2010

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

AUTOTAGIT: A SYSTEM FOR THE AUTOMATION OF IMAGE TAGGING IN THE FACEBOOK ARCHITECTURE

Brian Bullard

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science and Information Systems
University of North Carolina Wilmington

2010

Approved By

Advisory Committee

_____ _____

_____
Chair

Accepted By

_____
Dean, Graduate School

Table of Contents

Abstract

This paper describes the development of a web application for the automation of image tagging within the Facebook social network architecture. The underlying technologies for face detection and recognition are presented along with some of the inherent difficulties of such technologies. Also explored are the enhancement of face detection through the use of skin detection and the evaluation of a newly developed face detector created with the OpenCV toolset. A walkthrough of the completed web application, difficulties encountered through development, and screenshots of the application are included for further insight.

Acknowledgements

I would like to start by thanking my advisor Dr. Karl Ricanek. His teachings in the fields of Bioinformatics and Pattern Recognition were the spark of inspiration for AutoTagIt. To Dr. Bryan Reinicke and Dr. Eric Patterson, thank you for your guidance and support. To my family and fiancée, thank you all for your ever persistent love and encouragement. Thanks to all of my friends for lending their time, faces, and enthusiasm toward the project. And finally, I would like to thank Mr. Richard Alford for instilling the "I can do this" mentality.

List of Figures

Introduction

A popular feature among social networking sites is providing functionality for users to add meta-data about an image or a set of images. This meta-data may include names and locations associated with faces or other interesting features in the image. 'Tagging' is the term used by Facebook that refers to the process of a user selecting a bounding box around a subject of interest in an image and identifying who that subject is from the user's list of friends. Users most commonly select regions in the image that contain a face, but are not limited to selecting only faces. Facebook allows a user to upload an entire ablum at one time which could contain hundreds of images. The current tagging process requires that for every photo in which the user wishes to tag individuals, they must select the bounding regions and associated name for those chosen individuals. With hundreds of images and multiple subjects per image, the process can become quite time consuming.

This capstone project for the University of North Carolina Wilmington's Computer Science and Information Systems program consists of designing and implementing an application that would allow a user of Facebook to upload a set of images and have those images automatically tagged, where faces exist, based on a comparison of the faces in the image to faces of the user's local network of friends. Four components are required for a complete system that can detect and recognize faces in a web application environment.

First, face detection is achieved by using the Viola-Jones method of rapid object detection as described in Robust Real-time Object Detection [1]. Included as part of the face detection component, at least for color images, potential face locations will be screened using a skin detection process in an effort to reduce the number of false detections. Second, the system must be able to recognize the detected faces paired against a set of faces from the application user and the user's Facebook friends. Recognition is achieved through use of a well known pattern recognition algorithm known as Eigenfaces

for Recognition [2]. The third major component is the AutoTagIt application developed for the Facebook architecture consisting of the design and flow of user interface.

The final component is comprised of all the underlying technology and middleware needed for communication between the three other components. A separation exists between the web layer, being the Facebook application, and the detection and recognition components that will reside on a local server. A MySQL database acts as a central hub for data transfer between the client and server and provides a semi-stateful environment.

Background and Literature Review

Why build such a system?

The main goal for an Auto-Tagging feature in Facebook is to reduce the amount of time users spend manually tagging photos. The photo tagging process in Facebook is fairly simple, but can take a considerable amount of time when a user wishes to tag many images. The process, while being based on visual decisions, is quite repetitive. As a result, a sizable portion of images containing the user's face or friends' faces can be left untagged. Rich, network based information could be lost. The repetitive nature of the tagging process is motivation enough for automation.

Similar automatic tagging systems have been developed for photo storage or organization software such as Apple's iPhoto, Flickr, and Google's Picasa. The idea of object recognition paired with image tagging allows for automatic grouping of images based on the objects that appear in those images. Facebook handles image grouping in two ways. Facebook users have the option to create new albums to add photos that may have some significant correlation to the user. A separate album is created for each user and automatically updates with links to any image in which the user is tagged. As extra incentive for creating this system, no mainstream automatic tagging has been developed for Facebook.

Explanation of Facebook's photo tagging:

When a user uploads a photo to one of their Facebook albums, they have the option to select points where the faces or other objects of interest appear in the photo. An area is created around the subject and a drop down menu appears with a list of the user's Facebook friends. When a friend is selected from the list a link is created on the page holding the image. After the link is established, mouse-over events on those specific areas of the page show the linked user's name and the previously

created bounding box. These bounding box regions and corresponding links are known as a Tag in Facebook. Notifications are propagated to tagged friends where they option to remove the connection from the image.

Definition of a face with respect to computer vision:

To be able to automatically extract a feature from an image that will be used as the input to a recognition system, that feature must be well defined. For the purpose of face detection, a face will be defined as being in a full-frontal view where both eyes are visible and in vertical orientation such that the eyes are above the nose. The face must be minimally occlududed and must be illuminated such that all normal features of a face are distinguishable to the naked eye. Accessory items such as excessive facial hair, eye glasses, scarves and hats as well as other objects in the image that are positioned infront of the face are all considered to be occlusions. Another face can be an occluding object as well. Normal face features are to include the left and right eye as well as the nose and mouth. The maximum acceptable degree of in-plane rotation in either direction from the vertical axis is estimated to be less than 20 degrees. The same rule will hold true for out-of-plane rotation angles and its' respective axes as well. The dimensions of the face can be, at a minimum, no smaller than 24x24 pixels.

Difficulties of face detection and recognition:

Many detection or recognition studies use a set of training and test images in which the subjects exist in constrained conditions [8] where lighting, pose, and camera distance are controlled. These conditions provide for the optimal case for detection or recognition. Images in Facebook tend to be suboptimal for these purposes. Creative computer vision and image processing techniques become necessary to achieve results closer those results found when the optimal case exists. This section will

bring forth some of the difficulties involved with face detection and recognition components of this project.

Compensating for subject illumination and reflectance are important tasks when creating a robust detection or recognition system. We define subject illumination as the amount of light incident on the subject of interest in an image and reflectance as the amount of light reflected from objects in a scene [5]. Variations in the angle of illumination and intensity are of great concern to the fields of face detection and facial recognition because they can introduce large amounts of noise in training data. As evidence of this concern, several groups in the Computer Vision and Bioinformatics fields, including Yale and Carnegie-Melon have built publicly available training sets [Yale Face Database B and PIE Database, CMU] with multiple angles of illumination. Other members recommend building separate detectors for different illumination angles [18].

An extreme variation in illumination would be two images of the same face with a light source on opposite sides of the subject or at right angles to each other. The variation between these two images can be greater than the variation between two completely different faces with the same illumination angle [7]. Such differences between images of the same person can produce relatively large distances between the faces in the face space and could give rise to the system classifying that person as two different people.

Proper illumination of a face, for the purpose of this project, is defined as having all facial features sufficiently easy to distinguish in an image by the human eye. A wide degree of illumination variation upon a face is to be expected in typical Facebook images. The expectation is to be able to detect and recognize those examples where the extreme is not the case. In other words, if the face in question is fully visible to where there are no partial occlusions from the presence of shadows and we exclude all other variables such as pose, then we should be able to adequately detect or recognize the face.

Subject pose is another non-trivial obstacle to the field face detection and recognition. Neither the Viola-Jones detection method nor the Turk and Pentland facial recognition method lend themselves to be easily, rotationally invariant. However, the Viola-Jones algorithm is able to detect faces within an estimated +/- 15 degrees from the vertical position for in-plane variations [22]. Viola and Jones describe a process of training three detectors; one at the vertical position, another at 30 degrees, and a final at 60 degrees of rotation. Since the detectors can be rotated by 90 degrees, only three detectors need to be created to cover a full 360 degrees. A total of 12 detectors could instantly be generated from the first three. Any region of interest found by one of these detectors could then be geometrically rotated to match the vertical requirements of our facial recognition component effectively creating an in-plane rotation invariant system. Out-of-plane rotation would still stand to be more difficult to overcome, although work as in [21] on the subject of 3D transformations would prove to be useful for future adaptations of the system proposed for this project.

Viola-Jones Algorithm for face detection:

The Viola-Jones detection method makes use of a set of boosted classifiers composed of simple features that provide rapid detection of faces in an image. The algorithm gains speed boosts from use of a datastructure known as the integral image and the classifiers are scalable permitting detection of face over a range of sizes.

The classifiers used in the Viola-Jones method are composed of a set of rectangular Haar-like features. Haar-like features are based on Haar wavlets for approximation of complex objects that can exist as wave patterns, images, or surfaces [13]. The features are a set of squares and rectangles that are subdivided into various configurations of black and white regions. Examples are given in Figure 1. Each feature acts as a scalable sub-window that is overlaid on an area of the image. The difference is taken between total pixel intensities of the areas that map to black rectangles and regions mapped by

white rectangles. Any value obtained from the difference that is non-zero represents a change in intensity over that region of the image. Over a given set of normalized face images, there is a configuration of these features that can closely approximate the general intensity changes of a face [1], [12].

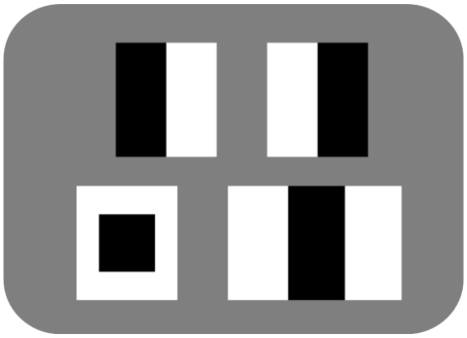

Figure 1 - A set of Haar-like features.

The integral image is a key part in reducing the computational time required for detection. It allows for complex operations over the image space to be performed with a few simple table look-ups. Viola and Jones describe the Integral Image such that each (x,y) coordinate maps to the sum of pixel intensities over a rectangular region above and to the left, with the origin of the image starting at the top left pixel [1]. The structure is also known as a Summed Area Table as first introduced by Franklin C. Crow for use in texture mapping [9]. The term Integral Image appears to be derived from the area sums that represent taking the integral over a set of pixel values.

Adaboost is the boosting method chosen for the Viola-Jones algorithm. It was developed by Yoav Freund and Robert E. Schapire who named the algorithm by shortening the term Adaptive Boosting. Adaboost is a variation of previous boosting methods explored by Freund and Schapire which required previous knowledge of weak learner accuracy [10]. Adaboost is an ensemble learning algorithm that makes use of a set of weak classifiers that combine to form a stronger classifier. Part of the algorithm involves maintaining a set of weights over the data points to be classified [11].

The Adaboost algorithm can be described in the following steps. For a given dataset, weight all examples in the set equally. Evaluate the first round of training by determining the weak classifier that best classifies the data. Recalculate the assigned weights such that correct classifications receive a reduced weight and incorrectly classified examples receive increased weights. Evaluate the next round of training with respect to the updated weights to obtain a second weak classifier. The example weights are, again, recalculated and the process is repeated. The reweighting scheme ensures that focus is applied, in the next iteration, to those examples that were incorrectly classified. The modified weights are used to produce a decision boundary that can correctly classify a majority of the examples that were previously misclassified. Subsequent iterations of this process should produce many weak classifiers or weak learners. A strong classifier should be the result of a weighted vote of the individual weak learners [11]. The process continues until a predetermined error goal is reached for the final classifier.

For the Viola-Jones algorithm, the example data is the set of labeled face and non-face images. Each weak classifier contains a single Haar-like feature at a given location. The best classifier per iteration of training is decided by finding the classifier having the greatest score over the entire set of images.

Processing Using Color Information:

When detecting faces in a color image the system can supplement the Viola-Jones algorithm with skin detection. There are two, immediately obvious ways that skin detection can boost the accuracy of detecting faces. First, face detection can proceed as normal with false detections being minimized by requiring that a certain percentage of skin pixels be present in a candidate face region. The second option involves applying skin detection over the entire image and routing only those regions containing skin pixels to the face detection component. Either option could potentially improve face detection accuracy or reduce the computational workload. Special consideration must be applied when

implementing either option since an inaccurate skin detection phase will be the limiting factor of the entire system.

We can observe skin pixels to exist within a certain range of combinations of Red, Green, and Blue in the RGB color space or as chrominance components, Cr and Cb, within the YCrCb color space. Luminance in the RGB color space can be derived from a weighted portion of each of the Red, Green, and Blue channels. Other color spaces include Hue-Saturation-Intensity, Tint-Saturation-Lightness, and Normalized RGB. Any of the color spaces can be used to detect skin features in an image but work presented in [24] suggest that those which separate the chrominance from luminance will show better performance. On the contrary, others in the field have conducted similar studies that show removing luminance does not improve skin detection [17]. A simple advantage of the YCrCb color space exists for skin detection. Having a lower dimensionality, as a result of only needing the two chrominance components, reduces the computational workload.

A skin detection system must have some prior knowledge of the boundaries of skin color, within a particular color space, to be able to classify pixels as skin or non-skin. Viewed in the three dimensions of Red, Green and Blue, the range of color values for human skin creates a fairly contiguous crescent, as seen in figure 2, and similarly seen in [16] and [25]. The boundaries of this crescent, represented in the RGB color space, are not smooth which makes explicitly defining the range of skin colors through a set of rules a fairly difficult task. The boundaries of skin color in the 2-dimensional, YCrCb color space also contain jagged edges and are thus similarly difficult to capture.

