI) analysis methods termed FARO and FACE. Using the aforementioned database assembled by [7], FARO achieved a Rank 1 recognition rate (RR) of 50% for local surgeries and 28% RR on global surgeries. The FACE algorithm, which is more computationally expensive, reached 59% RR for local surgeries and 35% for global surgeries. Comparatively, PCA scored 20% RR for local and global surgeries alike, whereas LDA scored about 35% RR. While their novel algorithms did substantially better than the common alternatives, face recognition on patients of plastic surgery remained challenging and unreliable [8].

Aggarwal et al took a similar approach in 2012 by using a sparse representation (SR) of individual facial regions and integrating the results of each region into a final prediction [9]. Also using the Singh database, their algorithm achieved an overall accuracy of 77.9%, although accuracies for different surgeries are not separately measured. To reiterate, the Singh database has no pre-surgery baseline so a relative drop in accuracy cannot be computed.

Also in 2012 Kose et al conducted a study focused on face recognition robust to nose alterations [10]. They created a synthetic database from images of the Face Recognition Grand Challenge v1.0 database [11] where subjects' nose regions are randomly swapped. This database has the advantage of having a baseline. Their approach was to break a pair of images into "blocks" and only incorporate the corresponding blocks with the most similarity for face recognition. This approach is similar to only analyzing the most informative facial regions. Once the most similar blocks are identified, they utilize PCA, LDA, and Circular Local Binary Pattern (CLBP) algorithms on the individual blocks. Without using the block-based approach, PCA's accuracy on the synthetic database was 31%, having dropped 29% from the baseline, LDA scored 55% with a drop of 20%, and CLBP scored 70% with a drop of 9%. Using only the k most similar blocks, PCA scored 64% with a drop of 18% on the synthetic database, LDA scored 68% with a drop of 14%, and CLBP scored 76% with a drop of 6% [10]. Clearly their approach improved accuracy but the results cannot be compared to the previous studies because of the difference in datasets. Regardless, the commonality of being discriminative in which areas to incorporate into recognition persists.

Bhatt et al [12] developed an approach that divided a photo into multiple

regions or “levels of granularity” and applied a multi-objective evolutionary algorithm (MOE) to determine which regions were the most useful to successful recognition. Using the Singh database, this approach achieved an impressive 87% accuracy. Performance regarding local surgeries related to or around the eyes was similarly high. The lowest performance, 71%, was for patients who underwent a global face lift.

More recently in 2015, De Marisco et al [13] expanded on their earlier research by applying a region-based approach unified by multimodal supervised collaborative architecture called Split Face Architecture (SFA) yielding results similar to [12] except without the need for an extensive training set.

Algorithm	[6]	[7]	[8]	[9]	[12]	[13]
PCA	19%	27%	35%	29%	27%	80%
FDA	20%	31%		33%	31%	64%
GF	28%					
LFA	22%	48%		39%	38%	
LBP	30%					77%
GNN	**34%**	**54%**		54%	54%	
CLBP		48%		48%	48%	
SURF		51%		51%	51%	
FARO			50%			59%
FACE			**70%**			**85%**
LDA			40%			
SR				**78%**		
MOE					**87%**	

Table 2.1: Summary of average Rank-1 Accuracy for all photos in Singh plastic surgery database per algorithm per study

Table 2.1 summarizes the results of all the studies which used the Singh plastic surgery database. These accuracies represent the overall Rank-1 accuracy across all the plastic surgery photos, from local operations to global procedures. The accuracies in bold represent the best-performing algorithm for each study. As the studies have been arranged in chronological order, one can observe an increasing trend in performance over time.

Even though newer approaches have achieved major improvements in accuracy, plastic surgery, especially on multiple key facial features, can still pose a major challenge to common face recognition algorithms. Due to a

lack of alternative plastic surgery image databases, the more recently successful approaches have yet to be thoroughly validated. Also, by their very nature of separately analyzing and incorporating different facial regions, the more innovative algorithms can become very computationally expensive or require intensive training. Nevertheless, undergoing plastic surgery simply to hide from automated face recognition systems is an extreme and uncommon response. Furthermore, plastic surgery does not address the scenario wherein one wishes to camouflage a particular picture which has already been taken.

2.3 Plastic Surgery Simulation Tools

As plastic surgery has become increasingly popular, tools have been created to simulate plastic surgery operations in order to preview the changes. These software tools are primarily targeted towards surgeons and patients. They can be used, however, to modify photographs in a way subversive to automated face recognition.

These tools have the advantage of removing the skill and experience necessary to create convincing modifications using commercial-grade photo editing software. Lee et al developed software which generates 3D models of a face from a photograph [14]. Once the model has been created, pre-programmed operations such as augmentation, cutting, and laceration can be executed on different facial features.

Chou et al has also created a user-friendly plastic surgery simulation tool [15]. In it, there is also support for adding glasses or facial hair. One can swap out facial regions from a photograph with the corresponding regions of a celebrity, for example. In doing so, the program can be used to generate pictures which morph one's face to look like someone else.

These programs are limited to performing viable plastic surgery operations. Since there is no cosmetic surgery to change inter-pupil distance, this cannot be simulated with software specific to plastic surgery. If one's goal is to disguise a photograph, it is not necessary to limit oneself to physically possible modifications.

2.4 Digital Modifications

Little research could be found directly studying the quantitative effects of digital modifications on face recognition accuracy. However, the relevant literature that does exist is insightful.

Newton et al studied automated face recognition from the perspective of privacy and law enforcement [16]. Their goal was to protect individuals' privacy by de-identifying their facial features so that video surveillance images can be shared with police without violating the privacy of innocents. This allows law enforcement to investigate footage without requiring a warrant. Once the suspicious persons are determined, specific warrants can be obtained for those individuals. Simply blacking out faces could hide important information such as pose and expression. Therefore, an approach was needed which preserved some information about the face but made the individual unidentifiable. Newton's solution was to create a new, composite face from the k-nearest similar faces. This approach successfully prevented individuals from being identified by an automated recognition system but also made them unrecognizable by humans [16]. Yet, if an appropriately small number of k similar faces are employed in the face averaging process, a balance could potentially be struck between maintaining human recognizability and degrading automated recognition accuracy. Whether or not the modifications would look realistic is less certain.

Ferrara et al conducted a study in 2013 on the effects to face recognition accuracy of digital modifications to photographs by using plastic surgery simulation software as well as basic geometric transforms on the image [17]. The geometric transforms performed were a barrel distortion, a vertical contraction, and a vertical expansion on the entire image. The geometric alterations were intended to simulate an unintentional warping of the image due to the photo capture device (ie camera) or scanning device. LiftMagic is a free plastic surgery simulation tool that was used to simulate intentional modifications of the face image. Three commercial-grade face recognition algorithms were used: Verilook, Luxand, and a SIFT-based algorithm. They used the AR face database to test the results. Ferrara et al found that barrel distortion had little to no impact on face recognition accuracy for all three of the algorithms. However, for all but Verilook, vertical expansion and contraction of the image resulted in a moderate drop in performance. Face images which underwent several simulated plastic surgery operations resulted in a decrease in performance for all three algorithms [17].

Szegedy et al were actually able to make imperceptible alterations to images in such a way that would cause an image classification neural network to misclassify 100% of the time [18]. For example, an image of a school bus which was normally correctly classified would always become unrecognizable to the neural network once the image was distorted. The study only experimented with images of objects and handwriting. Their distortions relied on being able to probe the neural network and undermine feature detection at the hidden layers [18]. However, without access to the neural network, it may not be possible to analyze and identify the necessary distortions required to trick the network. Someone trying to camouflage their face in a photograph probably does not have the luxury of knowing which algorithms will be used to recognize him or her. An adequate camouflage technique must be effective against a variety of algorithms. Nevertheless, the study by Szegedy et al highlights a worthy avenue for further research.

Chapter 3

Experiment Setup

This Chapter outlines the steps taken to systematically measure the effectiveness of different geometric eye modifications at confounding face recognition technology. A major shortcoming of existing research is the lack of precise quantitative measurements of the strength of different types of facial morphing. For example, plastic surgery related studies only categorize the type of the surgical operation. Along the same lines, [4] broadly categorizes makeup as “light,” ”intermediate,” or “heavy.” The approach taken in this thesis provides a means to measure the magnitude of the eye modifications in order to formulate expectations of impact on face recognition accuracy.

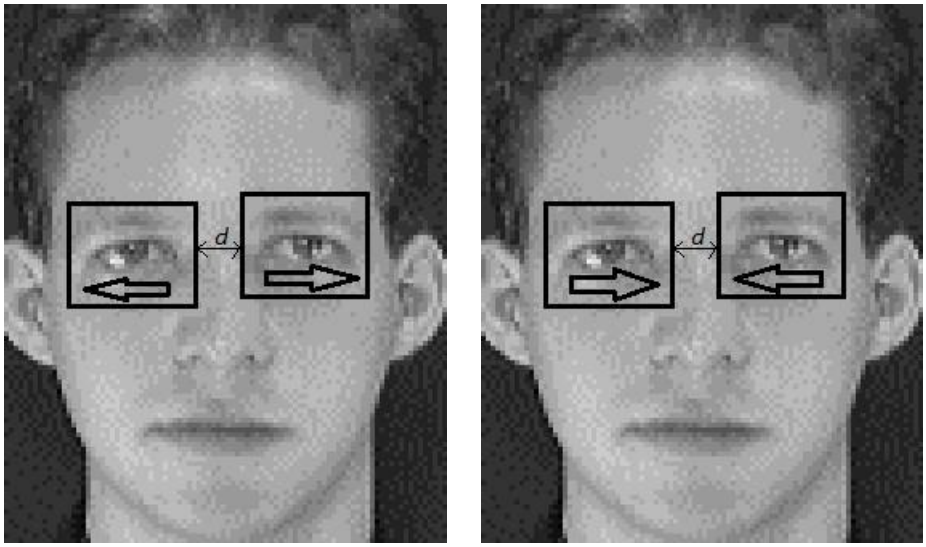
3.1 Modification Technique

In order to create a sufficiently large dataset, image modification could not be done manually. Furthermore, the modifications need to be replicated in a standardized and precise fashion across all subjects. To that end, a combination of tools were used. OpenCV is used for eye detection [19]. ImageMagick, an open-source image editing program, is used to perform the actual morphing of the eye regions.

Finding the eyes in the image is the first step in the process. OpenCV’s feature detection is based off the Viola-Jones algorithm and uses a feature-template-based Haar cascade architecture. OpenCV’s Haar cascade eye classifiers locate the pixel coordinates describing rectangular eye regions. If exactly two eyes are not found, the process is aborted. As a result, not every unmodified image has a corresponding modified counterpart. Eye regions are

manually verified to ensure the correct areas were selected.

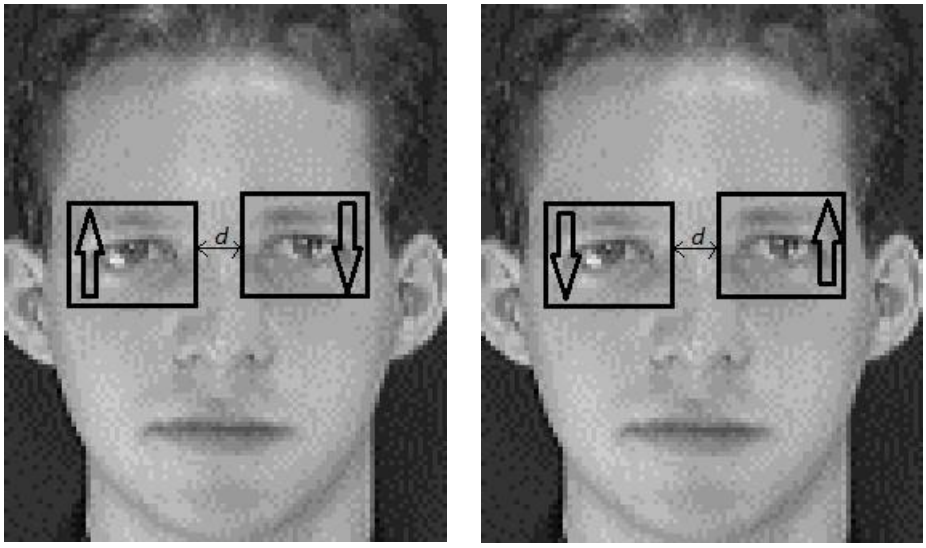
For each image where two eyes are successfully located, a distance d is computed representing the number of pixels between the two eye boxes. At this point, ImageMagick is employed to perform a "Linear Displacement" operation on the pixels within the eye regions. Linear Displacement shifts each pixel in a specified direction by a certain number of pixels. The distance, or magnitude, the pixels are displaced is a function of the previously computed inter-eye distance d and a variable percentage p. The equation for this magnitude (pixel distance) is $m = (p/2) * d$ where p is a value between 10% and 100% on 10% intervals. As an example, a 50% modification would perform a 25% displacement in one eye region and another 25% displacement in the other. Each eye modification is therefore proportionate to the distance between the subject's eye regions. An additional step is taken to make the modified photo appear more realistic. Pixels located near the borders of each eye region are moved at a gradually decreasing fraction of the displacement distance, resulting in a smoothing effect around the modified region. Source code for the eye modification process in included in Appendix B.