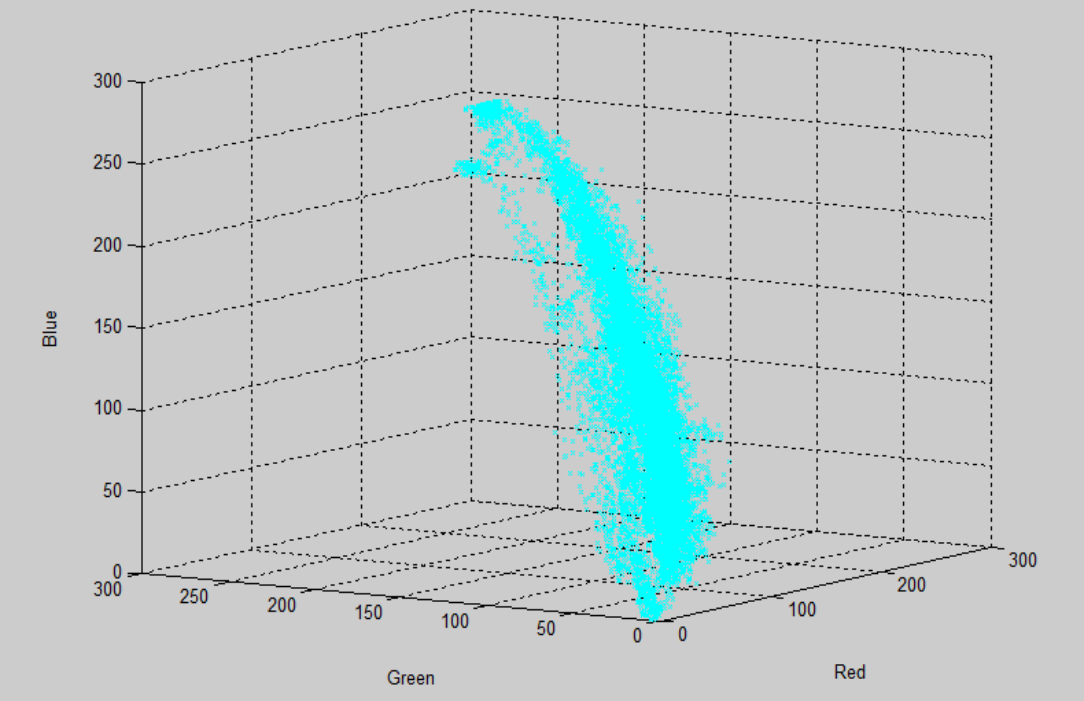

Figure 2 - Skin signature in the RGB color space. Pixel count = 12,549

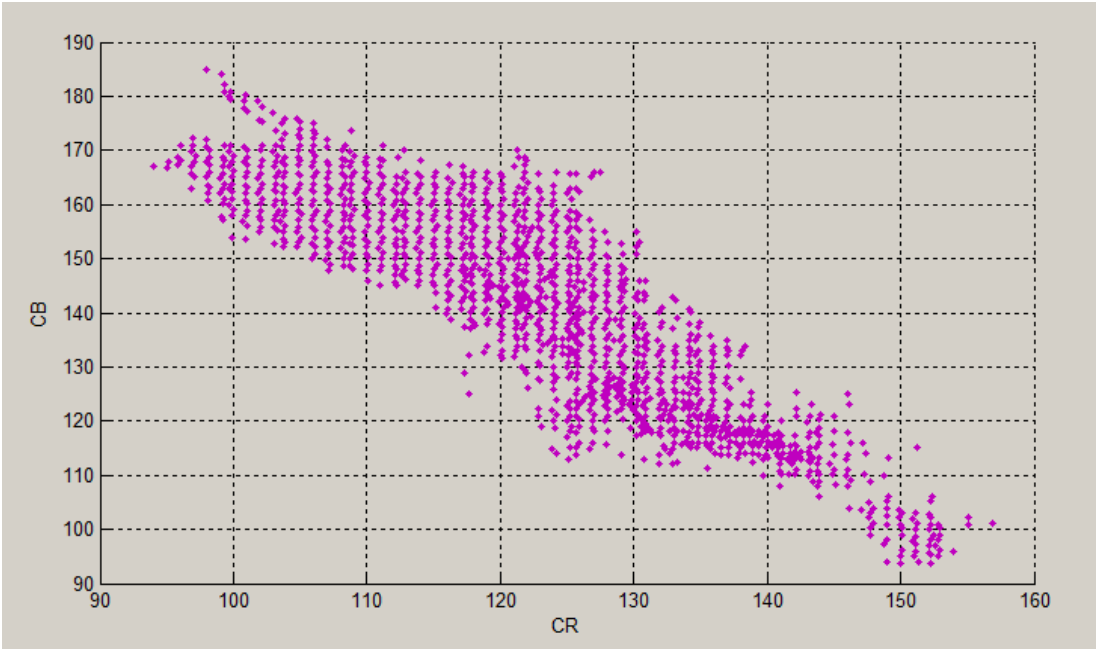

Figure 3 - Skin's Chrominance signature from CrCb. Pixel count = 12,549

Neural Networks for Skin Detection:

Neural Networks have a great amount of flexibility when learning patterns or approximating functions. Even simple, single layer networks, given enough perceptrons, can learn any linearly separable function [20]. Several powerful and well known neural net architectures exist, such as Hopfield Networks, Kohonen Self-Organizing Maps, and Adaptive Resonance Theory-2 networks, but the simplicity of Feed-Forward Back-Propagation network is very appealing. Given one hidden layer and enough hidden perceptrons, a Feed-Forward network can learn any function [19].

The basic architecture of a Feed-Forward Back-Propagation network is a connected map of input nodes to hidden nodes that are either connected to one or more sets of hidden layer nodes or a set of output nodes. A set of weights exists between each layer mapped to every node to node connection. The network functions as follows. A set of inputs are assigned to the input nodes. The hidden layer nodes and output layer nodes combine the weighted sum of inputs from the previous layer and apply an activation function, usually sigmoidal, to produce an output between 0 and 1. The final output is then compared to the expected output to produce an error signal. This error signal is then propagated backwards through the network to associate a level of responsibility to the connections between nodes for current output. The connection weights are updated based on the error signal via the Delta rule [19]. The next set of inputs is assigned and the process repeats until an arbitrary error rate is achieved. For skin detection, the training inputs can include either the Red, Green, and Blue components of the RGB space or the luminance and chrominance components of the YCrCb color space paired with a label indicating skin or not skin.

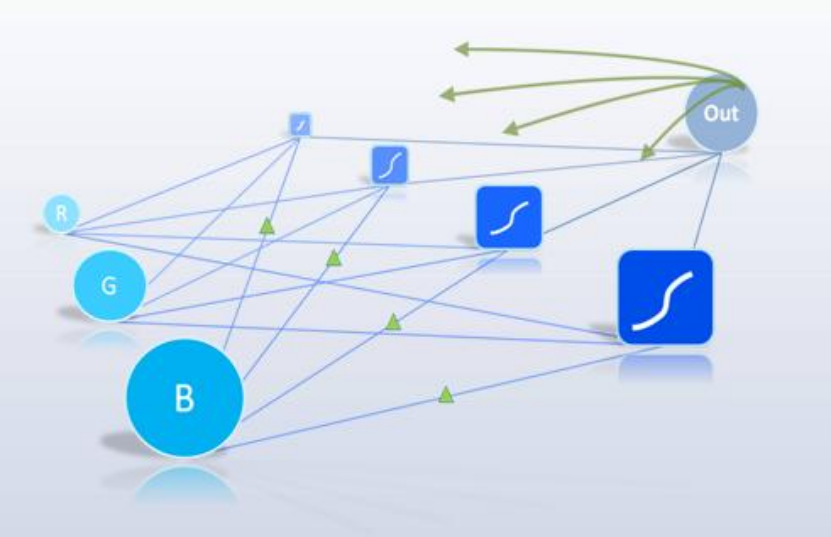

Figure 4 - Feed Forward Back Propagation Neural Network

EigenFaces -- Turk and Pentland algorithm:

The task of face recognition is achieved by use of an algorithm described by Mathew Turk and Alex Pentland that is known as Eigenfaces for Recognition [2]. Eigenfaces is a specialized form of Principal Component Analysis. This method utilizes a set of face images to construct a feature set known as the face space. The feature set can be described as the variance between the set of training data. Each feature in the feature set is called an Eigenface. The Eigenfaces are a set of vectors that contain the variance for each pixel spanning across the entire training set.

Another way to think about how Eigenfaces can be used for recognition is to imagine if we were to set a randomly selected weight for each of the Eigenvectors. The combination of Eigenvectors, transformed back into rows and columns, would synthesize a face. The generated face may look very close to someone either in the original dataset from which the Eigenvectors were created or it may look like someone completely different. It is the set of specific proportions from each Eigenface that encode the differences between the set of faces.

The basic Eigenfaces algorithm for recognition involves two higher level processes. First, the system must be trained with a set of face examples so that a representative model of the data exists that defines a face. That data model, a subset of the original data, is known as the feature space or face space and is used for comparison against newly introduced faces. Second, a new image must be obtained and projected into the face space created from step one. The projection of this face into the subspace allows it to be compared to faces in the training set. An optional step in the Eigenfaces algorithm not used in this project is to add newly classified faces to the set of training faces.

The feature space can be created by Eigen-decomposition of the covariance matrix of a set of data points. For Eigenfaces, the data points are the set of face images used for training [6]. A distance measure can be taken from the face space to determine whether the given subject closely matches a person in the training set or even if the subject closely matches a face at all. Once the face space is created, any image similar in dimension can be projected on to the subspace. The result of the projection is a vector of weights that can be compared to any face already existing in a set of projected faces.

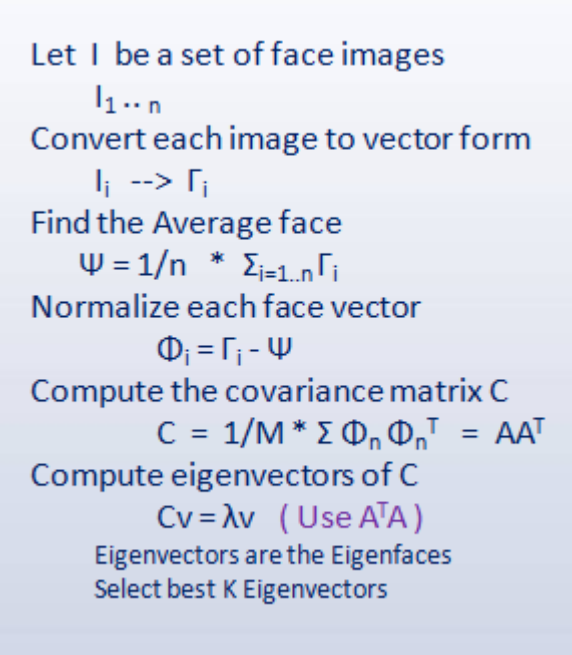

Figure 5 - Creating the Face Space for Eigenfaces

Why not use EigenFaces for detection?

Principle Component Analysis could indeed be used to detect faces as well as recognize a face compared to a set of known faces. The process would involve a slidding window approach that would take each sub-window of an image and project that as an image into the feature space of known faces. With this slidding window, every possible position of the window on the image is treated as its own image that needs to be classified as a face or non-face. The classification is made by taking a distance measure of the windowed image to every EigenFace in our face space. Given that this approach would also need to scale the window to match faces of different sizes it would not be computationally efficient for the image environment of Facebook.

For a PCA method such as EigenFaces to be able to perform the detection task it would be necessary to provide and maintain a scaled set of feature vectors to be used in comparison with faces of differing sizes [2]. The nature of the Viola-Jones algorithm greatly reduces the computational needs to

perform detection as compared to EigenFaces. These reductions in computation can be greatly attributed to the use of the integral image datastructure that needs to be computed only once for each image in the detection phase. The Viola-Jones algorithm needs only to compare averages of intensity over a region of the image which also factors into its computational cost reduction. Since the intensity averages can be obtained from a few simple integral image table lookups, scaling the search space requires the use of the same number of table lookups.

Software Development Life Cycle:

Every software system that is to be implemented and maintained in an environment requires some initial planning and analysis to derive an overview of how that system may work in that environment. Many development methodologies have been created to act as a guideline for successful implementation and deployment of such software systems. One of the most well known methodologies is the Waterfall method. The major steps of the Waterfall method include Planning, Analysis, Design, and Implementation. Within this methodology, as one step is completed, work begins in the next. The process continues until the final step, Implementation, is complete. After the final stage in the development process complete a working system should exist.

The Waterfall software development methodology has, over time, been mostly phased out. It is well known that the majority of cases require an altered version of this model in which the planning, analysis, design, and implementation steps are not necessarily sequential. These steps can be further broken down into smaller, intermediate steps as needed and can overlap. When requirements are not clearly defined or well known they may need to be refined as the developers explore the underlying technology.

The software development model that this project will most closely follow is that of Throwaway Prototyping. With this model, a thorough effort is put toward the planning and analysis phase to

provide a general understanding of the overall system. The underlying technology, however, may be unknown and prototypes of portions of the system are developed to explore these technologies [4]. Based on more of an iterative development style, requirements are continuously refined throughout later portions of the development life cycle.

It is important to note that the initial planning and analysis of the Throwaway Prototyping model, however incomplete, is still an important step to a successful deployment. Gathering and documenting system requirements aids in team synchronization, motivates a direction for development, and limits the scope within a project. To acquire the full benefit from these actions, the project presented in this paper will include a set of Use Case diagrams for portions of the system requiring user interaction and System State diagrams used to map out the backend processes.

An iterative approach has been set in place for this project with the development of a high level view of the system and the development of some of the core portions of the system. The technology has been explored and initial ideas about web to system communication have been redesigned. The beginning iterations for the project have focused on building core components for feasibility estimates and the design of component communication based on those estimates. The core components should be built and tested to work independent of one another. Later iterations will focus more on integration of the components and less on building or adding to them. Final iterations should include testing the system as a whole and updating specific features based on those tests.

Implementation

Overview of system requirements:

- Provide an interface to allow a user to sign-up for the Auto-Tagging application on Facebook.
- Prompt for images that exemplify how the user's face generally appears in their photos. A small set of images, between 5 and 10, is recommended. The users face should be the focus or foreground of the images. This feature should allow the system to maintain an up to date representation of a person's face.
- Images will only be stored temporarily
- Data to be stored in the database:
    - Facebook User ID (Storage of personal data not permitted (first and last name)
    - ID's of Facebook users who are also friends of a given user
    - A set of images of the user that have been projected onto the face space.
- Detect faces in a photo or set of photos in an album uploaded to the application server.
- Determine a match to each face, based on a threshold, so that if a face cannot be recognized, the system will not tag that face.
- The system should also allow the user to accept, decline (delete/change) the tagged faces.
- Once the user has completed a review of the tags an album will be posted to Facebook containing all of the images with tag data.

Facebook Application and User Interface:

The user interface was developed with Facebook's Application Programming Interface, or API, and the PHP web programming language. Facebook has developed a set of method calls within a language known as FBML or Facebook Markup Language and a web service to recognize those calls to deliver dynamic content to a user of an application. There are many configurations provided by Facebook to access these methods. The configuration used by the application portion of this capstone

project can be understood as a local web server running PHP that connects to the Facebook web service via the FBML method calls.

A familiar look and feel for Facebook users is achieved through use of the "Canvas Pages" option. A canvas page acts as a viewing window to content delivered by the Auto-Tagging web server. Through use of the PHP programming language, the Facebook Markup Language and the data stored in the MySQL database an event-state driven interface can be delivered for the Auto-Tagging service users. Certain user states and selections can trigger sequential event views such as an image upload menu for initializing the face recognition component or the menu for uploading albums to be processed.

Signup for the application:

The process of signing up for the application is fairly simple. When a user visits the sign-in page they are presented with the familiar Facebook login menu. This login is required for every new session whether it's the first time to the application site or not. A successful login into Facebook grants access the next view of the Auto-Tagging application, in which the user will have the choice to accept the Terms and Conditions or decline. Upon acceptance of these terms, the user will then have the AutoTagIt_HOME view and a record will be generated in the MySQL database. Users are remembered upon subsequent visits by their Facebook UID and dynamic content can be delivered based on the data stored for each user. A set of event state diagrams found in Appendix B provide a detailed view of the user experience within the Auto-Tagging application.

Application Security

As security is always a concern, due diligance will be applied to maintain the privacy of personal data. It may concern some that the application requires an initial Facebook login to gain access to the

application. Facebook has steps in place to ensure that the login information of its users is safe. Facebook login passwords are never seen and thus are never stored on the AutoTagIt system. A RequireLogin() call from PHP redirects the user to the Facebook login page. This action serves several purposes. First it ensures that the user logging into the AutoTagIt application is a registered Facebook user [15]. The call secondly serves as a way to allow the user to authorize the application meaning that the user gives the application access rights to specific Facebook user data.