(a) Positive Horizontal Displacement (b) Negative Horizontal Displacement

Figure 3.1: Horizontal Displacement Technique

Two similar but opposite horizontal displacement operations are performed: one where the distance between the eyes is increased and another



(a) Positive Vertical Displacement (b) Negative Vertical Displacement

Figure 3.2: Vertical Displacement Technique

where the distance is decreased. The pixels are shifted both outwards from the nose and inwards respectively, expanding and shrinking the amount of space between the eyes.

Vertical displacement is similarly performed by moving the pixels of the eye regions in opposite vertical directions, creating asymmetry. The inter-eye distance *d* is still used as a referential distance for computing vertical displacement. Figures 3.1 and 3.2 illustrate the inter-eye distance *d* and the directional movements of the pixels within the eye region. As can be seen from Figure 3.2, the terms ”positive” and ”negative” in the context of vertical displacement are arbitrary and only used as a convention to distinguish between two mirrored operations.

Bilateral displacement is done by performing a 40% positive vertical displacement paired with horizontal displacements ranging from -100% to +100%. A constant 40% vertical displacement was chosen as it represented a modification within the limits of visual believability as well as to avoid creating massive permutations of modifications.

Figures 3.3, 3.4, and 3.5 on page 17 illustrate the different modifications at varying levels of change. The believability of any given modification is different for each individual and photo but 100% modifications generally look

obviously distorted. However, Figure 3.5 shows the viability of the technique on subjects with glasses. Ultimately, the believability of any modified photo is subjective; users must decide for themselves what level of change is acceptable for the application.

(a) -100% (b) -50% (c) Original (d) +50% (e) +100%

Figure 3.3: Horizontal Displacement Examples

(a) -100% (b) -50% (c) Original (d) +50% (e) +100%

Figure 3.4: Vertical Displacement Examples

(a) -100% (b) -50% (c) Original (d) +50% (e) +100%

Figure 3.5: Bilateral Displacement Examples

3.2 Face Database

This study uses the AT&T Database of Faces courtesy AT&T Laboratories Cambridge [20]. The database contains 40 subjects each with 10 front-facing images.. The images vary with respect to changes in expression as well as slight changes in pose. There is also very slight variation with illumination. The size of each image is 92x112 pixels with 8-bit gray levels. Using the modification techniques mentioned in Section 3.1, we create several synthetic databases. Since successfully detecting the eye regions in the images is a prerequisite for eye modification, the modified databases are a subset of the original database. Of the images where the eye regions were successfully detected, three databases are generated corresponding to horizontal, vertical, and bilateral displacements, designated as DB_{OH}, DB_{OV}, and DB_{OB} respectively. Of the original 400 images, 235 underwent successful eye detection. As such, every degree of eye modification has 235 images, for a total of 4,700 modified images. Since each of these databases contains the same number of modified images, for the sake of simplicity, they are referred to collectively as DB_{O*} in Table 3.1 on page 19.

3.2.1 Normalization

The AT&T Database of Faces represents a "cooperative" face image database, meaning the images are all taken in a consistent, standardized manner. Face recognition is a much easier task in these situations as a great deal of variability is removed from the data. Many real-world scenarios do not have the luxury of the face images being captured in a homogeneous way. "Faces in the wild" is a term used for face images taken in an unconstrained manner. In order to perform face recognition on faces in the wild, a data preprocessing step known as normalization must occur. To simulate the conditions of a real-world example where normalization of photos is necessary, an additional set of databases have been created in which the images have been normalized.

The normalization algorithm was developed by Philipp Wagner and is available in OpenCV's online documentation and included in Appendix B. It utilizes OpenCV to perform a basic normalization technique wherein the face image is cropped, scaled, and rotated so that the eyes are horizontally aligned. To determine the amount of the face to include in the crop, 25% horizontal and vertical offset values were used. The resulting images are re-sized to 70x70 pixels. Photometric normalization (i.e., adjustments to brightness

or contrast) is not necessary as the images are already in gray scale and controlled for illumination. Normalization is performed on all 400 unmodified images as well as all horizontally, vertically, and bilaterally modified images. More formally, DB_{OH} is normalized to produce DB_{NH}, DB_{OV} is normalized to produce DB_{NV}, etc. The normalization process requires successful detection of both the face and the eyes. Consequently, the normalized databases are smaller than their non-normalized counterparts, as shown in Table 3.1. See Figures 3.6, 3.7, and 3.8 on page 20 for examples of normalization. The process of normalization is not perfect. Figure 3.8d on shows an example where the mouth was presumably classified as an eye. Both unmodified and modified images suffer from this misclassification. These mistakes are kept in the database in order to judge the impact of eye modifications.

Modification	DB_{O*}	DB_{NH}	DB_{NV}	DB_{NB}
0%	400	229	229	229
+10%	235	145	146	140
+20%	235	141	144	136
+30%	235	136	143	133
+40%	235	128	143	125
+50%	235	116	137	120
+60%	235	114	134	111
+70%	235	110	128	102
+80%	235	107	121	99
+90%	235	102	113	97
+100%	235	82	102	80
-10%	235	160	154	151
-20%	235	166	157	155
-30%	235	163	147	160
-40%	235	167	141	168
-50%	235	167	137	169
-60%	235	176	139	171
-70%	235	177	130	170
-80%	235	175	127	173
-90%	235	175	126	172
-100%	235	172	112	165
Total Mods	4700	3108	2910	2926

Table 3.1: Number of images for each modification and database

(a) Original (a) Normalized Original (c) Normalized -50% Mod (d) Normalized -100% Mod

Figure 3.6: Original vs Normalized: Horizontal Displacement

(a) Original (b) Normalized Original (c) Normalized -80% Mod (d) Normalized +80% Mod

Figure 3.7: Original vs Normalized: Vertical Displacement

(a) Original (b) Normalized Original (c) Normalized -100% Mod (d) Normalized +100% Mod

Figure 3.8: Original vs Normalized: Bilateral Displacement

3.3 OpenCV Face Recognition Algorithms

This study relies on the OpenCV suite of face recognition algorithms. The three algorithms included in the OpenCV library are Eigenfaces, Fisherfaces, and Local Binary Pattern Histograms (LBPH). Unfortunately, as the literature review indicates, there are many other types of algorithms which are not tested in this study. Although the algorithms packaged with OpenCV are

not as sophisticated as expensive commercial-grade products, they are still popular and commonly used for benchmarking purposes. They also hone in on different features in an image and represent different statistical approaches to face recognition. Eigenfaces, Fisherfaces and LBPH act as different facial representations. Classification is performed using Euclidean distances and the 1st-Nearest Neighbor.

Philipp Wagner, whose face recognition library was merged into OpenCV as of version 2.4, has also written his own implementations of Eigenfaces, Fisherfaces, and LBPH [21]. His implementations have the advantage of exposing the top *n* predicted classes and Euclidean distances, whereas OpenCV only returns the information of the 1st-Nearest Neighbor. As such, his algorithms are used when calculating Rank-*n* Accuracy and CMC curves. For reference, a comparison of OpenCV and Wagner's algorithms is included in Appendix A Figure A.3.

3.3.1 Eigenfaces and Fisherfaces

Eigenfaces and Fisherfaces are the applications of Principal Component Analysis (PCA) and Linear Discriminate Analysis (LDA), respectively, to the domain of face recognition. In face recognition literature, PCA and LDA are often used interchangeably with Eigenfaces and Fisherfaces. LDA is also sometimes referred to as Fisher Discriminant Analysis (FDA). Both Eigenfaces and Fisherfaces take holistic, appearance-based approaches [1]. In simpler terms, an "appearance-based" approach is one whose chosen features are the intensity values of the pixels; a "holistic" approach extracts these features from the entire image.

The main difference between Eigenfaces and Fisherfaces derives from the fundamental difference between PCA and LDA. PCA attempts to identify the "principal components" responsible for the most variation among all the images without concern for class (in this case, the identity of the subject) [1]. Alternatively, LDA builds from principal components but adds class information. In doing so, LDA maximizes interclass variance (the differences between subjects) while minimizing intraclass variance (the uniqueness of an individual's face) [1]. In this way, LDA works to mitigate non-discriminating natural variances such as illumination and expression.

3.3.2 Local Binary Pattern Histograms

Alternatively, LBPH encodes regional, or ”local”, information such as lines, edges, and corners and is said to be texture or generic-based [2][1]. This is done by inspecting the intensity value of a central pixel and comparing it with the intensity values of all the pixels in a surrounding neighborhood of a predefined size. The original LBP operator looks at a square *x* by *y* region around the central pixel. All the pixels within the sampling region are then encoded as a 0 or 1 based on meeting a threshold difference in value [1].

This operator has been enhanced by using a circular neighborhood instead of a square region, coined a Circular Local Binary Pattern (CLBP) [1]. A radius is chosen for the circle, establishing the size of the neighborhood. A number of sampling points are chosen along its circumference. These points assume the intensity values of the pixels at those coordinates. Points along the circumference which do not exactly correspond with a pixel coordinate have their values interpolated from its surroundings [1]. OpenCV implements CLPB with a linear interpolation strategy [19].

The binary encoding of each subregion then corresponds to a ”texture primitive” such as an edge, a corner, or a spot. Histograms are then built based on the numbers of each texture primitive and used to compare face images. As Jain summarizes, ”The success of LBP in face description is due to the discriminative power and computational simplicity of the operator, and its robustness to monotonic gray scale changes caused by, for example, illumination variations. The use of histograms as features also makes the LBP approach robust to face misalignment and pose variations. [1].”

3.4 Overview of Experiments

Using the aforementioned images and algorithms, four distinct experiments were designed to measure the disruptiveness of the eye modifications on face detection and recognition. First, a measure is needed of the eye modifications’ impact in constrained and cooperative face recognition scenarios. To address this, Experiment 1 scores Rank-1 accuracy for each of the three face recognition algorithms on the original and modified images prior to undergoing normalization.

Experiment 2 analyzes the performance of the face and eye detection algorithms employed during the normalization process. By doing so, face

and eye detection can be evaluated separately from face recognition.

Experiment 3 reassesses the face recognition algorithms on the normalized images using both Rank-1 and Rank-*n* accuracy. Using the Rank-*n* accuracies, CMC curves are graphed and discussed. Additional analysis is provided by combining these face recognition results with the results from Experiment 2 to yield a holistic, start-to-finish performance assessment. Overall, Experiment 3 simulates a more unconstrained, real-world face recognition scenario.

To anticipate the event of the face recognition system having access to positively identified modified images, a final experiment is conducted. Experiment 4 studies the change in face recognition Rank-1 accuracy when using modified images for the training data instead of unmodified images.

All accuracy plots are generated using the matplotlib Python library [22]. See Figure 3.9 on page 24 for a visual summary of the processes involved in creating the modified and normalized image data. This diagram also indicates from where each experiment derives its data.

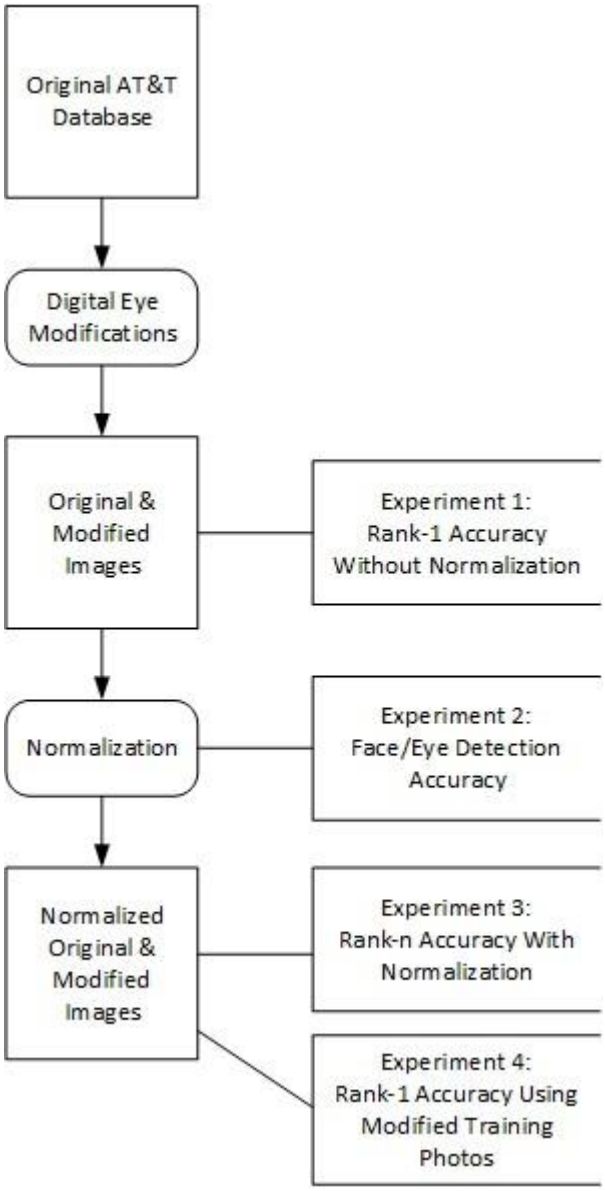


Figure 3.9: Transformation and Experimentation of Image Data

Chapter 4

Results and Analysis

The main goal of this research was to ascertain the effectiveness of the eye modification techniques in undermining the accuracy of automated face recognition systems. Using the data and technology described above, four experiments were designed to evaluate different aspects of the problem space.

4.1 Experiment 1: Without Normalization

The first experiment utilizes the non-normalized set of original and modified images, DB_{O*}. This set includes the database of horizontally displaced images (DB_{OH}), vertically displaced images (DB_{OV}), and bilaterally displaced images (DB_{OB}). DB_{O*} represents photos in highly controlled and cooperative face recognition scenarios. Examples of this domain include driver's license, passport, or mugshot photos.

The training images, or "gallery," are taken entirely from the original, unmodified AT&T database. All 40 classes (subjects) are represented in the training set. Both modified and unmodified images are used as the testing images, or "probe" images. A 10-fold cross-validation scheme with random sampling is used to validate the results. Each subject contributes one image to the gallery for each fold. Additionally, the folds are created in such a way that no modified image is ever tested on a training set containing the corresponding unmodified version of that image. Similarly, no unmodified images are ever tested on a fold containing their exact duplicate. The unmodified probe images establish a baseline Rank-1 Accuracy. Rank-1 Accuracies are then computed for each degree of modification.

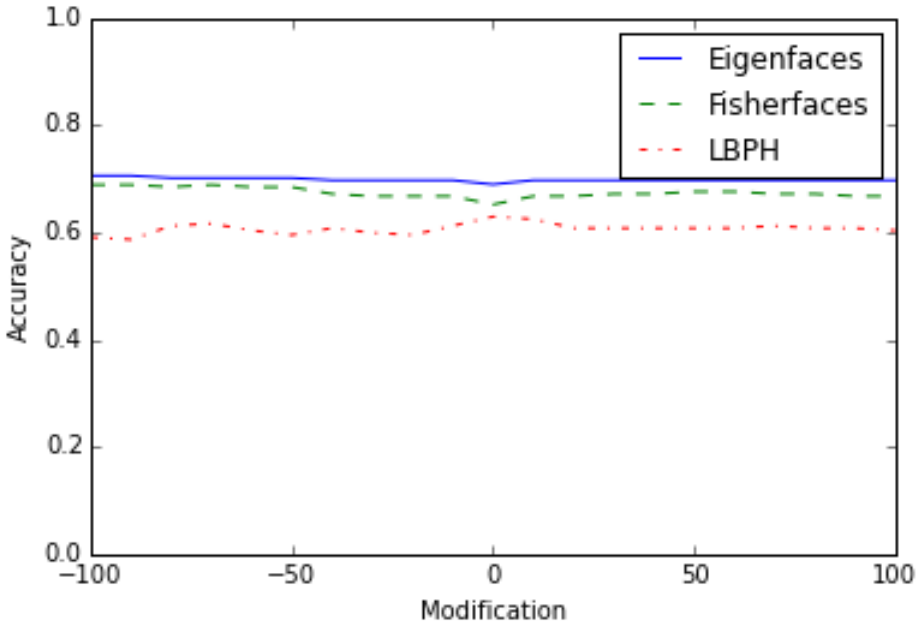


Figure 4.1: Bilateral Modification Rank-1 Accuracy (1 Training Image per Subject)

The results of this experiment are clear. Simple eye modifications are not enough to confound face recognition when image collection is undergone in a controlled environment. Even at the extreme levels of modification, Rank-1 accuracies remain almost completely unphased for every type of directional displacement. Figure 4.1 showcases this fact for bilateral modifications. Exact accuracies are included in Appendix A in Table A.1. The results for horizontal and vertical displacement are nearly identical and are included in Appendix A as Figures A.1 and A.2. Since a fully cooperative dataset is unlikely in online face recognition scenarios, these results should not be disparaging from a social media privacy perspective.

4.2 Experiment 2: Face & Eye Detection

As emphasized in Section 1.2.1, if a face cannot be correctly detected in a photograph, then face recognition becomes a major issue. In the case of the

constrained face recognition scenario with a cooperative face database, some algorithms can get away with simply assuming a face is there and performing classification on the holistic features of the image (as done in Experiment 1). More real-world scenarios and sophisticated algorithms do not have that luxury. A face, and often specific facial features, must be detected in order for recognition to work appropriately.

If privacy is the ultimate goal, then the impact of eye modifications on face detection accuracy is also a vital concern. After all, it is difficult to identify someone who is invisible.

Since face and eye detection both occur during the process used to create the normalized set of original and modified images (DB_{N*}), the rate of successful feature detection can be tracked. The resulting number of normalized unmodified photos can be compared with the number of normalized modified photos to measure the effect of varying magnitudes of modification on feature detection.

Of the original 400 unmodified images from the original AT&T database, 229 underwent successful face and eye detection, yielding a baseline accuracy of 57.25%. This is compared to the detection rates for the 235 modified images. Face and eye detection rates were not separately analyzed. Figure 4.2 on page 28 shows the change in detection rates on the modified images.

The relative drop in accuracy around the baseline is likely due to the fact that the 235 modified images only exist because eye detection was already successful. On the other hand, the baseline represents the detection rate for all 400 unmodified images. Baseline accuracy would be much higher if only the original images which led to successful eye modification were used to calculate the baseline detection rate. Therefore, the comparison with the baseline is not entirely fair due to selection bias favoring detection of modified photos.

Nevertheless, the results are still informative. Positive and negative vertical displacements from DB_{NV} reduced detection rates equally. This is expected as these are essentially mirrored operations.

More interestingly, positive horizontal displacement (increases in inter-pupil distance) were more likely to trick the Haar cascade classifiers into believing no faces or eyes were there. This is likely due to geometric and structural constraints ingrained in the feature-template-based Viola Jones algorithm. A similar success rate carries over into DB_{NB}, as its main variation was horizontal displacement.

The same is not true for negative horizontal displacements. The classifiers

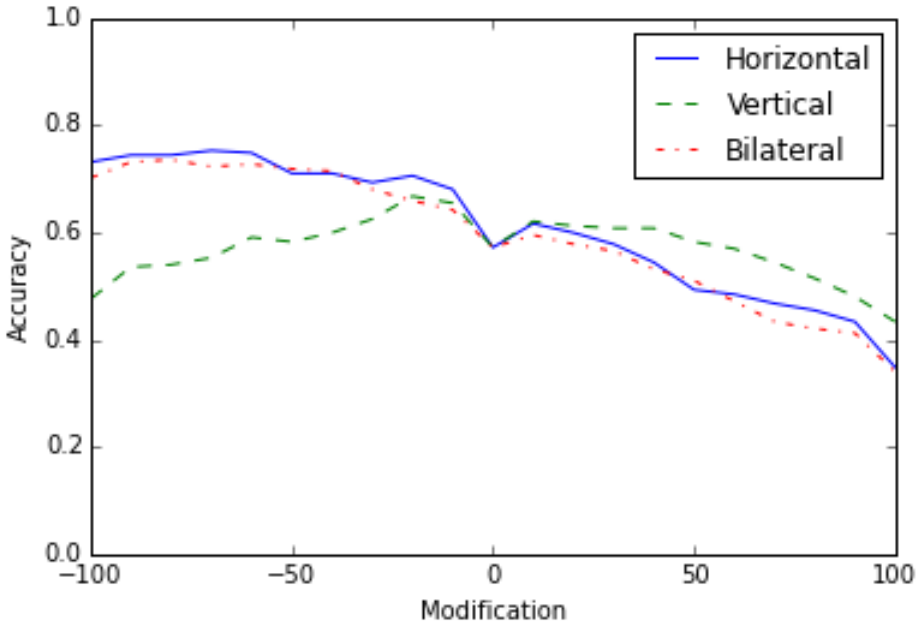


Figure 4.2: Face & Eye Detection Accuracy

are actually more likely to detect the face and eyes for negative horizontal and bilateral. In fact, as the modifications increase in magnitude from -10% to -100%, it actually becomes easier to distinguish the face and eyes.

4.3 Experiment 3: With Normalization

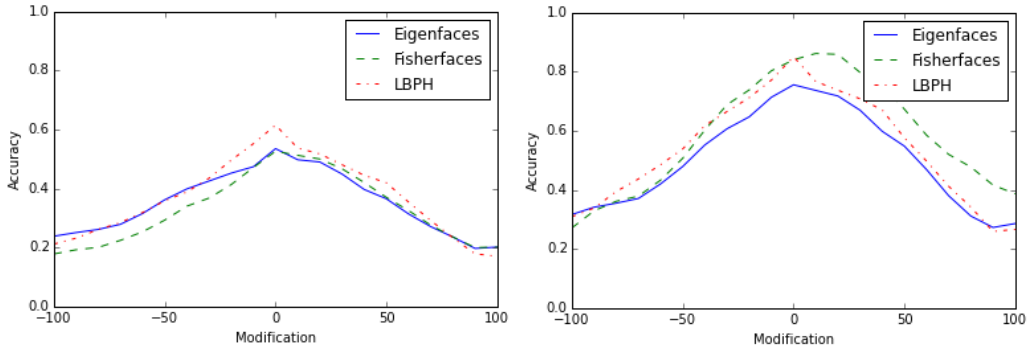
The third experiment operates on the normalized set of original and modified images (DB_{N*}). In this dataset are the horizontally displaced images (DB_{NH}), vertically displaced images (DB_{NV}), and bilaterally displaced images (DB_{NB}). The collection of normalized images approximates the photo variability found in unconstrained face recognition scenarios. The intention is to simulate photos collected from online sources or casually captured in person. This experiment more closely resembles the motivational scenario of protecting the privacy of online identities as presented in Section 1.1.

Training and testing is performed in the same manner as Experiment 1. However, as the initial results for this experiment were more interesting,

the experiment was expanded to vary the number of photos each subject contributes to the gallery. Training sizes of one and three images per subject are tested. If a subject does not have three images to provide for training, it will provide as many as are available. Only 38 subjects have enough normalized images to be part of the experiment.

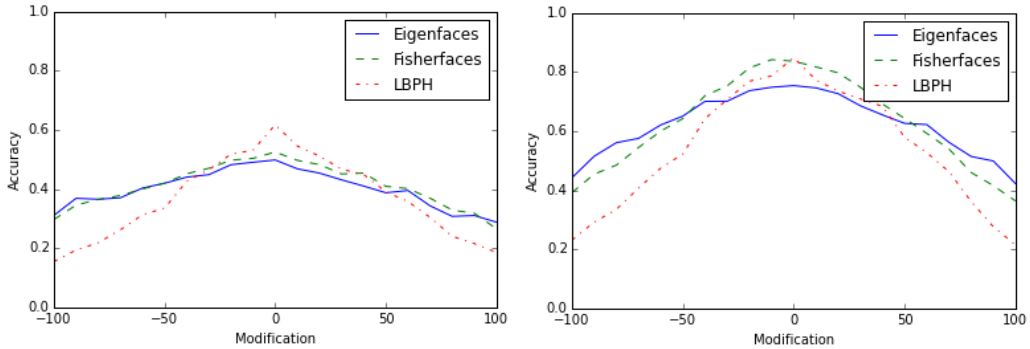
The impact of eye modification on face recognition is compounded by the image normalization process, specifically with regards to face cropping and eye alignment. Geometric features such as eye locations and inter-eye distance are inputs into the normalization algorithm. Referring back to Figure 3.6 on page 20, as the eyes move closer together, the face cropping becomes tighter. This is also displayed by contrasting Figures 3.8b and 3.8c on page 20. Another phenomenon occurs with vertical eye displacement. By introducing asymmetry, attempting to align the eyes generates improperly rotated images. This effect is exhibited on page 20 in Figure 3.7. This kind of variability in the images acts directly against the purpose of normalization. In turn, face recognition becomes more difficult.