An added feature of the latest Facebook authorization procedures allows for an application to explicitly ask for certain user data rights. Through this feature, an AutoTagIt user will know exactly what information will be accessible to the application. The AutoTagIt application requests the "User Photos" permission and the "Stream Publish" permission for notifying the user's friends and updating the user's "Wall" when photos have been tagged. Upon authorization, a registered Facebook user's unique identifier is supplied with a temporary session to be used by the AutoTagIt. This session can be used to help maintain the privacy of the user and provides FBML calls to be made on behalf of the Facebook user.

AutoTagIt Process Flow

This section will provide a full scenario of the intended flow of operations new AutoTagIt users should follow. Website screenshots can be found in Appendix D.

Adding Friends to the System

To be able to automatically tag a friend's face the system needs a way to keep track who the user's friends are. The question might be "How do I add my friends?" Adding friends is a passive process. The system takes care of this task automatically by checking the friend status in Facebook. Upon login once query is made to Facebook to get the users set of friends and another query to the

AutoTagIt database is made on a Join of the auto_user table and the friends table. PHP then iterates through the lists checking if are any Facebook friends that are not listed in the friends table for the user. If so then a pair of the friend_id and auto_user_id are inserted into the AutoTagIt friend table.

Define You

After the user is enrolled into the system, the first page encountered is the AutoTagIt home screen. They are presented with a welcome message and several links. The first step for a new user is to enroll their face into the database by selecting the "Define You" link. A java applet will appear on the page with a file open dialog where the user should select a set of images that contains their face. When the user clicks "Open" the image files are stored in the database. A system background process will then call FindAndExtract.py to extract a set of faces from the images. Any face encountered is resized and stored in the database as a new image with the is_define_you flag set in the database. The user is redirected to the same page now with a set of face images and check boxes. Any of the presented faces the user selects will be stored in the eigen_faces table along with the user's Facebook id. Once a few of the users face images are projected they will be able to start auto-tagging images.

Upload Images

The "Upload Images" link is where the user can upload a set of images for auto-tagging. This page acts almost identical to the "Define You" page except the user no longer selects from a set of extracted faces. Instead the user will be presented with a notification that faces were found, or not, and should proceed to the Review and Submit page.

Review and Submit

On the "Review and Submit" page the user is presented with a set of extracted faces from the previous upload operation. The user is not required to submit the images to Facebook right away. Images will be stored on the database for no more than one week so the review and submit process can

happen at a later date. For the review process any face that is incorrectly labeled can be checked as such. Also on this page, apart from rejecting labeled faces, the user is given a field to supply an album title that will be generated when the user clicks submit. If no title is supplied then the images are uploaded to an album that Facebook automatically generates.

I hate ur app!

In the unlikely event that a user is dissatisfied with the application and wish to be removed from auto tagging phenomenon they may choose such action on the "I hate ur app!" page. A simple "Remove Me" button is presented that will delete any record containing the user's Facebook ID.

System Architecture:

Since OpenCV was originally developed in the C++ programming language, a communication bridge is needed to pass data between PHP and C++. A shell command in the form of *exec("C:\Program\ScriptName.exe parameter1 parameter2…")* allows the PHP web programming language to call server-side executables and can accept a string or integer as a return. As a personal choice, the Python programming language was used to implement the server level executables or scripts. Support for OpenCV was possible through a set of wrapper classes that allow the Python programming language interpreter to call the necessary OpenCV face detection functions. A team of robotics experts known as WillowGarage maintain and distribute OpenCV and the Python wrapper classes. Their wrapper classes [14] were the first evaluated for the AutoTagIt face detection scripts. Development on the PyOpenCV classes by Pham Minh Tri [27] later produced a more desirable interface. At the time of implementation, PyOpenCV was more compatible with the Python libraries than the WillowGarage wrapper classes.

At the heart of the application resides the MySQL Database that serves as a central hub for data transfer between the separate components. In the client-server model, some information must be processed on the server side but may require initiation by a user or client machine. An example of that transaction can be seen in the AutoTagIt image upload feature. If the "Define You" link is selected, the user is presented with the java image upload applet. Images are temporarily stored in the database through a JDBC connection after selection from the applet. Once uploading is complete the user clicks a button on the web page that initiates the face detection, skin detection, and face recognition scripts. The Python component is able to connect to the database, read from the tables, and update the tables with extracted face images. Once face processing is complete, PHP can reload the current page to display any extracted faces stored in the database from the previous action. The MySQL database makes that process seamless since all the separate components can connect to and update the tables. An illustration of the component communication can be seen in Figure 4.

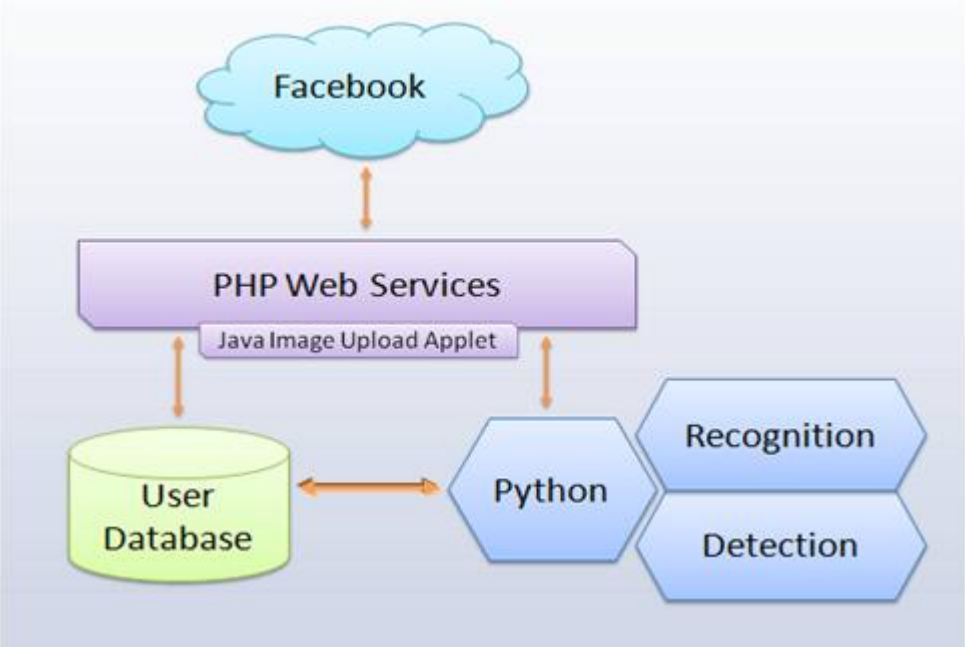

Figure 6 - Component communication architecture.

Face Detection:

An open library of classes and methods known as OpenCV was developed by the Intel Corporation for the C/C++ language. OpenCV has a set of methods created for general object detection. Object detectors, the Haar-like feature cascades, are stored as XML files and can be quickly interchanged based on the detection problem. A set of C/C++ wrapper classes has been created for the Python programming language that enable use of the OpenCV library for this language. PyOpenCV is the set of wrappers for Python developed by Pham Minh Tri [27]. Python is used in lieu of C/C++ to because of its rapid application or scripting development and flexibility. Face detection for the AutoTagIt system is achieved through use of a simple class developed in Python using PyOpenCV. The face detection code can be found in Appendix A. Skin detection occurs just after the face detection and is therefore called from within the same server side instance as the face detection portion.

Skin Detection:

Skin detection has the potential to improve computational workload and increase the accuracy of the face detection phase of the system. Observed regions of difficulty in an image for the face detectors used by OpenCV include areas of bold text, leafy tree areas, and rough water. The patterns presented in these examples often lead to false positives due to their similarity in the position of light and dark shadow areas of a face. The aid of the skin detector should prove useful to ruling out these more difficult areas as long as the image supplied is in normal color and lighting conditions. For this project, skin detection is used in a way that supplements the decisions made by the Viola-Jones face detector. There were two methods assessed for implementing skin detection in the proposed system. They will be denoted as method A and method B.

The process for method A works as follows, if a region selected by the face detector contains a certain percentage of skin pixels then the confidence of classifying that region as a face is increased.

Method B works in the opposite manner as a preprocessing filter, applying skin detection to the image first and sending only those regions containing skin pixels to the face detector. Deciding how to pair skin detection with face detection involved weighing the costs and benefits of each method. Method A would require searching the entire image with the face detector where method B only scans portions where skin pixels exist. An upside for method A is that face detection through OpenCV is quite fast for fairly large sized images. Since skin detection involves evaluating each pixel, method B takes a slight hit in performance. Method B would also require an intermediate step to find the bounding rectangles encompassing the skin pixels. A downside for method A is that skin detection could be applied over the same pixels multiple times. The reason for a possibility of multiple passes is that face candidates from OpenCV can overlap.

OpenCV uses Canny Pruning, an edge detection algorithm that serves as a search area reduction technique. In OpenCV Canny Pruning works by removing areas of the image that do not contain enough edge information to be interesting to face detection. The preprocessing filter approach, Method B, could potentially use the Canny Pruning feature to reduce extra computation for regions that may be classified as skin but have no edge information. The method chosen for this project was method A. The workload for skin detection over the combined face detections from OpenCV, with overlapping detections, will be smaller than evaluating skin pixels over the entire image.

There are various ways to implement skin detection. The first of two early implementations for the AutoTagIt system was a set of rules that tried to capture skin in the RGB color space and second, a lookup table that could be used with any color space. The set of rules for skin detection were designed by Tomaz et al. and presented in [26]. The rule set classifier benefits from speedy processing since it is quite small and acts as a cascaded classification system, much like the Viola-Jones face detectors. A pixel is labeled as non-skin if it meets the requirements of any rule in the set.

The results of this skin detection method are good considering the small number of decision operations needed, however, a fair amount of non-skin pixels are misclassified. The outcome of a gross misclassification could allow non-faces to be presented to the face recognition system.

The second implementation seemed more promising. The lookup table is a simple data structure in which the values of each component of the chosen color space make up the axes of an n-dimensional array. A value of 0 or 1 is assigned to each point in the array. To update the lookup table a separate program was constructed to utilize mouse click and drag events. When one of the events is encountered, the program will extract the current color from the location of the mouse pointer if it is hovering over the image presented. The values from the selected point are used as an index to set the corresponding value in the n-dimensional array to 1. The results from the lookup table are more accurate for skin detection but are limited to detecting only those skin values that have been encountered during training. A large set of samples covering all skin colors in various lighting conditions would be required to cover the complete range of skin tones in unconstrained images.

Precise skin detection is an inherently hard problem for any system, no matter the color space. Common objects in nature can have color signatures overlapping with human skin color [23]. Variations in lighting conditions will shift the skin color boundaries. It is no surprise that some false acceptance will be encountered. A smooth approximation of these skin regions might offer a fair balance between the excess misclassifications of a course decision system and the sparse regions created by a direct mapping from hand selecting examples.

A Feed-Forward Back-Propagation Neural Network was designed as a third skin detection implementation. The network has one hidden layer and uses a Sigmoid activation function on each hidden layer node as well as the output node. A neural network would have the capability to learn the set of functions that would create a smooth boundary for skin classification. Inputs to the network are presented for each pixel in the training set as a vector of the chrominance components from the YCrCb

color space. Different configurations of the number of hidden layer nodes have been tested with some positive results outputted from networks with a set between six and fifteen nodes. A hidden layer with a smaller number of hidden nodes that can effectively classify skin pixels is optimal since each pixel in an image must be presented to the neural network. A larger number of hidden nodes produces a significant increase in computational complexity and increases the time required to process each image.

The results of the Neural Network, while on track, were less than optimal. Acceptance rates of large values of green and magenta were observed after training in the YCrCb color space. Training in the RGB color space saw spikes in higher values moving toward white and including white. To reduce some of the computational workload, the average chrominance or RGB values are evaluated for an area of the image and presented to the classifier. The same averaging step was applied to the sample selection tool to reduce the inclusion of erroneous from noise in the training images. Given the observed results from each of the skin detection methods implemented, the lookup table approach will be used with the YCrCb color space. The system should be able to fill in some holes in the data by applying a dilation function to the table after training. The classification time is linear and skin model is quite easily visualized in two dimensions.

Face Recognition:

The face recognition component was developed purely in Python. A set of 884 faces, set at dimensions 50x50, were gathered to create the face space used by AutoTagIt. The image sources were again Flickr, Facebook, and the Labeled Faces in the Wild dataset where the approximate distribution from each respectively is 15%, 20%, and 65%. The python script Eigens.py was created to develop the face space to be used by AutoTagIt. The script first scans a directory for images files. These images should just individual faces aligned to be in the same orientation as closely as possible through visual inspection. The faces are averaged and mean adjusted to calculate the set of phi vectors. These vectors

represent the variance of each face from the mean. A covariance matrix is created from set of phi vectors and Eigen decomposition is applied. The set of Eigenvectors and Eigenvalues are sorted and following work by [30] the top three eigenvectors are removed as well as the lower 20%. It is believed that the first few Eigenvectors contain illumination data that, in practice, degrades the performance of recognition. The lower valued Eigenvectors are do not contain much useable information and are of little importance. A final dot product of the eigenvectors and phi vectors creates the set of weight vectors that comprise the face space. The set of Eigenvectors, the mean image, and the weight vectors are stored on the file system for later use. Eigens.py can found in Appendix A.

To complete the task of recognition, the system requires a set of projections for each user compare with newly encountered faces. The AutoTagIt application stores new projections through the "Define You" link. After the user selects a set of images, the system extracts any faces and presents them to the user. Any face select from the result set gets stored in the database as a projection onto the face space. The steps for projecting a face can be seen in Figure 7 – Projecting a Face onto the Face Space.

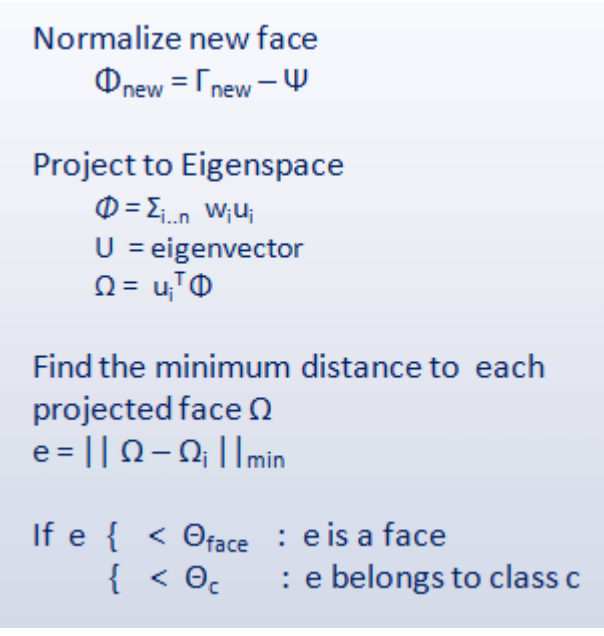

Figure 7 - Projecting a Face onto the Face Space

The AutoTagIt system uses a Cosine Angle Dissimilarity for distance measures. The system will classify a new face, f - the probe face, as being a match to a face class F in the face space if the distance between the f and the average of F is below some in-class threshold, Θ_c. That threshold, which can apply to both Euclidean and Cosine measures, will be the maximum distance of a face from class F to the mean of F. It can also stated that if the distance to the average face is below the threshold Θ_f for the face space but greater than the threshold Θ_c any stored face class then the face has not been seen by the system. Presented in Figure 8 is an example of the Cosine distance measure. The green circle represents the centroid of the class of blue circles and the distance is the measure of the angle from the green circle to another example. This particular case would give Θ_c for the class of blue circles.