Figures 4.3, 4.4, and 4.5 on page 30 show the performance of the three algorithms on the normalized image sets. Horizontal, vertical, and bilateral displacements generally share similar trends in accuracy. Additional training images boost overall accuracy, but give way to steeper declines in performance. Vertical eye modifications have the least singular impact on accuracy. The algorithms are more robust to positive horizontal displacements than negative. Bilateral modifications produce the sharpest drop-offs in accuracy, building credence for the strategy of ”the more modifications, the better.” Of all the modifications, negative bilateral displacement is the most detrimental. At -50% bilateral modification, Rank-1 accuracies are between 42% and 46% compared to a baseline success of 75% to 83%. Exact Rank-1 accuracies for the experiment can be found in Appendix A.



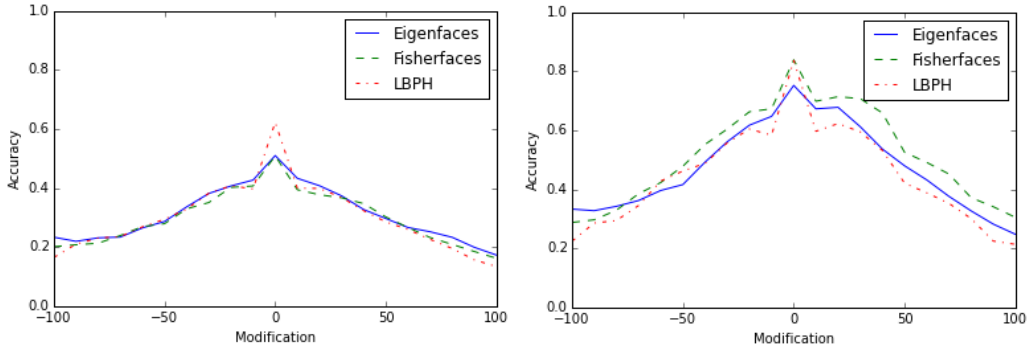
(a) 1 Training Image Per Subject (b) 3 Training Images Per Subject

Figure 4.3: Normalized Horizontal Rank-1 Accuracy



(a) 1 Training Image Per Subject (b) 3 Training Images Per Subject

Figure 4.4: Normalized Vertical Rank-1 Accuracy



(a) 1 Training Image Per Subject (b) 3 Training Images Per Subject

Figure 4.5: Normalized Bilateral Rank-1 Accuracy

While LBPH performs consistently well on unmodified photos, the algorithm is especially vulnerable to modifications, especially vertical displacements. This is surprising as the algorithm has been shown to be robust to face misalignment [1]. Less surprising is how LDA benefits the most from additional training images given its incorporation of class information.

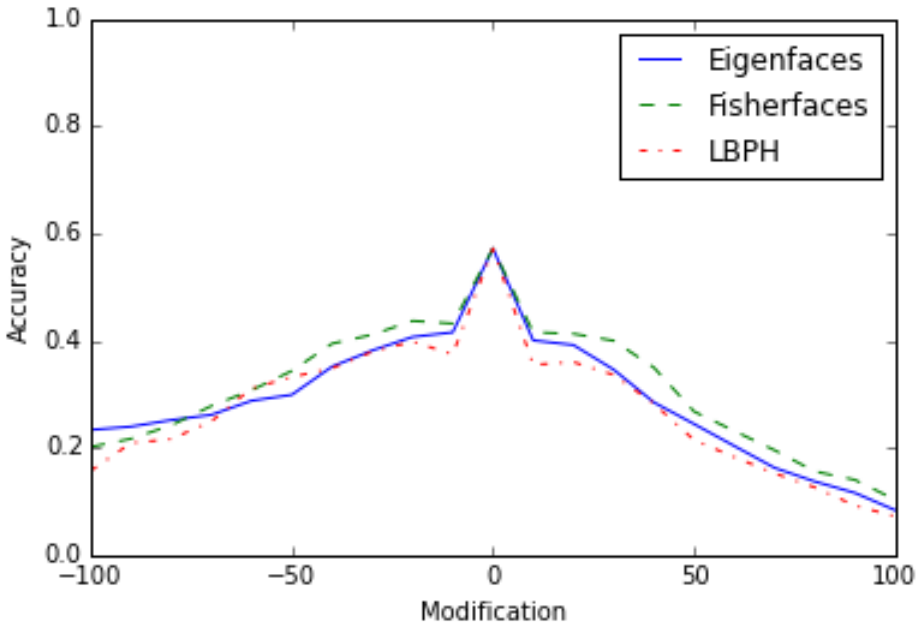


Figure 4.6: Combined Face/Eye Detection & Face Recognition Accuracy of Bilaterally Modified Images

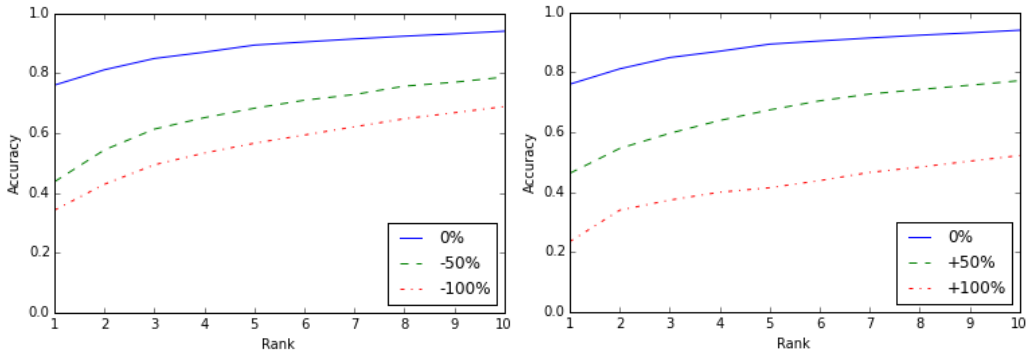
To reiterate, face recognition is typically impossible if a face cannot be found in an image. Since the primary concern is to assess the modifications' ability to prevent an individual from being identified, it is helpful to look at the face recognition process as a whole. Using Experiment 2's results on face detection accuracy, it is possible to make a prediction of the likelihood someone is both detected and recognized. If $P(D)$ is the probability of successful face and eye detection and $P(R)$ is the probability of successful recognition, then $P(D) * P(R)$ is simply the chance that both necessary steps succeed. Figure 4.6 is the plot of these probabilities. The poor performance of the Haar cascade classifiers in feature detection result

in an overall end-to-end baseline recognition accuracy of about 60%. The chance of both detecting and recognizing the face of a subject is about 30% with a -50% bilateral eye displacement and about 20% for a +50% bilateral eye displacement. Even though negative bilateral displacement lowered face recognition accuracy more than positive bilateral displacement, positive bilateral displacements are a stronger overall countermeasure when factoring in the significant impact positive horizontal modifications have on feature detection.

Face recognition systems are often evaluated beyond the accuracy of just their top prediction. Rank-*n* accuracy reveals if the individual in the test image is among the top *n* results. The more suspects a face recognition system is willing to provide, the greater the chance that the disguised individual is among them. In order to judge the countermeasure's robustness to an exhaustive search, CMC curves for Rank 1 through Rank 10 are calculated for all three algorithms using the DB_{NB} image set. Three training images are used per subject. These accuracies are calculated using Wagner's face recognition library [21]. Figures 4.7, 4.8, and 4.9 on page 33 chart the CMC curves for moderate and extreme negative and positive bilateral modifications for each algorithm.

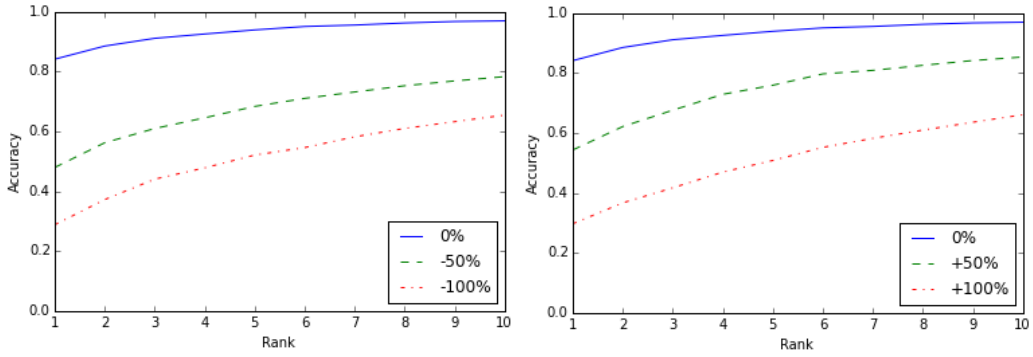
Moderate modifications (+/-50%) gradually fail to hide the individual as more ranks are inspected. Nevertheless, the moderate modifications still reduce accuracy by 20% to 30% from the baseline at each rank for PCA (Eigenfaces) and LDA (Fisherfaces). LBPH is more robust to the eye modifications at higher ranks. However, it is important to note that Wagner's LBPH implementation was shown to outperform OpenCV's (see Figure A.3), thus explaining the higher LBPH performance seen here.

Generally, it is much harder to go unnoticed in a database of 40 subjects compared to a database of hundreds or thousands of individuals. However, Rank-10 accuracy is only about 50% and 65% when using PCA and LDA respectively on photos with positive 100% bilateral modifications compared to 95% and 99% baseline performance. If one is prepared to sacrifice some believability of the modified photo, he or she is rewarded with greater anonymity.



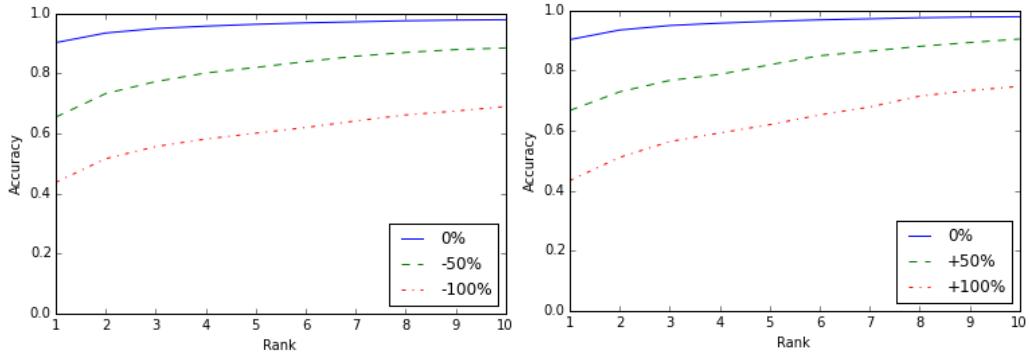
(a) Negative Displacements (b) Positive Displacements

Figure 4.7: Rank-*n* Accuracy using PCA on DB_{NB} (3 Training Images)



(a) Negative Displacements (b) Positive Displacements

Figure 4.8: Rank-*n* Accuracy using LDA on DB_{NB} (3 Training Images)



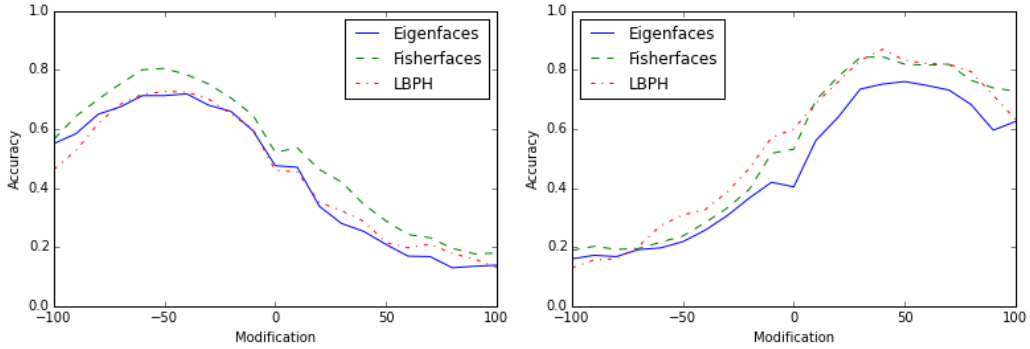
(a) Negative Displacements (b) Positive Displacements

Figure 4.9: Rank-*n* Accuracy using LBPH on DB_{NB} (3 Training Images)

4.4 Experiment 4: Modified Training Photos

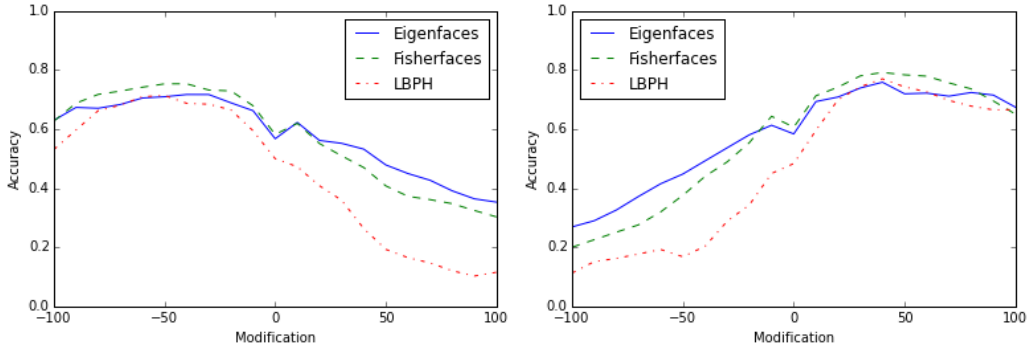
Experiment 4 covers the possibility of the automated face recognition system having access to a set of modified and correctly identified face images. This experiment resembles the experiment performed by Eckert et al [4] where faces with intermediate levels of makeup were used as the gallery. Another possibility giving rise to this same scenario is if face recognition systems anticipate an eye modification countermeasure by building their own database of modified images to use in training. In [4], an overall increase in accuracy was observed due to the ”enhancing” effect of the makeup on facial features. While the eye modifications performed in this thesis should not serve to enhance the distinguishability of subjects, it is a possibility worth exploring. This scenario is assessed by conducting an experiment similar to Experiment 3 in methodology but instead training on images modified by +/-50% and using three training images per subject.

Training on images which underwent a mid-range magnitude of modification allowed the algorithms to accurately recognize the subjects in probe images undergoing a similar modification (see Figures 4.10, 4.11, and 4.12 on page 35). However, accuracies plummeted to 20% or below on images with the extreme opposite level of modification. If this technique was deployed to anticipate and counter the digital disguises developed in this thesis, success would depend on being able to guess the correct configuration of modifications. More precisely, four configurations of eye displacement modifications would have to be correctly guessed. In face recognition trials where the modified training image was incorrectly configured, accuracies would likely be worse than training on an unmodified image.



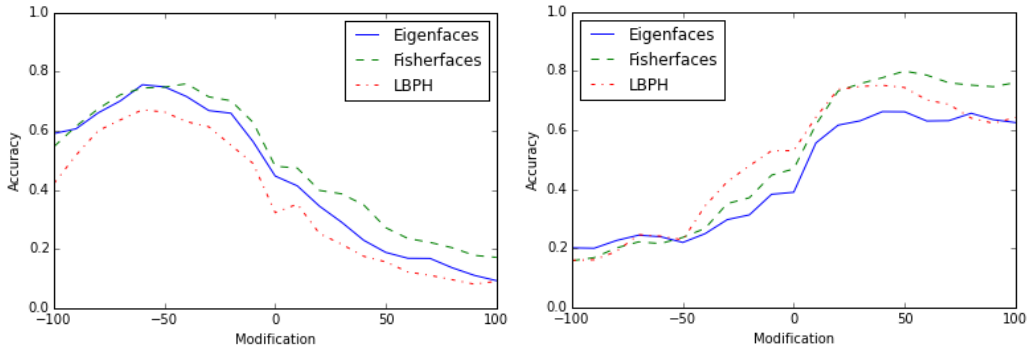
(a) Training on -50% Modifications (b) Training on +50% Modifications

Figure 4.10: Rank-1 Accuracy with Horizontally Modified Training Images



(a) Training on -50% Modifications (b) Training on +50% Modifications

Figure 4.11: Rank-1 Accuracy with Vertically Modified Training Images



(a) Training on -50% Modifications (b) Training on +50% Modifications

Figure 4.12: Rank-1 Accuracy with Bilaterally Modified Training Images

Another possibility would be to train on all potential dimensions and magnitudes of displacement. Unfortunately, this is not explicitly tested in this thesis. Given the trends, however, one might expect recognition rates to become averaged out as the varied modifications simply add noise to the data. Since many types of facial feature modifications could be developed beyond just alterations to eye locations, anticipating all possible facial modification configurations could quickly become impractical.

Chapter 5

Conclusion

5.1 Summary

In this thesis, a digital modification technique applied to the eye regions of a face image is assessed as a countermeasure to automated face recognition. Experimental results show face recognition on images captured in a controlled and cooperative environment achieves high performance despite any modifications. Yet, images found on the Internet have much more variation. Accurate recognition of these images requires proper feature detection and image normalization strategies. This research shows face recognition in this context to be far more susceptible to the variabilities introduced by digital modifications.

While face recognition accuracy at the extremes of modification is desirably low, many of the resulting images appear unrealistic. However, modifications in a more moderate range are believable and still significantly impact accuracy. Also important to point out is the negative effect modifications have on facial feature detection. Subtle modifications to the eyes (especially increasing the space between a subject's eyes) caused many images to fail face and eye detection, rendering recognition impossible. The overall chance that an individual is both detected and recognized was lower than 20% on images where the eyes had been spread apart both horizontally and vertically, compared to a baseline of 60%.

Using the relatively small AT&T database of 40 subjects, moderate magnitude countermeasures show some weakness when viewed against Rank-*n* accuracy as *n* approaches 10. But, even in these cases, the modification

technique still had a noticeable impact on accuracy when compared with the baseline. Therefore, this technique alone may still be effective in real-world scenarios involving massive face image databases.

5.2 Threats to Validity

The major threat to the validity of this thesis is its generalizability to a more real-world scenario. The AT&T database, while popular, is relatively small and highly cooperative with consistent illumination and image resolutions. Another drawback to the work herein is that only three face recognition algorithms and one normalization technique are tested. It is difficult to separate the effect the modifications have on proper image normalization from their impact on face recognition. Finally, the modifications performed on this database may not translate well to images of different resolutions. Even though the modifications are designed to be a function of image resolution and inter-eye distance, without further experimentation one cannot be sure the same degree of modification will translate similarly to photos of different sizes and qualities.

5.3 Future Work

Further research needs to address the threats to validity mentioned above. Experimentation on a larger database of faces in the wild would more accurately reflect a real-world scenario. The modifications should be tested on a broader variety of face recognition algorithms, including commercial-grade systems. Along the same lines, more sophisticated image normalization and facial feature detection algorithms need to be tested. Experiments involving Facebook or other social media sites employing face recognition technology could also present a realistic case study.

While this thesis analyzes countermeasure performance from a closed-set identification perspective, it should also be viewed as a 1:1 or 1:n verification problem. Verification tests using confidence thresholds, false accept rates, false reject rates, and ROC curves could prove insightful. Even social experiments where human observers attempt to discern whether an image appears legitimate or modified would help to objectively bound acceptable degrees of modification.

Despite these potential threats to validity, the results of this project encourage further exploration. The literature indicates multiple modifications to the entire face can significantly affect face recognition performance. A plethora of modifications should be developed and analyzed. The modifications could target various features such as the forehead, cheeks, nose, mouth, or chin. A variety of operations could be performed on these features to change their size, shape, orientation, and relative distance. By deploying a full salvo of modifications, even sophisticated face recognition systems may be confounded.

Bibliography

[1] A. K. Jain and S. Z. Li, *Handbook of Face Recognition*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[2] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, “Face recognition: A literature survey,” *ACM Comput. Surv.*, vol. 35, pp. 399–458, Dec. 2003.

[3] R. Singh, M. Vatsa, and A. Noore, “Face recognition with disguise and single gallery images,” vol. 27, no. 3, pp. 245–257, 2009.

[4] M.-L. Eckert, N. Kose, and J.-L. Dugelay, “Facial cosmetics database and impact analysis on automatic face recognition,” in *2013 IEEE 15th International Workshop on Multimedia Signal Processing*, pp. 434–439, IEEE, 2013.

[5] N. Kose and J.-L. Dugelay, “On the vulnerability of face recognition systems to spoofing mask attacks,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2357–2361, IEEE, 2013.

[6] R. Singh, M. Vatsa, and A. Noore, “Effect of plastic surgery on face recognition: A preliminary study,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 72–77, IEEE, 2009.

[7] M. D. Marisco, M. Nappi, D. Riccio, and H. Weschler, “Plastic surgery: A new dimension to face recognition,” in *IEEE Transactions on Information Forensics and Security*, vol. 5, pp. 441–448, IEEE, 2010.

[8] M. D. Marisco, M. Nappi, D. Riccio, and H. Weschler, “Robust face recognition after plastic surgery using local region analysis,” pp. 191–200, 2011.

[9] G. Aggarwal, S. Biswas, P. Flynn, and K. Bowyer, “A sparse representation approach to face matching across plastic surgery,” in *2012 IEEE Workshop on Applications of Computer Vision*, pp. 113–119, IEEE, 2012.

[10] N. Kose, N. Erdogamus, and J.-L. Dugelay, “Block based face recognition approach robust to nose alterations,” in *2012 IEEE Fifth International Conference on Biometrics: Theory, Applications and Systems*, pp. 121–126, IEEE, 2012.

[11] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, “Overview of the face recognition grand challenge,” in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) - Volume 1 - Volume 01*, CVPR ’05, (Washington, DC, USA), pp. 947–954, IEEE Computer Society, 2005.

[12] H. Bhatt, S. Bharadwaj, R. Singh, and M. Vatsa, “Recognizing surgically altered face images using multiobjective evolutionary algorithm,” in *IEEE Transactions on Information Forensics and Security*, vol. 8, pp. 89–100, IEEE, 2013.

[13] M. D. Marisco, M. Nappi, D. Riccio, and H. Weschler, “Robust face recognition after plastic surgery using region-based approaches,” vol. 48, no. 4, pp. 1261–1276, 2015.

[14] T.-Y. Lee, Y.-N. Sun, Y.-C. Lin, L. Lin, and C. Lee, “Three-dimensional facial model reconstruction and plastic surgery simulation,” in *IEEE Transactions on Information Technology in Biomedicine*, vol. 3, pp. 214–220, IEEE, 1999.

[15] J.-K. Chou, C.-K. Yang, and S.-D. Gong, “Face-off: automatic alteration of facial features,” vol. 56, no. 3, pp. 569–596, 2012.

[16] E. Newton, L. Sweeney, and B. Malin, “Preserving privacy by de-identifying face images,” in *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 232–243, IEEE, 2005.

[17] M. Ferrara, A. Franco, D. Maltoni, and Y. Sun, “On the impact of alterations on face photo recognition accuracy,” pp. 743–751, 2013.

[18] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.

[19] G. Bradski *Dr. Dobb’s Journal of Software Tools*.

[20] F. S. Samaria and A. C. Harter, “Parameterisation of a stochastic model for human face identification,” in *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pp. 138–142, IEEE, 1994.

[21] P. Wagner, “facerec.” https://github.com/bytefish/facerec.git, 2015.