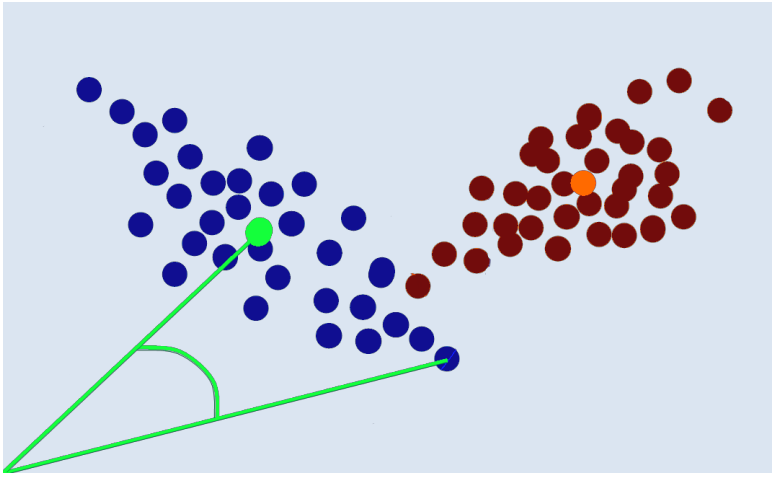

Figure 8 - Cosine Angle Dissimilarity

System Development Obstacles

A restriction to the Eigenfaces recognition algorithm is that all the example faces must be the same size which may result in information loss when resizing. The standardization is necessary for Principal Component Analysis to obtain the mean difference of each face. The training method for the Viola-Jones detection phase also requires a set of images with uniform width and height. Simple

intermediate steps were applied while building the training and testing sets for both face detection and recognition to obtain images with equal widths and heights. Sub-image resizing is also built into the application as an automated process when a face is encountered in images uploaded to the server.

For proper training of face recognition, faces were obtained from Facebook images and scaled to match in size. A small dimensionality of 24x24 pixels was used to reduce the amount of computation needed for training and comparison. It also fits to use a smaller face size since a sizable portion of images on Facebook contain groups of people where the purpose of the image is to capture the group in a particular location. Since the focus of this type of image is less concerned with the individuals' faces, the size of the extracted regions will be small in comparison to image size. It is also much more difficult to create usable information by increasing the size of a region of interest. Increasing image size produces unwanted artifacts that can be interpreted as belonging to a subject's facial features.

The changing Facebook environment has produced many obstacles for developing an application. Late in the development stage of AutoTagIt, odd behavior started to appear that does not generally plague a normal website. Missing variable data, HTML tag erros and broken API calls caused long hunts for documentation and general knowledge base inquiries, often with few results for an explanation.

To increase the security of private data in the Facebook environment, the company installed several restrictions for the HTML and PHP languages. A major feature in of the AutoTagIt application has been affected by these restrictions. A developer should be able to use an HTML form defined with the "POST" method and the encoding type set to "multipart/form-data". Within this form a file selection button can be presented to the user by use of the <input type="file"> tag. When the form is submitted by the user, PHP handles the encoded data and stores it in the $_FILES global variable. Facebook appears to have removed support for the $_FILES variable. This action could not be found within Facebook's PHP developer documentation. Attempts to use the variable within the canvas

application resulted in returning an empty array. Tests outside of the Facebook environment resulted in the correct data being present within $_FILES array.

Due to the restriction on certain PHP variables a different solution was needed for the file upload feature. A Java applet has the ability to transfer files and data from the client side to the server. For AutoTagIt, the data is inserted directly into the MySQL database. This solution presented its own set of difficulties with the Facebook canvas setting. The HTML <applet> and <object> tags, required for a Java applet, have been completely removed. To access the Java applet the users must be redirected to a page outside of the Facebook canvas environment. An odd solution to this problem is to embed an outside page into an iframe which is best achieved through use of the FBML <fb:iframe> tag. From within the fb:iframe session data can be passed to the applet through the applet <param> HTML tags. Facebook's Graph API is still fairly new and partly in development. It is currently missing a few very important features from the old REST API that would have made developing the AutoTagIt system much simpler. The most important example is the lack of ability to set a tag on an image in either the upload phase or after the image already exists within an album on Facebook. create the application in a pure Facebook "canvas" application. Use of FBML only used to create the iframe. FBML is restricted for use only within the canvas application environment… as in not able to be used within an iframe. Session varialbles must be passed to the iframe through the parameter html tags.

Testing and Analysis

Development and Analysis of a Custom Face Detector:

To get a better understanding of the Viola-Jones algorithm and its limitations, a study was conducted using the preconstructed face detection models designed by the Intel Corporation. These detectors are packaged with the OpenCV image processing library. The test set included 53 images containing 91 faces that were collected from Facebook and exemplify the types of images one might

expect to see in an unconstrained environment. The set is a mixture of examples that may contain zero to many faces which each match the definition of a face described in this document. Other faces may exist that do not reasonably fit the face definition. These faces are counted as any other non-face object appearing in the scene. Examples include profile or side view faces and faces that exhibit a considerable degree of rotation either in or out of plane.

Several face detectors were tested using OpenCV. A detector built for this system was generated from a base of 941 hand selected faces from Flickr, the Labeled Faces in the Wild dataset [3] and Facebook. The first OpenCV detector tested was the Frontal_Face_Default_Haarcascade. This detector performed very well on the test set with a positive detection rate of 99%. The other OpenCV supplied detector tested was the Frontal_Face_Alt haarcascade which performed similarly with a positive detection rate of 94.5%.

A downfall of the OpenCV default detectors is that their false accept rates can be rather high when trying to encompass as many true faces as possible. Many images contained detections that either did not contain a face at all or contained a face that did not match the constraints of the AutoTagIt system. The default cascade produced 80 false detections out of 53 images the alternate cascade produced 36. The detector that was developed for this system had a lower performance in positive detection with a rate of 88%. Where this detector displayed an improvement over the others is in a lower false positive rate with 7 false detections in 53 images. A lower false positive rate is beneficial to the system since any portion of the image that is computed to be a face becomes an input to the recognition component. A secondary non-face filter is applied at the recognition phase but less erroneous data is better.

Another limitation of the default OpenCV detectors, for the AutoTagIt system, is the area included around a face upon detection. The three detectors tested from the OpenCV suite include area around a face that will not be used by our system including the forehead, hair and chin. Multiple

detections from the test included a large portion of the background in the image as well. We estimated that a large portion of variance noise could be attributed to these extra portions of the face and background that are obtained from the default detectors developed for OpenCV. Observations gathered previously from detection testing about these variances between images were taken into consideration when formulating our definition of a face.

Reference

[1] Paul Viola and Michael Jones. “Robust Real-time Object Detection”.
Second International Workshop on Statistical and Computational Theories of Vision – Modeling, Learning, Computing, and Sampling Vancouver, Canada, July 13, 2001.

[2] Mathew Turk and Alex Pentland. "Eigenfaces for recognition". *Journal of Cognitive Neuroscience 3 (1): 71–86,* 1991.

[3] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller.
Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. *University of Massachusetts, Amherst, Technical Report 07-49,* October, 2007.

[4] Alan R. Dennis, Barbara H. Wixom, and David Tegarden. Systems Analysis and Design with UML Version 2.0: An Object-Oriented Approach, Third Edition. *John Wiley & Sons. © Books24x7*, 2009.

[5] Rafael C. Gonzalez, and Richard E. Woods. Digital Image Processing, 3rd ed., *Prentice Hall,* Upper Saddle River, NJ. 2008.

[6] Robin Hewitt. “Seeing With OpenCV”. *SERVO Magazine, T & L Publications, Inc.,* January 2007*.*
Reprinted on *Cognotics: Resources for Cognitive Robotics*.
http://www.cognotics.com/opencv/servo_2007_series/index.html, January 24, 2010.

[7] Athinodoros S. Georghiades, Peter N. Belhumeur, and David J. Kriegman. “From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose”.
IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23, No. 6, JUNE 2001.

[8] Aleix M. Martinez. “Recognizing Imprecisely Localized, Partially Occluded, and Expression Variant Faces from a Single Sample per Class”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, VOL. 24, NO. 6, JUNE 2002

[9] Franklin C. Crow. “Summed-area tables for texture mapping”. In *Proceedings of the 11th Annual Conference on Computer Graphics and interactive Techniques* H. Christiansen, Ed. SIGGRAPH '84. ACM, New York, NY, 207-212, 1984.

[10] Yoav Freund and Robert E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting” *Journal of Computer and System Sciences*, no. 55, 1997

[11] Yoav Freund Robert E. Schapire “A Short Introduction to Boosting” *AT&T Labs* – Research Shannon Laboratory 180 Park Avenue Florham Park, NJ 07932
www.research.att.com, 1999.

[12] Constantine Papageorgiou, Michael Oren and Tomaso Poggio, "A general framework for object detection", *International Conference on Computer Vision*, 1998.

[13] Eric J. Stollnitz , Tony D. DeRose and David H. Salesin. “Wavelets for Computer Graphics: A Primer”, Part 1. *IEEE Computer Graphics and Applications*, v.15 n.3, p.76-84, May 1995.

[14] Willow Garage. *Python Interface*.
http://opencv.willowgarage.com/documentation/python/index.html, 2009.

[15] *Facebook.com*. *Facebook Developers Wiki.*
http://wiki.developers.facebook.com/index.php/Main_Page, 22 January 2010.

[16] Ming-Jung Seow, Deepthi Valaparla, and Vijayan K. Asari. "Neural Network Based Skin Color Model for Face Detection*". VLSI Systems Laboratory, Department of Electrical and Computer Engineering*. Old Dominion University, Norfolk, VA 23529, 2003.

[17] Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. "A Survey on Pixel-Based Skin Color Detection Techniques". *Graphics and Media Laboratory, Faculty of Computational Mathematics and Cybernetics*, Moscow State University, Moscow, Russia, 2003.

[18] Alister Cordiner, Philip Ogunbona, and Wanqing Li. "Illumination Invariant Face Detection Using Classifier Fusion". *CVIPCM, University of Wollongong, Wollongong, Australia*
Digisensory Technologies, Sydney, Australia, December 11, 2008.

[19] Richard Duda, Peter Hart, and David Stork. Pattern Classification, 2nd ed. *John Wiley & Sons. ©,* New York, 2001.

[20] A. Novikoff," On convergence proofs on perceptrons". *Symposium on the Mathematical Theory of Automata*, 12, 615-622. Polytechnic Institute of Brooklyn, 1962.

[21] J. Lee, R. Machiraju, H. Pfister, and B. Moghaddam, "Estimation of 3D Faces and Illumination from Single Photographs Using a Bilinear Illumination Model", *Eurographics Symposium on Rendering Techniques*, June 2005.

[22] Michael Jones and Paul Viola, "Fast Multi-view Face Detection". *Mitsubishi Electric Research Laboratories, Inc.*, Cambridge, Massachusetts, 2003.

[23] Michael Jones and James Rehg, "Statistical Color Models with Application to Skin Detection". *Cambridge Research Laboratory, Compaq Computer Corporation,* One Kendall Square, Bldg 700*,* Cambridge, MA 02138, 1999.

[24] Yihu Yi, Daokui Qu, and Fang Xu, "Face detection method based on skin color segmentation and eyes Verification". *International Conference on Artificial Intelligence and Computational Intelligence,* Shanghai, China, November 2009.

[25] Ahmed Elgammal, Crystal Muang, and Dunxu Hu, "Skin Detection - a Short Tutorial". *Department of Computer Science, Rutgers University*, Piscataway, NJ, 08902, USA, 2009.

[26] Filipe Tomaz, Tiago Candeias, and Hamid Shahbazkia, "Improved Automatic Skin Detection in Color Images". *Universidade do Algarve, FCT Campus de Gambelas* - 8000 Faro, Portugal, 2003.

[27] Pham Minh Tri, *PyOpenCV.*
http://code.google.com/p/pyopencv/, 2010.

[28] Irfan Skiljan, *Irfanview*.
http://www.irfanview.com/, June 16, 2009.

[29] Guillaume Dargaud, *ImDim*.
http://www.gdargaud.net/Hack/Utils.html#ImDim, January 1, 2009.

[30] Peter N. Belhumeur, Jo~ao P. Hespanha, and David J. Kriegman, “Eigenfaces vs. Fisherfaces: Recognition
Using Class Specific Linear Projection”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 7, JULY 1997.

Appendix

Appendix A – PHP, Python, Java Code

Index.php

```
<?php
////////////////////////////////////////////////////////////////
/*************************************************************\
/*    Brian Bullard -- index page -- July 28, 2010          *\
/*                                                          *\
/*   AutoTagIt application for the facebook framework.      *\
/*   Requires php file Fbookhand to generate session        *\
/*   and facebook connection object.                        *\
/*                                                          *\
/*   Department of Computer Science and Informations Systems  *\
/*       University of North Carolina Wilmington           *\
/*************************************************************/
////////////////////////////////////////////////////////////////
 try{
      include_once('C:/wamp/www/Fbookhand.php');
                    session_start();

                    $autotagit = new autotagit();

                    $autotagit->createFBSession();
                    $_SESSION['autotagit'] = $autotagit;

   }
   catch(Exception $o){
      echo '<pre>';
      echo $o;
      echo '</pre>';
   }

/*For testing and demo purposes, provide a button to walk through NewUser branch
  and one for ExistingUser branch */
if(isset($_POST['NewUserTest'])){
          header ("Location: /public/terms.php"); // redirect to terms and conditions page
}
if(isset($_POST['notest'])){
   header ("Location: /public/logginsuccess.php");
}

echo '<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:fb="http://www.facebook.com/2008/fbml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
</head>
<div>';
```

```
if(isset($_SESSION)){
	echo '<form method="post">
	<input name="NewUserTest" type="submit" value="New User Test" >
	<input name="notest" type="submit" value="Current User" >
	</form>';
}
else {
	echo "Please login to Facebook to continue to the Autotagit app.";
}
echo '
</div>
</html>';
?>
```

Fbookhand.php

```
<?php

class autotagit{
	//Using setting for AutoTagItAlt

	function createFBSession(){
		try{
		include_once "facebook.php";
		$fbconfig['appid'] = '142718692417307';
		$fbconfig['api'] = '7231a4e012f12620379557eea3bc236c';
		$fbconfig['secret'] = '******************************';
		}
		catch (Exception $e2){ echo $e2;}

		try{ //Something new every day!
		//Do the Facebook wave!
		$facebook = new Facebook( array(
			'appId'  => $fbconfig['appid'],
			'api'    => $fbconfig['api'],
				'secret' => $fbconfig['secret'],
				'cookie' => true,));


		//Create a session
		$Session = $facebook->getSession();

		//Get permissions from the user
		if(!$Session){
			$datas = array(
			'canvas' => 1,
			'fbconnect' => 0,
			'req_perms'=>'user_photos,publish_stream,offline_access');
			$URL = $facebook->getLoginUrl($datas);
			echo '<fb:redirect url='.$URL.' />';
```

```
		}

			$_SESSION['UserID'] = $facebook->getUser();
			$_SESSION['logoutUrl'] = $facebook->getLogoutUrl();
			$_SESSION['loginUrl'] = $facebook->getLoginUrl();

			$handle = @fopen("C:\wamp\www\ipaddr.txt", "r");
			$ip = stream_get_line($handle,1024,"|");
			$_SESSION['ipaddr']= $ip;

		} catch (Exception $e){
			echo $e;
		}
	}//end createFBSession

	//Establishes a connection to Autotagit MySQL Database
	function establishDBConnection(){
		$dbhost = 'localhost';
		$dbuser = '*****';
		$dbpass = '*******';

		$conn = mysql_connect($dbhost,$dbuser,$dbpass) or die ('Error connecting to
mysql');
		$dbname = 'Autotagit';
		mysql_select_db($dbname) or die(mysql_error());
	}

	function getFB(){
		try{
		include_once "facebook.php";
		$fbconfig['appid'] = '142718692417307';
		$fbconfig['api'] = '7231a4e012f12620379557eea3bc236c';
		$fbconfig['secret'] = '**********************';
		}
		catch (Exception $e2){ echo $e2;}

		try{
		$facebook = new Facebook( array(
			'appId'  => $fbconfig['appid'],
			'api'    => $fbconfig['api'],
				'secret' => $fbconfig['secret'],
				'cookie' => true,));

		//getSession will return the current session if established
		$Session = $facebook->getSession();
		}catch (Exception $e2){ echo $e2;}
		return $facebook;
	}

	//this is how we can add a tag to an image ... old rest API still exists!
	function callFb($url, $params)
	{
		$ch = curl_init();
```

```
		curl_setopt_array($ch, array(
			CURLOPT_URL => $url,
			CURLOPT_POSTFIELDS => http_build_query($params),
			CURLOPT_RETURNTRANSFER => true,
			CURLOPT_VERBOSE => true
		));

		$result = curl_exec($ch);
		curl_close($ch);
		return $result;
	}

}// end class autotagit
?>
```

---

navLinks.php

```
<?php
class NavLinks{
	public static function CreateNavLinks(){

			/* Creates a set of navigation buttons for the AutoTagIt site header */

			$auto = new autotagit();
			$autoDBcon = $auto->establishDBConnection();
			$Session = $auto->createFBSession();
			$ipaddr = $_SESSION['ipaddr'];

			$user_face_defined = false;
			$result = mysql_query("SELECT count(auto_user_id) From auto_eigenface Where
auto_user_id = ".$_SESSION['UserID']."") or die(mysql_error());
			$num_eigs = mysql_fetch_array($result);

			//As a way of directing new users, use a grey image for the "Upload Images" and
"Review and Submit" buttons.
			if($num_eigs > 3)
				$user_face_defined = true;

			$ipaddr = $_SERVER['SERVER_NAME'];
			echo '<br/><br/>';

			echo '<a href="/autotagitalt/defineyourface.php"><img
src="http://'.$ipaddr.':8080/public/images/DefineYouModBrightLines.png" title="Define You" width=100
height=100></a>';
			if($user_face_defined){
				echo '<a href="/autotagitalt/uploadimages.php"><img
src="http://'.$ipaddr.':8080/public/images/UploadStart.png" title="Upload Images" width=100 height=100
/></a>';
				echo '<a href="/autotagitalt/reviewandsubmit.php"><img
src="http://'.$ipaddr.':8080/public/images/ReviewAndSubmit.png" title="Review And Submit" width=100
height=100 /></a>';
```

```
                        }
                        else {
                                echo '<a href="/autotagitalt/uploadimages.php"><img
src="http://'.$ipaddr.':8080/public/images/UploadStart_grey.png" title="Upload Images" width=100 height=100
/></a>';
                                echo '<a href="/autotagitalt/reviewandsubmit.php"><img
src="http://'.$ipaddr.':8080/public/images/ReviewAndSubmit_grey.png" title="Review And Submit" width=100
height=100 /></a>';
                        }
                        echo '<a href="/autotagitalt/removeme.php" ><img
src="http://'.$ipaddr.':8080/public/images/removemedissolve.png" title="Remove Me ::: Into the Abyss!"
width=100 height=100 /></a>';
                        echo '<br/><br/>';
        }
}
?>
```

Adduser.php

```
<?php
include('C:/wamp/www/Fbookhand.php');

try {

  /*Check to see if facebook id exists in DB
        Add user if ID does not exist in DB */
        session_start();
        $autotagit = $_SESSION['autotagit'];
        $UserID = $_SESSION['UserID'];

        $autotagit->establishDBConnection();

        $result = mysql_query("SELECT Auto_User_ID From Auto_User Where
        Auto_User_ID = ".$UserID."") or die(mysql_error());
                $line = mysql_fetch_array($result);

         if ($line[0] == $UserID){
                echo "</br>";
                echo "It appears that we already have you in our database!";
                echo "</br>";
        }
        else {
                mysql_query("INSERT INTO Auto_User (Auto_User_ID)
                VALUES ('".$UserID."')") or die(mysql_error());

                echo "</br>";
                echo "We have added you as a user to our system! Congrats!";
                echo "</br>";
        }

} catch (Exception $e) {
```

```
   echo 'Caught exception: ',  $e->getMessage(), "\n";
}

//Print message and show OK button to redirect to loginsuccess
?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:fb="http://www.facebook.com/2008/fbml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
</head>
<div>
<form action="/autotagitalt/logginsuccess.php" >
<input type= "submit" value= "Ok, thanks!" style= "width: 100px; color: #6699FF;" >
</form>
</div>
</html>
```

LoginSuccess.php

```
<?php

include_once('C:/wamp/www/Fbookhand.php');
include_once('C:/wamp/www/public/navLinks.php');
include_once('C:/wamp/www/facebook.php');
$home = 'C:/wamp/www/';

session_start();
$autotagit = $_SESSION['autotagit'];

//Check to see if the user is already in the database
$autotagit->establishDBConnection();

$result = mysql_query("SELECT Auto_User_ID From Auto_User Where Auto_User_ID = ".$_SESSION['UserID']."") or
die(mysql_error());
$U = mysql_fetch_array($result);

$wegottherightone = false;

if($U[0]==$_SESSION['UserID']){//U[0] should always be just one return
        $wegottherightone = true;
}

if ($wegottherightone == true){
        //Display Menu
        if(isset($_SESSION)){
                navLinks::CreateNavLinks();
        }
}
else{
                echo ("This was only a simulation, had you taken the red pill
                        your mind would be free.... (And you would now be a member
```

```
                                of Autotagit)");
}

echo '<br/><br/>Welcome!<br/><br/> ';
echo '<img src="https://graph.facebook.com/'.$U[0].'/picture"/>';
echo '<br/>
Please use the buttons above to navigate through the application.<br/>
If this is your first time here then please use the "Define You" button<br/>
to enroll your face into the AutoTagIt database.  We will use a representation<br/>
of your face to Auto-Tag images through the "Upload Images" page.';

//get the user's set of friends
$facebook = $autotagit->getFB();
$friends = $facebook->api('/me/friends');
if($friends){
        $friends = $friends['data'];
}

//get the user's friends as defined in autotagit database
$result = mysql_query("SELECT fb_friend_id From fb_friend Where auto_user_id = ".$_SESSION['UserID']."") or
die(mysql_error());
$count = 0;
$auto_friends = NULL;
while($row = mysql_fetch_array($result)){
        $auto_friends[$count] = $row;
        $count = $count + 1;
}

//if a Facebook friend of the user has been enrolled in the database then update the friends table
for($i=0;$i<sizeof($friends);$i++){
        $friends[$i] = $friends[$i]['id'];
        $result = mysql_query("Select Auto_User_ID from auto_user where Auto_User_ID = ".$friends[$i]."") or
die(mysql_error());
        $current_auto_user = mysql_fetch_array($result);
        if($current_auto_user){
                $exists = false;
                for($j=0;$j<sizeof($auto_friends);$j++){
                        if($auto_friends[$j]['fb_friend_id']==$friends[$i])
                                $exists = true;
                }
                if($exists==false){
                        $qstr = 'Insert into fb_friend (auto_user_id,fb_friend_id) values
(\''.$U[0].'\',\''.$friends[$i].'\')';
                        mysql_query($qstr) or die(mysql_error());
                }
        }
}
?>
```

Terms.php

```
<?php
include('C:/wamp/www/Fbookhand.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
<style type="text/css">
.style1 {
	margin-left: 0px;
}
.style2 {
	text-align: center;
}
</style>
</head>

<div class="style2">
		<textarea name="Terms_of_Use" style="width: 557px; height: 227px; display: table-caption;
font-size: x-small" rows="1" cols="20">
The AutoTagIt application, created for users of Facebook, is designed to reduce
the amount of time a user spends tagging images on Facebook.

Our application works by detecting Faces in images and attempting to match your
face and the faces of your friends on Facebook who have also signed up for the
application.  Once the application has finished tagging images, the user will
have the option to correct/deny tags and submit to a Facebook album.  Sometimes
automated systems do not produce the correct results.  We ask that you take a
small amount of time to review the tags before submitting them to a Facebook album.
No personal information other than your Facebook User ID and a small set of face
initialization images will be maintained on our system.
After accepting these terms, a correct login to Facebook will grant you access
to the application.

Images that have been uploaded to our server that are destined for Facebook albums
will not be maintained more than 7 days.

The author of this application and the University Of North Carolina Wilmington
are not responsible for lost information, including images, from the use of this
software.  Please be aware that the system will not be 100% accurate.  We strongly
suggest that you closely review the preview tags before submitting to Facebook.

Please keep backup copies of your images!

By selecting accept, you agree to these terms and will become part of the
AutoTagIt community.  Enjoy!
</textarea><br>
	<div style= "width: 200px">
```

```
		<form method="" action="/autotagitalt/adduser.php">
			<input name= "Accept" type= "submit" value= "Accept" style= "width: 59px; color: #6699FF;" >
		</form>
		<form method="" action="<?php echo '<a href='.$logoutUrl.'>';?>">
			<input name= "Decline" type= "submit" value= "Decline" style= "width: 60px; color: #6699FF;"
class="style1" >
		</form>
		</div>
 <br >
		  
</div>
</html>
```

Defineyourface.php

```
<?php
include_once('C:/wamp/www/Fbookhand.php');
include_once('C:/wamp/www/public/navLinks.php');
navLinks::createNavLinks();
$home = 'C:/wamp/www/';
session_start();

$ip = $_SESSION['ipaddr'];
$uid = $_SESSION['UserID'];

//create an iframe so we can use the applet tag ... not pretty!
echo '<fb:iFrame src="http://'.$ip.':8080/public/dyfiframe2.php?ipaddr='.$ip.'&uid='.$uid.'"
smartsize="true"></fb:iFrame>';
?>
```

---

dyfiframe2.php

```
<?php
include_once("C:/wamp/www/Fbookhand.php");

$uid = $_GET['uid'];
$ipaddr = $_GET['ipaddr'];
if(!isset($_POST['updone'])  and !isset($_POST['chosen'])){
echo "<p>Please select one or a few images that include your
 <br/>face as one of the main subjects.  Other faces can be
 <br/>present in the image.  We will take care of that later.
 <br/>Once uploaded, the images will be scanned for faces.
 <br/>The results will be displayed here for your approval.</p>";
 /*Inserts applet object into the web page -- accounts for IE and
   Firefox - Chrome - Safari browsers*/
 //Edit: firefox does not handle the java applet
```

```
echo '
<object
 classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
 height=40 width=200
 codebase="http://java.sun.com/products/plugin/autodl/jinstall-1_5-windows-i586.cab#Version=1,5,0,0">
 <param name=code value="ATUP.class">
 <param name=archive value="ATUP.jar">
 <param name="type" value="application/x-java-applet">
 <param name="uid" value="'.$uid.'">
 <param name="ip" value="'.$ipaddr.'">
 <param name="isFace" value="true">
  <!--the following adds support for Mozilla-like browsers-->
        <comment>
   <embed java_code="ATUP.class" archive="ATUP.jar"
    type="application/x-java-applet;jpi-version=1.5.0"
                uid="'.$uid.'"
                ip="'.$ipaddr.'"
                isFace="true">
   </embed>
  </comment>
 </object>';
echo "<p>Once you see the message \"Upload Complete!\" click \"Done\"</p>";
echo '
<form action="" method="POST">
<input name="updone" type="submit" value="Done">
</form>
';
}

/* This section only appears after the "Done" button is clicked.
        Photos from the uploader applet have been stored in the Database.
        A call to FindAndExtract will extract any faces from the images
        selected from the DB given a userID.  The skin detection process
        is called from within the python environment and is used to more
        accurately collect faces. */
if (isset($_POST['updone'])){
        $execString = 'C:/python26/python.exe C:/wamp/www/FindAndExtract.py '.$uid.' define';
        `$execString`;
        echo "Let us know who you are.  Please select the images of your face.";
        $auto = new autotagit();
        $autoDBcon = $auto->establishDBConnection();

        $result = mysql_query("SELECT image_data,auto_user_id,photo_id From photo
                                                        Where is_define_you = 1 and is_just_face = 1 and
Auto_User_ID = ".$uid."") or die(mysql_error());
        $rowCount = mysql_num_rows($result);

        $imgNames = array();

        /*Create a set of temporary images on the file system
         *PHP does not like the images straight from MYSQL */
        for($i=0;$i<$rowCount;$i++){
                $dat = mysql_fetch_row($result);
```

```
                $imgNames[$i] = $dat[2];
                $img = imagecreatefromstring($dat[0]);
                if ($img)
                        imagejpeg($img,'tempImages/t'.$uid.'_'.$dat[2].'.jpeg');
        }

        /*Display the images in a form with a check box next to each*/
        echo '<form method="POST" action="">';
        for($i=0;$i<$rowCount;$i++){
           echo '<img src="tempImages/t'.$uid.'_'.$imgNames[$i].'.jpeg" >';
                echo '<input type="checkbox" name="'.$imgNames[$i].'" value="'.$imgNames[$i].'" />';
                echo '     ';
                if($i%5==0 and $i!=0)
                        echo '<br/>';
        }
        echo '<input type="submit" name="chosen" value="Define Me" />';
        echo '</form>';
}

/*
        This section only appears after the user clicks the "Define Me" button.
        Original images have been deleted from the Database and only the
        face images exist.
*/
 if (isset($_POST['chosen'])){
                if ($dh = opendir('tempImages/')) {
                        while (($file = readdir($dh)) !== false) {
                           $regex = '/t'.$uid.'*/';
                                if(preg_match ($regex,$file)){
                                        unlink('tempImages/'.$file);
                                }
                        }
                        closedir($dh);
                }

                $auto = new autotagit();
                $autoDBcon = $auto->establishDBConnection();
                $result = mysql_query("SELECT image_data,auto_user_id,photo_id From photo
                                                        Where is_define_you = 1 and is_just_face = 1 and
Auto_User_ID = ".$uid."") or die(mysql_error());
                $rowCount = mysql_num_rows($result);
                $imgNames = array();
                for($i=0;$i<$rowCount;$i++){
                        $dat = mysql_fetch_row($result);
                        if(isset($_POST[$dat[2]])){
                                //Make a call to python script to convert image to eigenface
                                $execString = 'C:/python26/python.exe C:/wamp/www/AddToEigfaces.py
'.$uid.' '.$dat[2];
                                $output = shell_exec($execString);
                        }
                }
                //Delete temporary face images
```

```
                    mysql_query('Delete from photo where Auto_User_ID ="'.$uid.'" and is_just_face = 1 and
is_define_you = 1') or die(mysql_error());
                    echo "Thank you for showing us who you are!";
                    echo "<br/>";
                    echo "Please use the links above to start uploading your photos.";
 }//end if chosen
?>
```