[22] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

Appendix A

Additional Data

A.1 Experiment 1

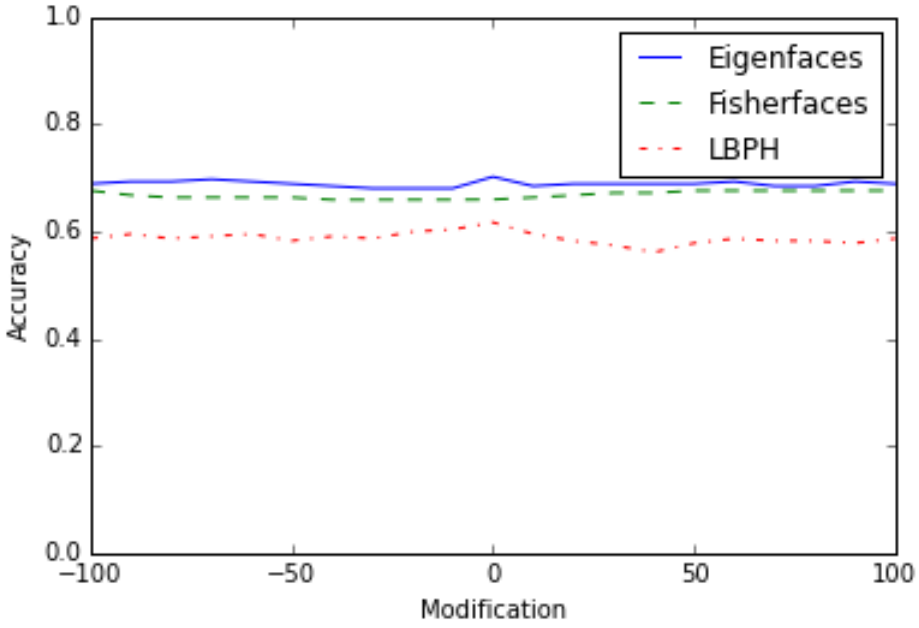


Figure A.1: Horizontal Modification Rank-1 Accuracy of Non-Normalized Images (1 Training Image per Subject)

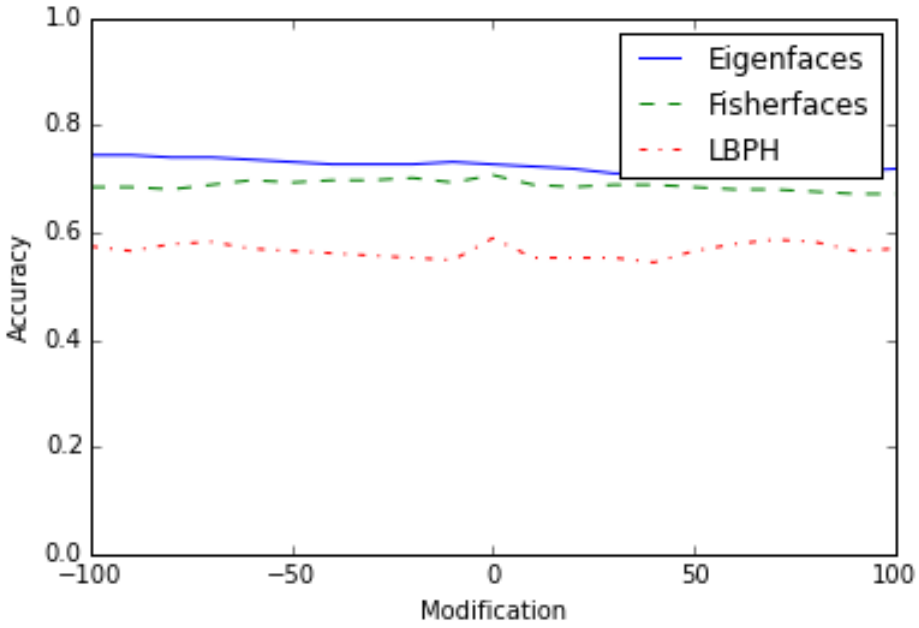
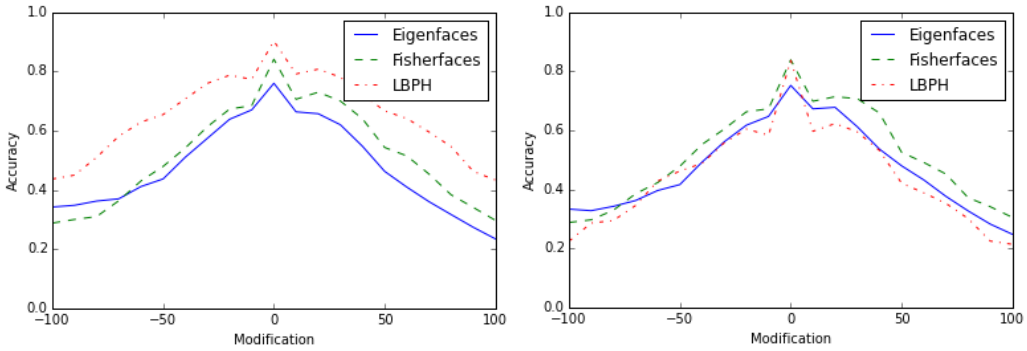


Figure A.2: Vertical Modification Rank-1 Accuracy of Non-Normalized Images (1 Training Image per Subject)

Modification	PCA	LDA	LBPH
-100%	.706	.689	.591
-90%	.706	.689	.587
-80%	.702	.685	.613
-70%	.702	.689	.617
-60%	.702	.685	.604
-50%	.702	.685	.596
-40%	.698	.672	.609
-30%	.698	.668	.6
-20%	.698	.668	.596
-10%	.698	.668	.613
0%	.69	.652	.63
+10%	.698	.668	.626
+20%	.698	.668	.609
+30%	.698	.672	.609
+40%	.698	.672	.609
+50%	.698	.677	.609
+60%	.698	.677	.609
+70%	.698	.672	.613
+80%	.698	.672	.609
+90%	.698	.668	.609
+100%	.698	.668	.604

Table A.1: Bilateral Modification Rank 1 Accuracy of Non-Normalized Images (1 Training Image per Subject)

A.2 Experiment 3



(a) Wagner (b) OpenCV

Figure A.3: OpenCV/Wagner Performance Comparison (Bilateral Modification Rank 1 Accuracy Normalized Images 3 Training Images per Subject)

Modification	PCA	LDA	LBPH
-100%	.239	.179	.212
-90%	.251	.193	.232
-80%	.262	.201	.262
-70%	.279	.225	.285
-60%	.316	.254	.319
-50%	.362	.293	.359
-40%	.399	.341	.389
-30%	.426	.368	.436
-20%	.453	.414	.5
-10%	.474	.472	.553
0%	.536	.527	.615
+10%	.498	.513	.538
+20%	.49	.501	.518
+30%	.45	.467	.481
+40%	.397	.421	.445
+50%	.366	.371	.42
+60%	.315	.326	.355
+70%	.271	.277	.292
+80%	.236	.236	.234
+90%	.197	.2	.179
+100%	.202	.202	.171

Table A.2: Horizontal Modification Rank 1 Accuracy of Normalized Images (1 Training Image per Subject)

Modification	PCA	LDA	LBPH
-100%	.317	.272	.31
-90%	.342	.33	.339
-80%	.356	.362	.395
-70%	.371	.379	.438
-60%	.421	.434	.486
-50%	.48	.508	.539
-40%	.553	.603	.618
-30%	.607	.689	.666
-20%	.648	.739	.712
-10%	.714	.803	.772
0%	.756	.84	.851
+10%	.737	.862	.767
+20%	.718	.859	.738
+30%	.67	.798	.709
+40%	.599	.754	.671
+50%	.549	.673	.577
+60%	.469	.586	.498
+70%	.381	.52	.413
+80%	.312	.478	.342
+90%	.273	.416	.26
+100%	.287	.387	.267

Table A.3: Horizontal Modification Rank 1 Accuracy of Normalized Images (3 Training Images per Subject)

Modification	PCA	LDA	LBPH
-100%	.341	.29	.156
-90%	.39	.35	.194
-80%	.416	.363	.219
-70%	.429	.373	.262
-60%	.445	.418	.314
-50%	.458	.438	.336
-40%	.495	.454	.427
-30%	.499	.473	.466
-20%	.513	.514	.518
-10%	.521	.526	.532
0%	.542	.513	.618
+10%	.506	.504	.546
+20%	.501	.476	.512
+30%	.485	.46	.468
+40%	.459	.458	.45
+50%	.431	.423	.391
+60%	.431	.409	.358
+70%	.381	.362	.305
+80%	.347	.326	.241
+90%	.351	.326	.216
+100%	.335	.284	.186

Table A.4: Vertical Modification Rank 1 Accuracy of Normalized Images (1 Training Image per Subject)

Modification	PCA	LDA	LBPH
-100%	.443	.393	.232
-90%	.515	.455	.293
-80%	.561	.485	.336
-70%	.576	.545	.408
-60%	.621	.6	.475
-50%	.651	.643	.523
-40%	.701	.72	.643
-30%	.702	.753	.707
-20%	.737	.814	.769
-10%	.749	.842	.787
0%	.754	.37	.849
+10%	.747	.818	.773
+20%	.727	.798	.736
+30%	.686	.749	.709
+40%	.655	.692	.684
+50%	.627	.643	.58
+60%	.623	.593	.528
+70%	.563	.54	.463
+80%	.515	.460	.362
+90%	.499	.416	.277
+100%	.423	.364	.216

Table A.5: Vertical Modification Rank 1 Accuracy of Normalized Images (3 Training Images per Subject)

Modification	PCA	LDA	LBPH
-100%	.234	.202	.165
-90%	.22	.208	.211
-80%	.231	.214	.231
-70%	.234	.24	.236
-60%	.265	.27	.269
-50%	.287	.28	.295
-40%	.337	.33	.329
-30%	.382	.351	.383
-20%	.407	.403	.408
-10%	.428	.408	.398
0%	.51	.509	.623
+10%	.433	.395	.4
+20%	.408	.378	.399
+30%	.374	.367	.368
+40%	.328	.348	.32
+50%	.297	.303	.286
+60%	.267	.267	.26
+70%	.252	.233	.23
+80%	.234	.21	.196
+90%	.19	.186	.157
+100%	.173	.163	.134

Table A.6: Bilateral Modification Rank 1 Accuracy of Normalized Images (1 Training Image per Subject)

Modification	PCA	LDA	LBPH
-100%	.334	.288	.225
-90%	.328	.297	.287
-80%	.343	.33	.295
-70%	.363	.385	.346
-60%	.397	.424	.429
-50%	.42	.479	.463
-40%	.493	.553	.488
-30%	.562	.605	.558
-20%	.618	.663	.605
-10%	.648	.673	.584
0%	.753	.839	.837
+10%	.673	.699	.597
+20%	.678	.714	.623
+30%	.613	.708	.595
+40%	.537	.659	.532
+50%	.48	.526	.421
+60%	.432	.492	.389
+70%	.377	.452	.353
+80%	.328	.372	.302
+90%	.283	.342	.225
+100%	.248	.305	.214

Table A.7: Bilateral Modification Rank 1 Accuracy of Normalized Images (3 Training Images per Subject)

A.3 Experiment 4

Modification	PCA	LDA	LBPH
-100%	.591	.547	.422
-90%	.607	.616	.518
-80%	.661	.675	.601
-70%	.701	.722	.637
-60%	.756	.745	.671
-50%	.749	.748	.664
-40%	.716	.76	.631
-30%	.669	.715	.614
-20%	.66	.702	.552
-10%	.564	.629	.49
0%	.447	.481	.323
+10%	.414	.474	.352
+20%	.345	.399	.253
+30%	.291	.387	.217
+40%	.23	.35	.176
+50%	.189	.272	.157
+60%	.169	.236	.123
+70%	.169	.223	.111
+80%	.137	.205	.097
+90%	.111	.179	.082
+100%	.093	.172	.091

Table A.8: Rank-1 Accuracy of Bilateral Modifications when Training on -50% Bilaterally Modified Images (3 Training Images per Subject)

Modification	PCA	LDA	LBPH
-100%	.205	.172	.174
-90%	.206	.174	.18
-80%	.232	.208	.199
-70%	.248	.225	.26
-60%	.244	.221	.252
-50%	.226	.221	.236
-40%	.255	.257	.335
-30%	.307	.345	.419
-20%	.319	.383	.461
-10%	.394	.458	.532
0%	.405	.474	.535
+10%	.576	.621	.652
+20%	.646	.727	.746
+30%	.649	.75	.757
+40%	.669	.78	.759
+50%	.666	.788	.749
+60%	.635	.767	.715
+70%	.644	.737	.701
+80%	.647	.748	.655
+90%	.641	.743	.64
+100%	.633	.716	.655

Table A.9: Rank-1 Accuracy of Bilateral Modifications when Training on +50% Bilaterally Modified Images (3 Training Images per Subject)

Appendix B

Source Code

B.1 Batch Eye Modifier

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 11 11:46:20 2015

@author: Domenick Poster
"""

import os
import cv2
import re

wd = r"C:¥Users¥user¥Documents¥School¥Research¥poster"
originals = r"C:¥Users¥user¥Documents¥School¥Research¥
    poster¥att-faces"
maps = r"C:¥Users¥user¥Documents¥School¥Research¥poster
    ¥displace-maps"
composites = r"C:¥Users¥user¥Documents¥School¥Research¥
    poster¥composites"
eye-classifier1-fp = r"E:¥opencv¥build¥share¥OpenCV¥
    haarcascades¥haarcascade-eye.xml"
eye-classifier2-fp = r"E:¥opencv¥build¥share¥OpenCV¥
    haarcascades¥haarcascade-eye-tree-eyeglasses.xml"
```