UploadImages.php

```
<?php
include_once('C:/wamp/www/Fbookhand.php');
include_once('C:/wamp/www/public/navLinks.php');
navLinks::createNavLinks();

$home = 'C:/wamp/www/';
session_start();

$ip = $_SESSION['ipaddr'];
$uid = $_SESSION['UserID'];

//Once again, an iframe is needed here to use the applet html tag
echo '<fb:iFrame src="http://'.$ip.':8080/public/uploadiframe.php?ipaddr='.$ip.'&uid='.$uid.'"
smartsize="true"></fb:iFrame>';
?>
```

UploadIframe.php

```
<?php
include_once("C:/wamp/www/Fbookhand.php");
$uid = $_GET['uid'];
$ipaddr = $_GET['ipaddr'];
if(!isset($_POST['updone'])  and !isset($_POST['chosen'])){
echo '<p>Select a set of images for Auto-tagging.  You will have the option of<br/>
      to select one of your pre-existing facebook albums or to create a new one.<br/>
                    The current implementation only handles auto-tagging new photos but may soon be<br/>
                    expanded to existing facebook photos and albums.';
 echo '<br/>';
 echo '
 <object
  classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
  height=40 width=200
  codebase="http://java.sun.com/products/plugin/autodl/jinstall-1_5-windows-i586.cab#Version=1,5,0,0">
  <param name=code value="ATUP.class">
  <param name=archive value="ATUP.jar">
  <param name="type" value="application/x-java-applet">
  <param name="uid" value="'.$uid.'">
  <param name="ip" value="'.$ipaddr.'">
  <param name="isFace" value="false">
```

```
  <!--the following adds support for Mozilla-like browsers-->
        <comment>
   <embed java_code="ATUP.class" archive="ATUP.jar"
    type="application/x-java-applet;jpi-version=1.5.0"
                uid="'.$uid.'"
                ip="'.$ipaddr.'"
                isFace="false">
   </embed>
  </comment>
 </object>';
echo '<br/>';
echo "<p>Once you have finished uploading your photos click \"Done\"</p>";
echo '
<form action="" method="POST">
<input name="updone" type="submit" value="Done">
</form>
';
}
if (isset($_POST['updone'])){
        #Find and Extract the faces in each image
        #Stored in DB with auto_user_id, photo_id, x,y location and w,h
        $execString = 'C:/python26/python.exe C:/wamp/www/FindAndExtract.py '.$uid.' default';
        `$execString`;

        //Call ProjectAndCompare.py
        /*Result should be a set of tags in temp tag table*/
        $execString = 'C:/python26/python.exe C:/wamp/www/ProjectAndCompare.py '.$uid;
        `$execString`;

        $auto = new autotagit();
        $autoDBcon = $auto->establishDBConnection();

        echo"We have a set of tags for the uploaded images.<br/>
                Please use the \"Review and Submit\" link above to submit the photos <br/>
           to a facebook album.";
}
?>
```

ReviewAndSubmit.php

```
<?php
include_once("C:/wamp/www/Fbookhand.php");
include_once('C:/wamp/www/public/navLinks.php');
navLinks::createNavLinks();

$autotagit = new autotagit();
$fb = $autotagit->getFB();
$Session = $fb->getSession();

$uid = $Session['uid'];
$autotagit->establishDBConnection();
```

```
$fb->setFileUploadSupport(true);
$ACCESS_TOKEN = $Session['access_token'];

	/*
	Not my favorite way to do this but we will go forth and conquer.
	Scan through the photo table to find all images that need to be uploaded to the facebook album.
	- For each photo:
		: find all tags that need to be added
		: save the image to disk (the part i don't care for)
		: upload the image to the AutoTagIt facebook album (auto-generated by facebook if album is not
specified)
			*** be sure to collect the image id that's returned from the facebook api call
		: add each tag to the image
		: delete image from disk with unlink
	*/
	//get list of images
	$result = mysql_query("SELECT photo_id,image_data,width,height From photo
							Where auto_user_id = ".$uid."
							and is_just_face = 0
							and is_define_you = 0") or die(mysql_error());

	$rowCount = mysql_num_rows($result);
	$imgNames = array();
	//save the images to disk
	for($i=0;$i<$rowCount;$i++){
			$dat = mysql_fetch_row($result);
			$imgNames[$i] = array($dat[0],$dat[2],$dat[3]);
			$img = imagecreatefromstring($dat[1]);
			if ($img)
				imagejpeg($img,'tempImages/t'.$uid.'_'.$dat[0].'.jpeg');
	}

	//Up to Facebook :)
	echo 'Hang tight! We\'re sending images to facebook now.';

	$albums = $fb->api('/me/albums');
	$aid = 0;
	foreach ($albums['data'] as $album) {
		if($album['name'] == 'AutoTagItAlt Photos'){
			$aid = $album['id'];
		}
	}

	$fbImageIDs = array();
	for($i=0;$i<sizeof($imgNames);$i++){
		$FILE_PATH = 'tempImages/t'.$uid.'_'.$imgNames[$i][0].'.jpeg';
		$args = array();
		$args['image'] = '@' . realpath($FILE_PATH);
		$data = $fb->api('/photos?access_token='. $ACCESS_TOKEN, 'post', $args);
		//get the id of the photo just uploaded... i smell an adventure!
		$PID = null;
		$photos = $fb->api('/'.$aid.'/photos?access_token='.$ACCESS_TOKEN);
```

```
		$last = sizeof($photos['data'])-1;
		$img = $photos['data'][$last];
		$str = $img['link'];
		$PID = substr($str ,strpos($str,'pid=')+4,strpos($str,'&id')-(strpos($str,'pid=')+4));

		$fbImageIDs[$i] =
array($PID,$imgNames[$i][0],$FILE_PATH,$imgNames[$i][1],$imgNames[$i][2]);
		echo'<br/>';
		print_r( $fbImageIDs[$i]);
		echo '<br/>';
	}

	//Delete Temporary images created on file system
	for($i=0;$i<sizeof($fbImageIDs);$i++){
		unlink($fbImageIDs[$i][2]);
	}

	//Gonna add us some tags
	//!! New Graph API cannot yet handle tags :: Use old REST API!!
	for($i=0;$i<sizeof($fbImageIDs);$i++){
		$result = mysql_query("SELECT auto_user_id,loc_x,loc_y From temp_tag
							Where from_photo_id = ".$fbImageIDs[$i][1]."") or
die(mysql_error());
		$rowCount = mysql_num_rows($result);
		for($j=0;$j<$rowCount;$j++){
			$fbimg = mysql_fetch_row($result);
			$x = intval(($fbimg[1]/$fbImageIDs[$i][3])*100);
			$y = intval(($fbimg[2]/$fbImageIDs[$i][4])*100);
			$imgid = $fbImageIDs[$i][0];
			$uidimgid = $uid.'_'.$imgid;
			$upstr =
"https://api.facebook.com/method/photos.addTag?pid=".$uidimgid."&tag_uid=".$fbimg[0]."&x=".$x."&y=".$y."&
access_token=".$ACCESS_TOKEN."&format=json";
			echo '<iframe style="display:none" src='.$upstr.' width=1 height=1></iframe>';
		}
	}

	//Delete the temporary images : they exist in facebook now!
	if ($dh = opendir('tempImages/')) {
			while (($file = readdir($dh)) !== false) {
			  $regex = '/t'.$uid.'*/';
				if(preg_match ($regex,$file)){
					unlink('tempImages/'.$file);
				}
			}
			closedir($dh);
		}


################################################################################
###########  Clean up the database: Remove temporary images  ##################
################################################################################
for($i=0;$i<sizeof($fbImageIDs);$i++){
```

```
        $result = mysql_query("Delete from temp_tag where from_photo_id =".$fbImageIDs[$i][1]."") or
die(mysql_error());
        $result = mysql_query("Delete from photo where photo_id = ".$fbImageIDs[$i][1]." and is_just_face=0
and is_define_you=0") or die(mysql_error());
}
echo 'You\'re images have been uploaded to facebook in the "AutoTagIt" album.
    <br/>So click that big Ol\' facebook button up there and go check it out.
          </br>You might be supprised!';
?>
```

Removeme.php

```
<?php
include_once("C:/wamp/www/Fbookhand.php");
include_once('C:/wamp/www/public/navLinks.php');
navLinks::createNavLinks();

$autotagit = new autotagit();
$fb = $autotagit->getFB();
$Session = $fb->getSession();

$uid = $Session['uid'];
$autotagit->establishDBConnection();

if(!isset($_POST['deleted'])){
echo '
<P>You are about to delete yourself from the AutoTagIt system.<br/>
Selecting "Delete Me" will remove any record of you in our database.<br/>
You will no longer be available as an AutoTagIt user when your friends<br/>
upload images through this application.
<br/>
<br/>
Are you sure about this decision?';
echo '<form action="" method="POST">
        <input type="submit" name="deleted" value="Delete Me!">
        </form>';
}
if(isset($_POST['deleted'])){
//Go through each table deleting any record containing auto_user_id = $uid
$result = mysql_query("Delete from auto_user
                                                    where auto_user_id = ".$uid."") or die(mysql_error());
echo "You just got DELETED... OH snap!";
}
?>
```

FindAndExtract.py

```
#!C:/Python26
from pyopencv import *
```

```
from PIL import Image
from StringIO import StringIO
import MySQLdb
import sys
import os
from pyopencv.pyopencvext import *
import SkinDetectionComponent


'''
1) Accept as an argument (from comand line) the UserID which "should" already be in the DB...
2) Find the set of images that have the is_define_you falg set...
3) Convert images to jpeg from binary in DB
4) Call method to find faces in each image
5) Create a separate image for each detected face and store in DB
6) Return "Done" message
'''
class FindAndExtract:
   def __init__(self):
      findfaces(sys.argv[1],sys.argv[2])

def findfaces(uid,isDefineYou):

   #Load NeuralNet when fixed
   #Load LUT for now
   #load the Lookup table from file -- we only need to do this once
   LUTfile = open("C:/wamp/www/skinsamples/Skin2DLUT.txt",'r')
   LUT = []
   for i in LUTfile:
      LUT.append(i.split())

   #Determin if this instance is to define a users face or to simply upload images
   defineYou = 0
   if isDefineYou == 'define':
      defineYou = 1

   conn = MySQLdb.connect (host = "localhost",
                  user = "*****",
                  passwd = "*******",
                  db = "autotagit")
   cursor = conn.cursor ()
   cursor.execute ("SELECT image_data,photo_id from photo \
               where auto_user_id ='"+uid+"' and \
               is_define_you = '"+str(defineYou)+"' and\
               is_just_face = '0'")


   imgs = []
   faces = []

   while (1):
      row = cursor.fetchone()
      if row == None:
```

```
            break
        imgs.append(row)
    cursor.close ()

    for row in imgs:
        imageOrig = None
        fromPhotoID = None
        image = None
        mat = None
        try:
            fromPhotoID = row[1]
            imageOrig = Image.open(StringIO(row[0]))
            image = imageOrig.convert("L")
            mat = Mat.from_pil_image(image)
        except Exception as ex:
            f = open('C:/wamp/www/TraceLog.txt','a')
            f.write(str(sys.exc_info()))
            f.close()

        caspath = "C:/wamp/www/Flipped1882cascade.xml"


        cscd = CascadeClassifier()
        cascade = cscd.load(caspath)


        minSize= Size2i(24,24)

        faces = cscd.detectMultiScale(mat, 1.05, 1, CascadeClassifier.DO_CANNY_PRUNING, minSize )
        for n in faces:
            #faceLocationString = faceLocationString + str(n.x)+" "+str(n.y)+" "+str(n.width)+" "+str(n.height)+" "
            try:
                pilface = imageOrig.crop((n.x, n.y, n.x + n.width, n.y + n.height ))
                pilresize = pilface.resize((50,50))

                out = StringIO()
                pilresize.save(out, "jpeg")
                data = out.getvalue()

                #make a call to SkinDetectionComponent.py  ***** Currently using look up table *****
                isFaceBySkin = SkinDetectionComponent.isFaceBySkinAmount(LUT,pilface,n.width,n.height)
                print isFaceBySkin

                if isFaceBySkin:
                    cursor = conn.cursor()
                    cursor.execute ("Insert into photo \

(auto_user_id,image_data,from_photo_id,width,height,xloc,yloc,is_just_face,is_define_you,date_inserted) \
                                values (%s,%s,%s,%s,%s,%s,%s,%s,%s,NOW())"
                                ,(uid,data,fromPhotoID,50,50,n.x,n.y,1,defineYou))
                    cursor.close()
            except Exception as ex:
                f = open('C:/wamp/www/TraceLog.txt','a')
```

```
            f.write(str(sys.exc_info()))
            f.close()

    #delete all non-face images that were used to define the user's face
    # --- skip this step for when a user is simply uploading photos ---
    if defineYou == 1:
        cursor = conn.cursor ()
        cursor.execute ("Delete from photo where auto_user_id ='"+uid+"' and is_define_you = '1' and is_just_face =
'0'")
        cursor.close()

    conn.close ()


FindAndExtract()
```

SkinDetectionComponent.py

```
import Image
import math
#import NeuralNetSkin

def isFaceBySkinAmount(LUT,img,w,h):

    data = img.load()
    isface = False;

    #IntImg = myIntegralImage(self,data,picWidth,picHeight)
    #Convert RGB to YCrCb colorspace
    YCrCb = []
    for i in range (w):
        YCrCb.append([[] for j in range(h)])
    for i in range(w):
        for j in range(h):
            p = data[i,j]
            Y =    0 + 0.299    * p[0] + 0.587    * p[1] + 0.114    * p[2]
            Cr = 128 - 0.168736 * p[0] - 0.331264 * p[1] + 0.5      * p[2]
            Cb = 128 + 0.5      * p[0] - 0.418688 * p[1] - 0.081312 * p[2]
            YCrCb [i][j]=(Y,Cr,Cb)

    count = 0.0
    for i in range(w):
        for j in range(h):
            Y = YCrCb[i][j]
            if(LUT[int(Y[1])][int(Y[2])] == '1'):
                count = count + 1

    if count/(w*h) > 0.6:
        isface = True
    return isface
```

ProjectAndCompare.py

```
from numpy import *
import scipy
import sys
import Tkinter, ImageTk
import os,sys
from PIL import Image
import MySQLdb
from StringIO import StringIO
import Eigens
import scipy.spatial.distance
import HistEqImg
import operator
'''
This method is to be used at the command line from a PHP call.
Compares a newly introduced face to the current set of eigenfaces
that exist for an AutoTagIt user as well as to the user's friends.
'''

def project():
   uid = sys.argv[1]

   #Connect to the database
   conn = MySQLdb.connect (host = "localhost",
                           user = "*****",
                           passwd = "*******",
                           db = "autotagit")
   cursor = conn.cursor()

   #Select the set of eigenfaces as comparison set
   execstr = "SELECT auto_user_id, eigface_data from auto_eigenface \
                   where auto_user_id ="+uid+" or \
                       auto_user_id in (Select distinct fb.fb_friend_id \
                                   from fb_friend as fb where fb.auto_user_id \
                                   = "+uid+" ) \
           order by auto_user_id"

   #execute the query via MySQLdb cursor
   cursor.execute (execstr)

   OMEGAS = cursor.fetchall()
   cursor.close()
   '''eigfaces variable populated -- used later'''

   #select list of AutoTagIt friends
   cursor = conn.cursor()

   #Select the set of eigenfaces as comparison set
   execstr = "Select distinct fb_friend_id \
```

```
        from fb_friend \
        where auto_user_id = "+uid+" \
        order by auto_user_id"

#execute the query via MySQLdb cursor
cursor.execute (execstr)

FRIENDS = cursor.fetchall()
cursor.close()

#Load facespace and average face
stored_eigfaces = load('C:/wamp/www/eigens/eigfaces.npy')
avgFace = load('C:/wamp/www/eigens/avgface.npy')

########################################################################
#for each face in the set of detected faces for user X
#convert to greyscale / intensity map
#convert to eigenface representation
#compare to list of eigfaces
#find min cosine distance 'd' that meets requirements
# d < THETA_match   d < THETA_face : thresholds for max distance a
#new face can exist from a known face and average face, respectively
########################################################################
image = None
cursor = conn.cursor()
execstr = "Select photo_id,from_photo_id,xloc,yloc,image_data\
        from photo \
        where auto_user_id = "+uid+" and \
            is_define_you = 0     and \
            is_just_face = 1"
cursor.execute(execstr)
faces = cursor.fetchall()
#Cursor for DB is no longer needed ... close it
cursor.close()

#project avg face to facespace to obtain a threshold on face vs. non-face
avgOmega = dot(stored_eigfaces,array(avgFace))


#create a set of in-class averages, max distance to average of class, pair with auto_user_id
#eigenspace data for self and friends is stored in OMEGAS
#OMEGAS should be ordered by auto_user_id
inClassAvgs = []
inClassDat = []
classAvg = None
datMax = 0
#for each friend (a class) find class average and max distance (class boundary)
for friend in FRIENDS:
   auto_uid = friend[0]
   for bigO in OMEGAS:
      #if same user id, add facespace vectors to get average
      if(auto_uid == bigO[0]):
         inClassDat.append(bigO[1].strip('[]').split())
```

```
    if len(inClassDat) > 0:
        classAvg = asarray(inClassDat[0])
        #create the average
        for ICD in range(1,len(inClassDat)):
            classAvg = map(operator.add, float32(classAvg), float32(inClassDat[ICD]))
        for point in range(0,len(classAvg)):
            classAvg[point] = float32(classAvg[point]) / len(inClassDat)
        #find the max inclass distance from average
        for f in inClassDat:
            x = scipy.spatial.distance.pdist([classAvg,f], 'cosine')
            if  x > datMax:
                datMax = x
        inClassAvgs.append((friend[0],classAvg,datMax))
    datMax = 0
    classAvg = None
    inClassDat = []

#now we should have a set of in-class averages and the max-in-class distance
#compare the new faces to this set of class-average faces

tagArray = []

for face in faces:
    #convert to PIL image
    image = Image.open(StringIO(face[4]))
    #convert to greyscale
    image = image.convert("L")
    image = HistEqImg.HistEqByImgData(image)
    #Subtract face_i  from face_avg
    imgData = array(list(image.getdata()))
    phi = array(imgData) - array(avgFace)
    #Project onto facespace
    Omega = dot(stored_eigfaces,phi)


    ''' Find Max cosine distance from average face, any greater distance should be considered not-a-face.'''
    ''' Find the within-class averages and max distance from that average
            if the distance to any given in-class average greater than some Threshold (max in-class dist)
            then we have an unkown face -- apply no tag.
            If the face in question lies within the max distance to an in-class average
            the consider it as a match'''


    '''Within class averages can be calculated and stored in the DB later to save on computation'''
    min = 100000000000
    matchID = 0
    matchedMaxFromAvg = 0
    for cav in inClassAvgs:
        ica = cav[1]
        x = scipy.spatial.distance.pdist([Omega,ica], 'cosine')
        print x
        if x <= min:
            min = x
```

```
            matchID = cav[0]
            matchedMaxFromAvg = cav[2]
        #accumulate array of tag data to insert into temp_tags table
        #auto_user_id,from_photo_id,loc_x,loc_y
        print 'matchID: '+str(matchID) + '   distance: ' + str(min)+'   maxDistInclass: ' + str(matchedMaxFromAvg)
        #add tag to array if the face exists within a class boundary
        print str(min) +" < "+ str(matchedMaxFromAvg)+" ?"
        if matchID != 0 and min < matchedMaxFromAvg:
           tagArray.append([matchID,face[1],face[2],face[3]]);

    #insert accumulated tag data

    for t in tagArray:
        cursor = conn.cursor()
        cursor.execute("Insert into temp_tag\
            (auto_user_id,from_photo_id,loc_x,loc_y) \
            values (%s,%s,%s,%s)"
            ,(t[0],t[1],t[2],t[3]))

    #Cursor for DB is no longer needed ... close it
        cursor.close()
    conn.close()

project()
```

HistEqImg.py – (histogram processing function)

```
import sys
import os,sys
from PIL import Image
from PIL import ImageOps

def HistEqByPath(imgpath):
            img = Image.open(faceFileLocation)
            return HistEqByImgData(img)


def HistEqByImgData(img):
    return ImageOps.autocontrast(img, cutoff=5)
```

Eigens.py

```
from numpy import *
import scipy
import sys
import Tkinter, ImageTk
import os,sys
import math
from PIL import Image
```

```
w = 0
h = 0


'''Creates the set of Eigenfaces that comprise a face space
Two commandline modes exist for this class.
By calling "Eigens.py create S[int]", this class will create a set of eigenfaces of dimension S x S
Calling Eigens.py classify "path/to/face/image",this class will project a new image onto the face space

'''
class Eigens:


   def __init__(self):
      global w,h
      if len(sys.argv) > 1:
         if sys.argv[1] == "create":
            print "    Creating Eigenvalues and EigenVectors:"
            print "    Check location: "+os.getcwd()+"\\eigens\\"
            print "    for eigenvalue and eigenvector files"
            w = int(sys.argv[3])
            h = w
            createEigenFaces(sys.argv[2])

         elif sys.argv[1] == "classify":
            classifyFace(sys.argv[2])

      else:
         print "eigens python object created"

def createEigenFaces(path):
            global w,h
            imgVecAr = []
            meanVec = []

            #Convert all images to vector form
            imgVecAr = prepData(path)

            #Get the mean face image
            meanVec = getMean(array(imgVecAr))
            meanVecOut = os.getcwd()+"/eigens/avgface"
            save(meanVecOut,meanVec)

            #Subtract each face from the mean: "normalize each face"
            phivecs = createPhiVectors(imgVecAr,meanVec)

            #Calculate the covariance matrix from the normalized faces
            covmat = cov(phivecs)

            #Calculate the eigenvectors and eigenvalues
            eigens = linalg.eigh(covmat)
```

```
        #Choose K best vectors ( minus a few from the top )
        #let's say that K best is the top 80%
        K = int(len(eigens[0]) * 0.8)
        #K = int(len(eigens[0]) * 0.8)
        print "K= "+str(K)
        L = len(eigens[0])
        eigens = [eigens[0][L-K:L-3],eigens[1][L-K:L-3]]
        #eigens = [eigens[0][0:K],eigens[1][0:K]]

        #Write the eigenvals and vecs to a file for later use
        vectorsOut = os.getcwd()+"/eigens/eigvecs"
        save(vectorsOut,eigens[1])

        valuesOut = os.getcwd()+"/eigens/eigvals"
        save(valuesOut,eigens[0])

        #Create the Set of EigenFaces and write them to a file
        #eigfaces = dot(asarray(phivecs).transpose(),eigs[1])
        eigfaces = dot(eigens[1],asarray(phivecs))
        eigfacesout = os.getcwd()+"/eigens/eigfaces"
        save(eigfacesout,eigfaces)

        #scale the images back to 0-255 for viewing
        for i in range(0,len(eigfaces)):
                im = Image.new("L",(w,h))
                dat = (eigfaces[i]-eigfaces[i].min()) * (125/eigfaces[i].max())#edit scale to 0-125 for viewing
                #dat = eigfaces[i]/(eigfaces[i].max()/255.0)
                im.putdata(dat)
                im.save(os.getcwd()+"/eigens/eigfaces/eigFace"+str(i)+".jpg")



'''
Steps for Calculating Eigenfaces with PCA
Is data prepared?  Use ResizeImg.py and FaceFromImage.py if not.
Convert all images to vector form.
'''
def prepData(path):
   imgVecs = []
   for file in os.listdir(os.getcwd()+path):
      img = Image.open(os.getcwd()+path+file)
      imgData = array(list(img.getdata()))
      imgVecs.append(imgData)
   return imgVecs
   #test = numpy.ravel()



'''
Subtract the Mean: calculate the meanVector first
Then substract
'''
def getMean(imgs):
```

```
    global w,h
    #sum over the columns of the array of vector images
    meanVec = imgs.sum(axis=0)
    meanVec = meanVec / len(imgs)

    #This is just for display purposes only -- show the mean face
    im = Image.new("L",(w,h))
    im.putdata(meanVec)
    im.save(os.getcwd()+"/eigens/meanFace.jpg")
    return meanVec


''' Create a matrix of phi vectors. (Normalize the images)
        Subtract each image from the mean.
'''
def createPhiVectors(imgvecs,meanvec):
        phivecs = []

        for i in imgvecs:
                tempvec = i-meanvec
                phivecs.append(tempvec)

        return phivecs


''' We need a function to project a face
    onto a previously created face space
'''
def projectFaceFromFile(faceFileLocation):
    #print "Classifying face from file: "+faceFileLocation

    img = Image.open(faceFileLocation)
    imgData = array(list(img.getdata()))

    return projectFace(imgData)


def projectFace(imgData):
    eigfaces = load(os.getcwd()+"/eigens/eigfaces.npy")
    avgFace = load(os.getcwd()+"/eigens/avgface.npy")

    phi = array(imgData) - array(avgFace)

    Omega = dot(eigfaces,phi)
    for i in range(len(Omega)):
        Omega[i] = math.sqrt(Omega[i]*Omega[i])

    return Omega


Eigens()
```

AddToEigfaces.py

```
from numpy import *
import scipy
import sys
import Tkinter, ImageTk
import os,sys
from PIL import Image
import MySQLdb
from StringIO import StringIO
import Eigens

'''
This method is to be used at the command line by a call from PHP
Finds the set of stored face images for a user wishing to enroll their face
Into the database.
'''

def project():
   #Connect to the database
   conn = MySQLdb.connect (host = "localhost",
                           user = "******",
                           passwd = "*******",
                           db = "autotagit")
   cursor = conn.cursor()

   #create a query string that finds a certain images based on UserID and photo_id
   execstr = "SELECT image_data from photo \
                   where auto_user_id ="+sys.argv[1]+" and \
                   photo_id = "+sys.argv[2]

   #execute the query via MySQLdb cursor
   cursor.execute (execstr)


   #acquire image, check if exists, convert to intensity image
   image = None
   row = cursor.fetchone()
   if(row):
      imageOrig = Image.open(StringIO(row[0]))
      image = imageOrig.convert("L")
   else:
      print 'not getting an image out'

   #Cursor for DB is no longer needed ... close it
   cursor.close()


   imgData = array(list(image.getdata()))
```

```
    eigfaces = load('C:/wamp/www/eigens/eigfaces.npy')
    avgFace = load('C:/wamp/www/eigens/avgface.npy')

    phi = array(imgData) - array(avgFace)

    Omega = dot(eigfaces,phi)

    cursor = conn.cursor()
    cursor.execute ("Insert into auto_eigenface (auto_user_id,eigface_data) \
                values(%s,%s)",
                (sys.argv[1],str(Omega)))


    conn.close()

project()
```

---

createLookup2D.py

```
from numpy import *
import os,sys

def createLUT2D(data):
    #create a 2D array to be used as the lookup table
    LUT = zeros([256,256],int)

    #mark values in lookup table as 's' (skin) or '0' non-skin
    for i in data:
        row = i[0]
        col = i[1]
        LUT[row,col] = 1
    return LUT

#Use this option if data exists in sortedSkinSamplesCrCb.txt
def createLUT2DFromFile():
    f = open(os.getcwd()+"/SkinSamples/sortedSkinSamplesCrCb.txt")

    #create a temporary array to access individual parts
    points = []
    for i in f:
        points.append(i.split())
    f.close

    LUT = createLUT2D(points)

    #write this structure out to a file
    outfile = open(os.getcwd()+"/SkinSamples/Skin2DLUT.txt","w")
    for i in range(0,256):
        row = ""
        for j in range(0,256):
```

```
        row = row + str(LUT[i,j])+ " "
    outfile.write(row+"\n")
outfile.close()
```