```
eye_classifier1 = cv2.CascadeClassifier(
   eye_classifier1_fp)
eye_classifier2 = cv2.CascadeClassifier(
   eye_classifier2_fp)
regex = re.compile(r"(¥d+)¥.pgm")
os.chdir(wd)


if not os.path.exists(maps):
    os.mkdir(maps)
if not os.path.exists(composites):
    os.mkdir(composites)

def createMap(map_fp, eyes, original_image):
    height, width, depth = original_image.shape
    blur = ((height + width) / 2) * 0.03
    eye1 = str(eyes[0][0]) + "," + str(eyes[0][1]) + "
       " + str(eyes[0][0] + eyes[0][2]) + "," + str(
       eyes[0][1] + eyes[0][3])
    eye2 = str(eyes[1][0]) + "," + str(eyes[1][1]) + "
       " + str(eyes[1][0] + eyes[1][2]) + "," + str(
       eyes[1][1] + eyes[1][3])
    cmd = "convert -size " + str(width) + "x" + str(
       height) + """ xc:gray50 -fill white -draw "
       rectangle """ + eye1 + """" -fill black -draw "
       rectangle """ + eye2 + """" -blur 0x""" + str(
       blur) + """  """ + map_fp
    os.system(cmd)

def createHorizontalComposite(original_fp, map_fp,
   composite_image_fp, displace):
    cmd = "composite " + map_fp + " " + original_fp + "
        -displace " + str(displace) + "x0 " +
       composite_image_fp
    os.system(cmd)

def createVerticalComposite(original_fp, map_fp,
   composite_image_fp, displace):
```

```
    cmd = "composite " + map_fp + " " + original_fp + "
        -displace 0x" + str(displace) + " " +
       composite_image_fp
    os.system(cmd)

def createBilateralComposite(original_fp, map_fp,
   composite_image_fp, displace_x, displace_y):
    cmd = "composite " + map_fp + " " + original_fp + "
        -displace " + str(displace_x) +"x" + str(
       displace_y) + " " + composite_image_fp
    os.system(cmd)

def createComposites(original_fp, map_fp, composite_fp,
    eyes, direction):
    step = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
       0.9, 1.0]
    if not os.path.exists(composite_fp):
        os.mkdir(composite_fp)
    for i in step:
        filename1 = str(int(i * 100)) + ".jpg"
        filename2 = "n" + str(int(i * 100)) + ".jpg"
        fp1 = os.path.join(composite_fp,filename1)
        fp2 = os.path.join(composite_fp,filename2)
        d = distance_between(eyes)
        displace1 = round(d * (i / 2.0))
        displace2 = round(0 - (d * (i / 2.0)))
        if direction == "bilateral":
            displace_y = round(0 - (d * (0.4 / 2.0))) #
               Hardcoded +40% displacement
            createBilateralComposite(original_fp,
               map_fp, fp1, displace1, displace_y)
            createBilateralComposite(original_fp,
               map_fp, fp2, displace2, displace_y)
        elif direction == "horizontal":
            createHorizontalComposite(original_fp,
               map_fp, fp1, displace1)
            createHorizontalComposite(original_fp,
               map_fp, fp2, displace2)
```

```
        else:
            createVerticalComposite(original_fp, map_fp
                , fp1, displace1)
            createVerticalComposite(original_fp, map_fp
                , fp2, displace2)

def distance_between(eyes):
    return eyes[1][0] - (eyes[0][0] + eyes[0][2])

def modify(eyes, direction="horizontal"):
    m = regex.match(filename)
    f = m.group(1) + ".jpg"
    map_fp = os.path.join(subject_map_fp,f)
    createMap(map_fp, eyes, original_image)
    composite_fp = os.path.join(subject_comp_fp, m.
        group(1))
    createComposites(original_image_fp, map_fp,
        composite_fp, eyes, direction)

for dirpath, dirnames, filenames in os.walk(originals):
    for subdirname in dirnames:
        subject_path = os.path.join(dirpath, subdirname
            )
        subject_map_fp = os.path.join(maps, subdirname)
        subject_comp_fp = os.path.join(composites,
            subdirname)
        for filename in os.listdir(subject_path):
            original_image_fp = os.path.join(
                subject_path,filename)
            original_image = cv2.imread(
                original_image_fp)
            eyes = eye_classifier1.detectMultiScale(
                original_image)
            if len(eyes) != 2:
                eyes = eye_classifier2.detectMultiScale
                    (original_image)
            if len(eyes) == 2:
                eyes = eyes.tolist()
```

```
                eyes = sorted(eyes, key=lambda x: x[0])
                if distance_between(eyes) > 0:
                    if not os.path.exists(
                       subject_map_fp):
                        os.mkdir(subject_map_fp)
                    if not os.path.exists(
                       subject_comp_fp):
                        os.mkdir(subject_comp_fp)
                    modify(eyes)
```

B.2 Recognition Testing Framework

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 16 12:44:31 2015

@author: Domenick Poster
"""

import random
import cv2
import numpy as np
import re
import os
import sys

sys.path.insert(0, r'C:¥Users¥user¥Documents¥School¥
   Research¥facerec¥py')
from facerec import model
from facerec import classifier
from facerec import feature
from facerec import distance

class Dataset:
    def __init__(self, name)
        :self.name = name
        self.subjects = {}
        self.totalOriginals = 0
```

```
        self.totalMods = {}
        self.scores = Scores(self.name)

    def load_originals(self, originals_path):
        print "Loading Original Images"
        subject_regex = re.compile(r"s(\d+)")
        original_image_regex = re.compile(r"(n?\d+)\.
            pgm")
        for dirname, dirnames, filenames in os.walk(
            originals_path):
             for subdirname in dirnames:
                    subject_path = os.path.join(dirname
                        , subdirname)
                    m = subject_regex.match(subdirname)
                    c = int(m.group(1))
                    subject = Subject(c)
                    self.subjects[c] = subject
                    for image_file in os.listdir(
                        subject_path):
                         m = original_image_regex.match(
                             image_file)
                         key = int(m.group(1))
                         image = Image(key, subject)
                         image_file_path = os.path.join(
                             subject_path, image_file)
                         im = cv2.imread(image_file_path
                             , cv2.IMREAD_GRAYSCALE)
                         im = np.asarray(im, dtype=np.
                             uint8)
                         original = ModifiedImage(im,
                             image_file_path, subject,
                             '0')
                         image.mods['0'] = original
                         subject.images[key] = image

    def load_mods(self, composites_path):
        print "Loading Modified Images"
        subject_regex = re.compile(r"s(\d+)")
```

```
        composite_image_regex = re.compile(r"(n?¥d+)¥.
            jpg")
        for subject_folder in os.listdir(
            composites_path):
            subject_folder_path = os.path.join(
                composites_path, subject_folder)
            m = subject_regex.match(subject_folder)
            subject_number = int(m.group(1))
            subject = self.subjects[subject_number]
            for image_folder in os.listdir(
                subject_folder_path):
                image_folder_path = os.path.join(
                    subject_folder_path, image_folder)
                if int(image_folder) not in subject.
                    images:
                    image = Image(int(image_folder),
                        subject.name)
                    self.subjects[subject_number].
                        images[int(image_folder)] =
                        image
                image = subject.images[int(image_folder
                    )]
                for mod_file in os.listdir(
                    image_folder_path):
                    m = composite_image_regex.match(
                        mod_file)
                    key = m.group(1)
                    mod_file_path = os.path.join(
                        image_folder_path, mod_file)
                    im = cv2.imread(mod_file_path, cv2.
                        IMREAD_GRAYSCALE)
                    im = np.asarray(im, dtype=np.uint8)
                    mod = ModifiedImage(im,
                        mod_file_path, subject, key)
                    image.mods[key] = mod

    def score(self, recognizer, train_size, folds
        ,train_on='0'):
```

```
        for subject in self.subjects.itervalues():
            subject.assembleFolds(train_size, folds
                , train_on)
        for subject in self.subjects.itervalues():
            print "Training Subject: " + str(subject.
                name)
            subject.train(self, recognizer, train_size,
                folds, train_on)
            print "Testing Subject: " + str(subject.
                name)
            self.scores.updateScores(subject.test())


    def getOtherSubjects(self, subject):
        other_subjects = []
        for s in self.subjects.itervalues():
            if subject.name != s.name:
                other_subjects.append(s)
        return other_subjects

    def clearScores(self):
        self.scores = Scores()
        for subject in self.subjects.itervalues():
            subject.clearScores()

class Fold:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def merge(self, fold):
        if len(fold.x) > 0:
            for i in range(0, len(fold.x)):
                self.x.append(fold.x[i])
                self.y.append(fold.y[i])

class Score:
    def __init__(self, name):
```

```
        self.name = name
        self.rank = dict((key,0) for key in range(1,11)
           )
        self.attempts = 0

    def score(self, classifier, image, subject)
        :p = classifier.predict(image)
        labels = p[1]['labels']
        for i in range(0, len(self.rank)):
            ranks = labels[0:i+1]
            if subject in ranks:
                self.rank[i+1] = self.rank[i+1] + 1
        self.attempts = self.attempts + 1

    def update(self, new_score):
        for i in range(0, len(self.rank)):
            self.rank[i+1] = self.rank[i+1] + new score
               .rank[i+1]
        self.attempts = self.attempts + new score.
           attempts

    def getAccuracy(self, n):
        return float(self.rank[n])/float(self.attempts)

    def __str__(self):
        return

class Scores:
    def __init__(self, name)
        :self.scores = {}
        self.name = name

    def updateScore(self, new_score):
        if new_score.name in self.scores:
            self.scores[new_score.name].update(
               new_score)
        else :
            self.scores[new_score.name] = new score
```

```
    def updateScores(self, new_scores):
        for new_score in new_scores.scores.itervalues()
            :
            self.updateScore(new_score)

    def __str__(self):
        d = {}
        for score in self.scores.itervalues():
            d[score.name] = [score.rank, score.attempts
                ]
        return d

class Subject:
    def __init__(self, name)
        :self.name = name
        self.images = {}
        self.scores = Scores(self.name)
        self.public_folds = []

    def assembleFolds(self, train_size, folds, train_on
        ):
        for i in range(0, folds):
            self.public_folds.append(self.
                getTrainingFold(train_size, train_on))

    def getTrainingFold(self, train_size, train_on,
        leave_out=None):
        #print "Retrieving Training Images for Subject
            #" + str(self.name)
        keys = self.images.keys()
        #print "Image List: " + str(keys)
        for key in list(keys):
            if train_on not in self.images[key].mods:
                keys.pop(keys.index(key))
        if leave_out is not None and leave_out in keys:
            #print "Leaving out " + str(leave_out)
            keys.pop(keys.index(leave_out))
```

```
        #print "Cleaned List: " + str(keys)
        X = []
        y = []
        sample_keys = random.sample(keys, min(
            train_size, len(keys)))
        #print "Final List: " + str(sample_keys)
        for k in sample_keys:
            X.append(self.images[k].mods[train_on].img)
            y.append(self.name)
        return Fold(X,y)

    def train(self, db, recognizer, train_size,
            fold_size, train_on='0'):
        other_subjects = db.getOtherSubjects(self)
        folds = []
        for i in range(0,fold_size):
            folds.append(Fold([],[]))

        for subject in other_subjects:
            for i in range(0, len(folds)):
                folds[i].merge(subject.public_folds[i])

        for image in self.images.itervalues():
            image.trainClassifiers(self, recognizer,
                folds, train_size, train_on)

    def test(self):
        for image in self.images.itervalues():
            self.scores.updateScores(image.test())
        return self.scores

    def clearScores(self):
        self.scores = Scores()
        for image in self.images.itervalues():
            image.clearScores()

    def __str__(self):
        return str(self.name)
```

```
    def __repr__(self): retur
        n str(self.name)

class Image:
    def __init__(self, name, subject):
        self.name = name
        self.subject = subject
        self.mods = {}
        self.classifiers = []
        self.scores = Scores(self.name)

    def trainClassifiers(self, subject, recognizer
       ,folds,train_size, train_on):
        final_folds = []
        for i in range(0,len(folds)):
            my_fold = subject.getTrainingFold(t
               rain size, train_on, self.name)
            if len(my_fold.x) == 0:
                return
            final_folds.append(Fold([],[]))
            final_folds[i].merge(folds[i])
            final_folds[i].merge(my_fold)
            classifier = createClassifier(recognizer,
               len(final_folds[i].y))
            classifier.compute(final_folds[i].x, np.
               asarray(final_folds[i].y, dtype=np.int32
               ))
            self.classifiers.append(classifier)

    def test(self):
        for mod in self.mods.itervalues():
            self.scores.updateScore(mod.test(self.
               classifiers))
        self.classifiers = []
        return self.scores

    def clearScores(self):
        self.scores = Scores()
```

```
        for mod in self.mods.itervalues():
            mod.clearScore()

    def __str__(self):
        return str(self.name)
    def __repr_(self):
        return str(self.name)

class ModifiedImage:
    def __init__(self, img, path, subject, change)
        :self.img = img
        self.path = path
        self.subject = subject
        self.change = change

    def test(self, classifiers):
        score = Score(self.change)
        for c in classifiers:
            score.score(c, self.img, self.subject.name)
        return score

    def clearScore(self):
        self.score = Score(self.change)

    def __str__(self): ret
        urn    self.change
    def __repr_(self):ret
        urn    self.change

def createClassifier(algorithm, k):
    c = classifier.NearestNeighbor(dist_metric =
       distance.EuclideanDistance(), k=k)
    if algorithm == "PCA":
        return model.PredictableModel(feature = feature
           .PCA(), classifier = c)
    elif algorithm == "LDA":
        return model.PredictableModel(feature = feature
           .Fisherfaces(), classifier = c)
```

```
    elif algorithm == "LBPH":
        return model.PredictableModel(feature = feature
            .SpatialHistogram(), classifier = c)
```

B.3 Batch Image Normalizer

```
# -*- coding: utf-8 -*-
"""
Created on Sun Sep 20 16:21:27 2015

@author: Domenick Poster
"""

import os
import re
import cv2_align
from PIL import Image

subject_regex = re.compile(r"s(¥d+)")
original_image_regex = re.compile(r"(n?¥d+)¥.pgm")

originals_path = r"C:¥Users¥user¥Documents¥School¥
    Research¥poster¥att_faces"
originals_normalized_path = r"C:¥Users¥user¥Documents¥
    School¥Research¥poster¥originals_normalized"
if not os.path.exists(originals_normalized_path):
    os.mkdir(originals_normalized_path)


for dirpath, dirnames, filenames in os.walk(
    originals_path):
    for subdirname in dirnames:
        subject_path = os.path.join(dirpath, subdirname
            )
        normalized_subject_fp = os.path.join(
            originals_normalized_path, subdirname)
        if not os.path.exists(normalized_subject_fp):
            os.mkdir(normalized_subject_fp)
```

```
        for filename in os.listdir(subject_path):
            original_image_fp = os.path.join(
               subject_path,filename)
            normalized_image_fp = os.path.join(
               normalized_subject_fp, filename)
            normalized_image_array = cv2_align.
               align_face(original_image_fp)
            if normalized_image_array is not None:
                normalized_image = Image.fromarray(
                   normalized_image_array)
                normalized_image.save(
                   normalized_image_fp)


mods_path = r"C:¥Users¥user¥Documents¥School¥Research¥
   poster¥composites"
mods_normalized_path = r"C:¥Users¥user¥Documents¥School
   ¥Research¥poster¥composites_normalized"
if not os.path.exists(mods_normalized_path):
    os.mkdir(mods_normalized_path)

for dirpath, dirnames, filenames in os.walk(mods_path):
    for subdirname in dirnames:
        subject_path = os.path.join(dirpath, subdirname
           )
        normalized_subject_fp = os.path.join(
           mods_normalized_path, subdirname)
        if not os.path.exists(normalized_subject_fp):
            os.mkdir(normalized_subject_fp)
        for image_folder in os.listdir(subject_path):
            image_folder_fp = os.path.join(subject_path
               , image_folder)
            print image_folder_fp
            normalized_image_folder_fp = os.path.join(
               normalized_subject_fp, image_folder)
            if len(os.listdir(image_folder_fp)) > 1:
                if not os.path.exists(
                   normalized_image_folder_fp):
```

```
            os.mkdir(normalized_image_folder_fp
                )
        for mod in os.listdir(image_folder_fp):
            mod_fp = os.path.join(
                image_folder_fp, mod)
            normalized_mod_fp = os.path.join(
                normalized_image_folder_fp, mod)
            normalized_image_array = cv2_align.
                align_face(mod_fp)
            if normalized_image_array is not
                None:
                normalized_image = Image.
                    fromarray(
                    normalized_image_array)
                normalized_image.save(
                    normalized_mod_fp)
```

B.4 Normalization Algorithm

```
#!/usr/bin/env python
# Software License Agreement (BSD License)
#
# Copyright (c) 2012, Philipp Wagner
# All rights reserved.
#
# Redistribution and use in source and binary forms,
    with or without
# modification, are permitted provided that the
    following conditions
# are met:
#
#  * Redistributions of source code must retain the
    above copyright
#    notice, this list of conditions and the following
    disclaimer.
#  * Redistributions in binary form must reproduce the
    above
#    copyright notice, this list of conditions and the
```

```
    following
#     disclaimer in the documentation and/or other
    materials provided
#     with the distribution.
#   . Neither the name of the author nor the names of
    its
#     contributors may be used to endorse or promote
    products derived
#     from this software without specific prior written
    permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
    AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
    INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
     AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
    SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
    DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
    DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
    OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
    INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
    CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE
    ) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
    ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

import sys, math
import PIL.Image as Image
import feature_detect
```

```
import numpy as np

def Distance(p1,p2):
  dx = p2[0] - p1[0]
  dy = p2[1] - p1[1]
  return math.sqrt(dx*dx+dy*dy)

def ScaleRotateTranslate(image, angle, center = None,
   new_center = None, scale = None, resample=Image.
   BICUBIC):
  if (scale is None) and (center is None):
    return image.rotate(angle=angle, resample=resample)
  nx,ny = x,y = center
  sx=sy=1.0
  if new_center:
    (nx,ny) = new_center
  if scale:
    (sx,sy) = (scale, scale)
  cosine = math.cos(angle)
  sine = math.sin(angle)
  a = cosine/sx
  b = sine/sx
  c =
  x-nx*a-ny*b d
  =-sine/sy
  e = cosine/sy
  f = y-nx*d-ny*e
  return image.transform(image.size, Image.AFFINE, (a,b
     ,c,d,e,f), resample=resample)

def CropFace(image, eye_left=(0,0), eye_right=(0,0),
   offset_pct=(0.2,0.2), dest_sz = (70,70)):
  # calculate offsets in original image
  offset_h = math.floor(float(offset_pct[0])*dest_sz
     [0])
  offset_v = math.floor(float(offset_pct[1])*dest_sz
     [1])
  # get the direction
  eye_direction = (eye_right[0] - eye_left[0],
```

```
    eye_right[1] - eye_left[1])
  # calc rotation angle in radians
  rotation = -math.atan2(float(eye_direction[1]),float(
    eye_direction[0]))
  # distance between them
  dist = Distance(eye_left, eye_right)
  # calculate the reference eye-width
  reference = dest_sz[0] - 2.0*offset_h
  # scale factor
  scale = float(dist)/float(reference)
  # rotate original around the left eye
  image = ScaleRotateTranslate(image, center=eye_left,
    angle=rotation)
  # crop the rotated image
  crop_xy = (eye_left[0] - scale*offset_h, eye_left[1]
    - scale*offset_v)
  crop_size = (dest_sz[0]*scale, dest_sz[1]*scale)
  image = image.crop((int(crop_xy[0]), int(crop_xy[1]),
     int(crop_xy[0]+crop_size[0]), int(crop_xy[1]+
    crop_size[1])))
  # resize it
  image = image.resize(dest_sz, Image.ANTIALIAS)
  return image

def eye_center(eye_coord=[]):
    ex, ey, ew, eh = eye_coord
    x = int(ex+(ew/2))
    y = int(ey+(eh/2))
    return (x,y)

def align_face(img_fp, resize_dim=(70, 70)):
    :aligned_img = None
    fd = feature_detect.feature_detect(img_fp)
    face = fd.find_face()
    if len(face) == 1:
        e = fd.find_eyes(detected_faces=face)
        if len(e) == 2:
            subj_img = Image.open(img_fp)
```

```
            eleft, eright = min(e[0], e[1], key=lambda
                x: x[0]), max(e[0], e[1], key=lambda x:
                x[0])
            subj_img = CropFace(subj_img, eye_left=
                eye_center(eleft), eye_right=eye_center(
                eright), offset_pct=(0.25, 0.25),
                dest_sz=resize_dim)
            subj_img = subj_img.convert(mode="L")  #
                converts from RGB mode to GrayScale
            aligned_img = np.asarray(subj_img, dtype=np
                .uint8)
            subj_img.close()
    return aligned_img
```

B.5 Feature Detection

```
# This will detect the face, and their features
# Source was modified from: http://opencv-python-
    tutroals.readthedocs.org/en/latest/py_tutorials/
    py_objdetect/py_face_detection/py_face_detection.
    html
#       and #from https://github.com/shantnu/FaceDetect/
    blob/master/face_detect.py

#import numpy as np
import cv2
import os

class feature_detect:
    def __init__(self, image_file):
        self.img = cv2.imread(image_file, cv2.
            IMREAD_GRAYSCALE)

    def find_face(self):
        ''' Using the haarcascade, by default,
            find_face will see if a face is found in the
        image provided when the feature_detect class
            was initialized.
```

```
        returns a tuple in the form (x, y, w, h)
            where   x is the x coordinate of the face
                    y is the y coordinate of the face
                    w is the width of the face
                    h is the height of the face
        '''
        clf_home = r'E:¥opencv¥build¥share¥OpenCV¥
           haarcascades'
        clf_lst = ['haarcascade_frontalface_default.xml
           ', 'haarcascade_frontalface_alt.xml', '
           haarcascade_frontalface_alt2.xml', '
           haarcascade_frontalface_alt_tree.xml']
        clf_lst = [os.path.join(clf_home, clf) for clf
           in clf_lst]
        found_face = []
        for classifier in clf_lst:
            face_cascade = cv2.CascadeClassifier(
               classifier)
            found_face = face_cascade.detectMultiScale(
               self.img, scaleFactor=1.3, minNeighbors
               =5)
            if len(found_face) != 0:
                break
        return found_face

    def find_eyes(self, eye_classifier=r'E:¥opencv¥
       build¥share¥OpenCV¥haarcascades¥hharcascade_eye.
       xml', detected_faces=[[0,0,0,0]]):
        ''' Using the Eye Haarcascade Classifier, by
           default, find_eyes will search the
        provided detected face for eyes.

        Input: eye classifier  : an XML document of an
           eye classifier file for detecting eyes
                detected_faces : previously found face(
                   s), if multiple faces are  in the
                   array
```

```
                then a list of lists is
                   expected in the format of
                    [[list1] [list2] ...] -> [
                      list 1] = (x, y, w, h)
                Where  x: the upper left most
                   x coordinate of a detected
                   face (in pixels)
                        y: the upper left most
                           y coordinate of ad
                           etected face (in
                           pixels)
                        w: the width (in pixels
                            from the upper left
                            corner) of the face
                        h: the height (in
                           pixels from the
                           upper left corner)
                           of the face
Output: a nested list of coordinates for each
   eye detected in the image.
each set of coordinates is in the format
        (eye_coord_x, eye_coord_y, eye_coord_w,
            eye_coord_h)
Where   eye_coord_x : the upper left most x
   coordinate of the eye
        eye_coord_y : the upper left most y
           coordinate of the eye
        eye_coord_w : the width (in pixels from
            eye_coord_x) the eye is
        eye_coord_h : the height (in pixels
           from eye_coord_x) the eye is
'''

cascade_home = r'E:¥opencv¥build¥share¥OpenCV¥
   haarcascades'

eye_cascade_list = ['haarcascade_eye.xml', '
   haarcascade_eye_tree_eyeglasses.xml', '
   haarcascade_mcs_lefteye.xml', '
```

```
    haarcascade_mcs_righteye.xml']
clf_lst = [os.path.join(cascade_home, clf) for
    clf in eye_cascade_list]
eyes_found = []
for cas in clf_lst:
    eyes_found = []
    eye_cascade = cv2.CascadeClassifier(cas)
    eyes_found = eye_cascade.detectMultiScale(
        self.gray)
    if len(eyes_found) == 2:
        eyes_found = eyes_found.tolist()
        break
    else:
        eyes_found = []

if not len(eyes_found) == 2:
    # last effort
    right_eye_cascade = cv2.CascadeClassifier(
        os.path.join(cascade_home, '
        haarcascade_righteye_2splits.xml'))
    left_eye_cascade  = cv2.CascadeClassifier(
        os.path.join(cascade_home, '
        haarcascade_lefteye_2splits.xml'))
    if len(right_eye_cascade.detectMultiScale(
        self.gray)) == 1 and len(
        left_eye_cascade.detectMultiScale(self.
        gray)) == 1:
        eyes_found.append(right_eye_cascade.
            detectMultiScale(self.gray).tolist()
            [0])
        eyes_found.append(left_eye_cascade.
            detectMultiScale(self.gray).tolist()
            [0])

if len(eyes_found) == 0:
    eyes_found = []
return eyes_found
```