SkinSelector.py

```
import Tkinter, ImageTk
import os,sys
import Image, ImageDraw

exitval = 0
imgdata = 0
buttonD = 0
w = 0
wstep = 11
hstep = 11
h = 0
skinsamples = []
SkinTrainingData = ''
lastevent = -1
fileExtensionList = ('.jpeg','.jpg','.png','.gif')

canvas = ''

''' This class is to be used as a skin data gathering tool.
The user is presented a set of images, one at a time. Left clicking on an image
will take the average pixel data from a region wstep * hstep and append that data to a text
file.  Space bar or any key other than Q will cause the next image to show.  Q quits the program.
Use createLookup2D.py to transform that data into the Lookup table used for skin classification.'''

class SkinSelect(object):

    def __init__(self):
        self.getTrainingData()


    def getTrainingData(self):
        global SkinTrainingData
        path = os.getcwd()
        SkinTrainingData = open(path+"/SkinSamples/skinSampleData.txt",'a')
        path=path+"/SkinSamples/"
        global exitval
        dirList=os.listdir(path)
        for fname in dirList:
            if exitval == 0 and self.compareFileExt(fname) == True:
                self.recordDataPoints(path+fname)
        SkinTrainingData.close()


    def recordDataPoints(self,fileStr):
        global imgdata,w,h,canvas
```

```
    root = Tkinter.Tk()
    root.focus_force()
    root.protocol("WM_DELETE_WINDOW", root.quit)
    pic = Image.open(fileStr)
    w = pic.size[0]
    h = pic.size[1]
    print w
    if pic.size[0]<1200:
        percent=1200.0/pic.size[0]
        rez = pic.resize((int(percent*w),int(percent*h)),Image.BICUBIC)
        w = rez.size[0]
        h = rez.size[1]
        pic = rez.copy()
    print w
    imgdata = pic.load()
    root.bind("<Key>", self.e1)
    root.bind("<Button-1>",self.getvals)
    root.bind("<B1-Motion>",self.getvals)
    root.bind("<Button-3>",self.getvals)
    root.bind("<B3-Motion>",self.getvals)
    root.geometry('+%d+%d' % (5,5))
    canvas =  Tkinter.Canvas(root,width=w,height=h)
    tkpi = ImageTk.PhotoImage(pic)
    canvas.create_image(0,0,image = tkpi,anchor=Tkinter.NW)
    canvas.pack()
    root.mainloop()
    root.destroy()

def drawOnImage(self,x,y):
    global canvas
    for i in range(wstep):
        x = x + (i - wstep/2)
        for j in range(hstep):
            y = y + (j - hstep/2)
            canvas.create_line((x,y,x+1,y+1),fill=("green"))
    canvas.update_idletasks()


def e1 (self,event):
    global exitval,skinsamples,SkinTrainingData
    if event.char == 'r':
        skinsamples = []
    elif event.char == 'q':
        SkinTrainingData.writelines(skinsamples)
        exitval = 1
        event.widget.quit()
    else:
        SkinTrainingData.writelines(skinsamples)
        skinsamples = []
        event.widget.quit()

def getvals(self,event):
    global skinsamples,w,h,canvas
```

```
        global lastevent
        if event.num == 1:
            lastevent = 1
        if event.num == 3:
            lastevent = 3

        R = 0
        G = 0
        B = 0
#Use wstep and hstep to get an average pixel over an area of pixels
        if lastevent == 3:
            if event.x < w and event.x > 0 and event.y >0 and event.y < h:
                for i in range(wstep):
                    for j in range(hstep):
                        x = event.x + i-wstep/2
                        y = event.y + j-hstep/2
                        pixel = imgdata[x,y]
                        R = R + pixel[0]
                        G = G + pixel[1]
                        B = B + pixel[2]
                R = R / (wstep*hstep)
                G = G / (wstep*hstep)
                B = B / (wstep*hstep)
                self.drawOnImage(x,y)
                skinsamples.append(str(R)+" "+str(G)+" "+str(B)+" ns\n")
        elif lastevent == 1:
            if event.x < w and event.x > 0 and event.y >0 and event.y < h:
                for i in range(wstep):
                    for j in range(hstep):
                        x = event.x + i-wstep/2
                        y = event.y + j-hstep/2
                        pixel = imgdata[x,y]
                        R = R + pixel[0]
                        G = G + pixel[1]
                        B = B + pixel[2]
                R = R / (wstep*hstep)
                G = G / (wstep*hstep)
                B = B / (wstep*hstep)
                self.drawOnImage(x,y)
                skinsamples.append(str(R)+" "+str(G)+" "+str(B)+" s\n")

    #Compares a file extension with the extensions
    #defined in a list  :: fileExtensionList ::
    #Make sure we are looking at an image file
    def compareFileExt(self, fileName):
        global fileExtensionList
        for i in fileExtensionList:
            print fileName.lower()
            print i.lower()
            if fileName.lower().endswith(i.lower()):
                return True
        return False
```

SkinSelect()

ResizeImg.py

```
import numpy
import scipy
import sys
import os
from PIL import Image

dim = int(sys.argv[1])

''' Resizes all image files in a directory to size "dim"
   and saves the resulting resized images in jpeg format.'''

imgPath = os.getcwd()+"/eigens/thumbs50x50/"
for file in os.listdir(os.getcwd()+"/eigens/thumbs50x50/"):
   img = Image.open(imgPath+file)
   img = img.resize((dim,dim),Image.BICUBIC)
   img.save(imgPath+file,"JPEG")
```

FaceFromImage.py

```
import numpy
import scipy
import sys
import os
from PIL import Image

'''
Extracts a faces that have location predefined in a text file with the following
format  filename count x1 y1 width1 height1 [x2 y2 width2 height2 .... if more than one face]

'''

#Open markup.txt, for each row -- split()
#token 0 will be file name, following tokens will be :face count:x1:y1:w1:h1:x2:y2....

markup = open(os.getcwd()+"/eigens/all/all.txt",'r')
imagePath = os.getcwd()+"/eigens/all/"
facesTXT = []
faces = []

#read all lines and store in facesTXT
for line in markup:
   facesTXT.append(line.split())
markup.close()
```

```
#creat a subset of face images extracted from the originals
for line in facesTXT:
   img = Image.open(imagePath+line[0])
   for i in range(0,int(line[1])):
      x = int(line[i*4+2])
      y = int(line[i*4+3])
      w = int(line[i*4+4])
      h = int(line[i*4+5])
      box = (x,y,x+w,y+h)
      tiny = img.crop(box)
      tiny = tiny.convert('L')
      tiny.save(os.getcwd()+"/eigens/thumbs50x50/thumb50x50_"+str(i)+"_"+line[0],"JPEG")
```

Java Image Upload Applet – Using JDBC for MySQL connection

```
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JPanel;
import java.sql.Connection;
import java.sql.DriverManager;
import java.awt.image.BufferedImage;
import javax.swing.JTextArea;
import java.io.*;
import java.util.Date;
import javax.imageio.ImageIO;

public class ATUP extends JApplet implements ActionListener{

   private static JFileChooser fc;
   private static JButton select;
   private static File [] filesproper;
   private static BufferedImage [] files;
   private static Container content;
   private static JPanel jp;
   private static JTextArea info;
   private static int uid = 0;
   private static String ipaddr = "127.0.0.1";
   private static boolean isFaceDefineImg = false;

   @Override public void init(){
      content = getContentPane();
      content.setBackground(new Color(0,200,245));
```

```
   fc = new JFileChooser();
   fc.setMultiSelectionEnabled(true);

   jp = new JPanel();
   jp.setLayout(new FlowLayout());

   select = new JButton("Select");
   select.setSize(new Dimension(20,10));
   select.addActionListener(this);
   jp.add(select);

   info = new JTextArea("");
   info.setSize(new Dimension(200,200));

   uid = Integer.parseInt(getParameter("uid"));
   ipaddr = getParameter("ip");
   isFaceDefineImg = Boolean.parseBoolean(getParameter("isFace"));

   jp.add(info);
   content.add(jp);
}//End init()

@Override public void start(){}
@Override public void stop(){}
@Override public void destroy(){}

public void actionPerformed(ActionEvent event) {
   if(event.getSource()==select){
      fc.showOpenDialog(null);
      filesproper = fc.getSelectedFiles();
      files = new BufferedImage [filesproper.length];

      /*Images can be very large
       *Reduce to a Max of 720x720 */
      for (int i =0; i<filesproper.length; i++){
         try{
            info.setText("Up: "+i+"/"+filesproper.length);
            BufferedImage buff = ImageIO.read(filesproper[i]);
            double width = 720.0 / buff.getWidth();
            double height = 720.0 / buff.getHeight();

            double factor = Math.min(width,height);

            AffineTransform tx = new AffineTransform();

            if (factor < 1){
               tx.setToScale(factor,factor);
            }
            else {
               tx.setToScale(1,1);
            }
            AffineTransformOp op = new AffineTransformOp (tx,null);
            buff = op.filter(buff, null);
```

```
            files[i] = buff;

          }
          catch(Exception e){
            writeTrace(e.getMessage());
          }
        }

        try{
          Connection conn = null;
          Class.forName ("com.mysql.jdbc.Driver");
          conn = DriverManager.getConnection("jdbc:mysql://"+ipaddr+":3306/autotagit",
               "*****","*******");
          BufferedImage buff;
          for (int i =0; i<files.length; i++){
            //BufferedImage buff = ImageIO.read(files[i]);
            buff = files[i];
            int width = buff.getWidth();
            int height = buff.getHeight();

            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ImageIO.write(buff,"jpeg", baos);

            java.sql.PreparedStatement ps =
                                                    conn.prepareStatement("insert into autotagit.photo "
              + "(auto_user_id,width,height,image_data,is_define_you,date_inserted)"
              + " values(?,?,?,?,?,NOW())");
            ps.setInt(1,uid);
            ps.setInt(2,width);
            ps.setInt(3,height);
            ps.setBytes(4,baos.toByteArray());
            if(isFaceDefineImg)
              ps.setBoolean(5,true);
            else
              ps.setBoolean(5,false);
            ps.executeUpdate();

            info.setText("Upload Complete!");
            //bais.close();
            baos.close();
          }//End for: add files to DB

          conn.close();
        }catch(Exception e){
          writeTrace(e.getMessage());
          info.setText("Upload Failed");
        }//End catch everthing
      }//End IF user hits upload button
    }//End ActionPerformed

    public void writeTrace(String message){
      try {
```

```
            FileWriter fw = new FileWriter("C:/wamp/www/TraceLog.txt",true);
            BufferedWriter buf = new BufferedWriter(fw);
            buf.write(new Date()+"   "+
                  message+
                  System.getProperty("line.separator"));
            buf.close();
         }catch(Exception e2){
            //Do nothing since we are writing errors out to a file that only
            //exists on the server.
         }
      }
   }//End class
```

Appendix B –Event Diagrams and Database Model

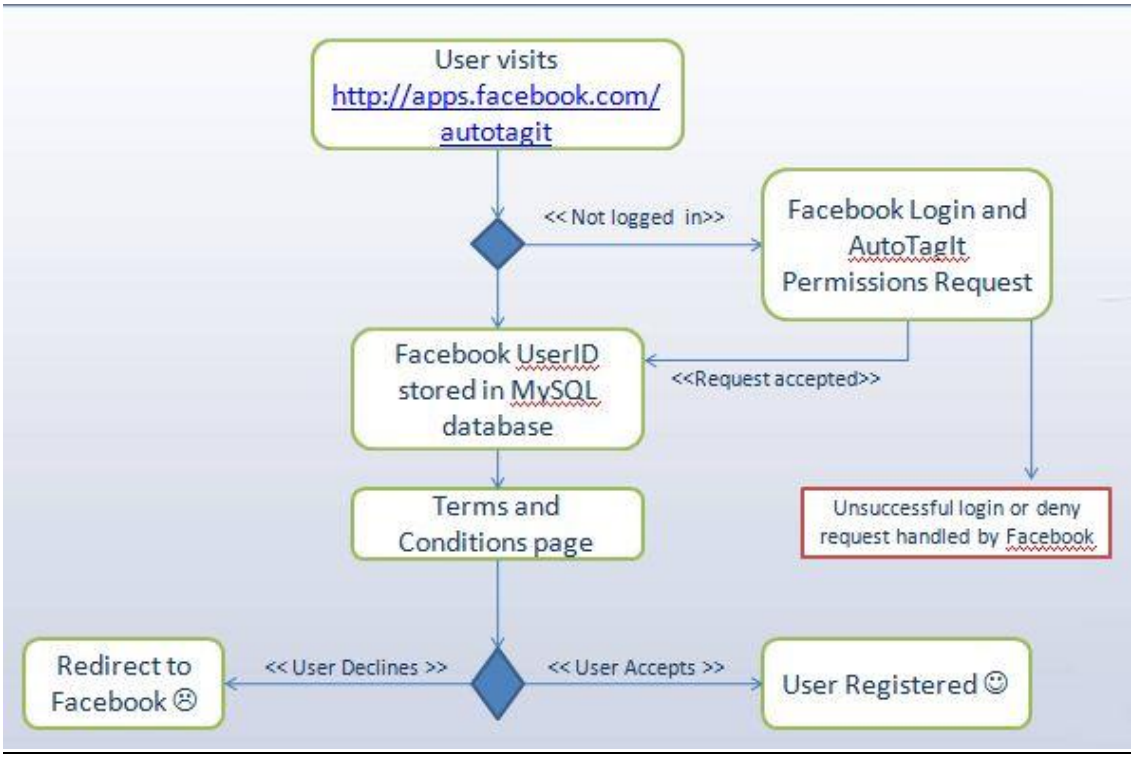

Figure 9 - AutoTagIt user registration

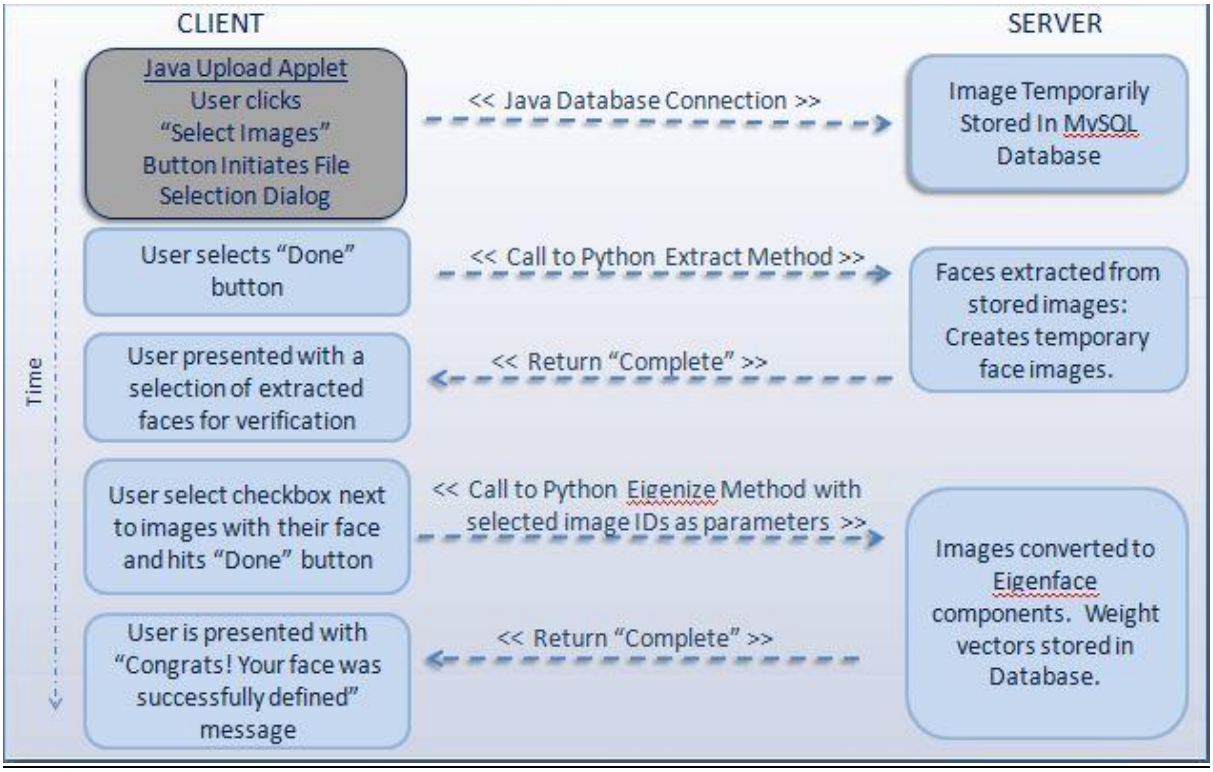

Figure 10 - AutoTagIt user face enrollment

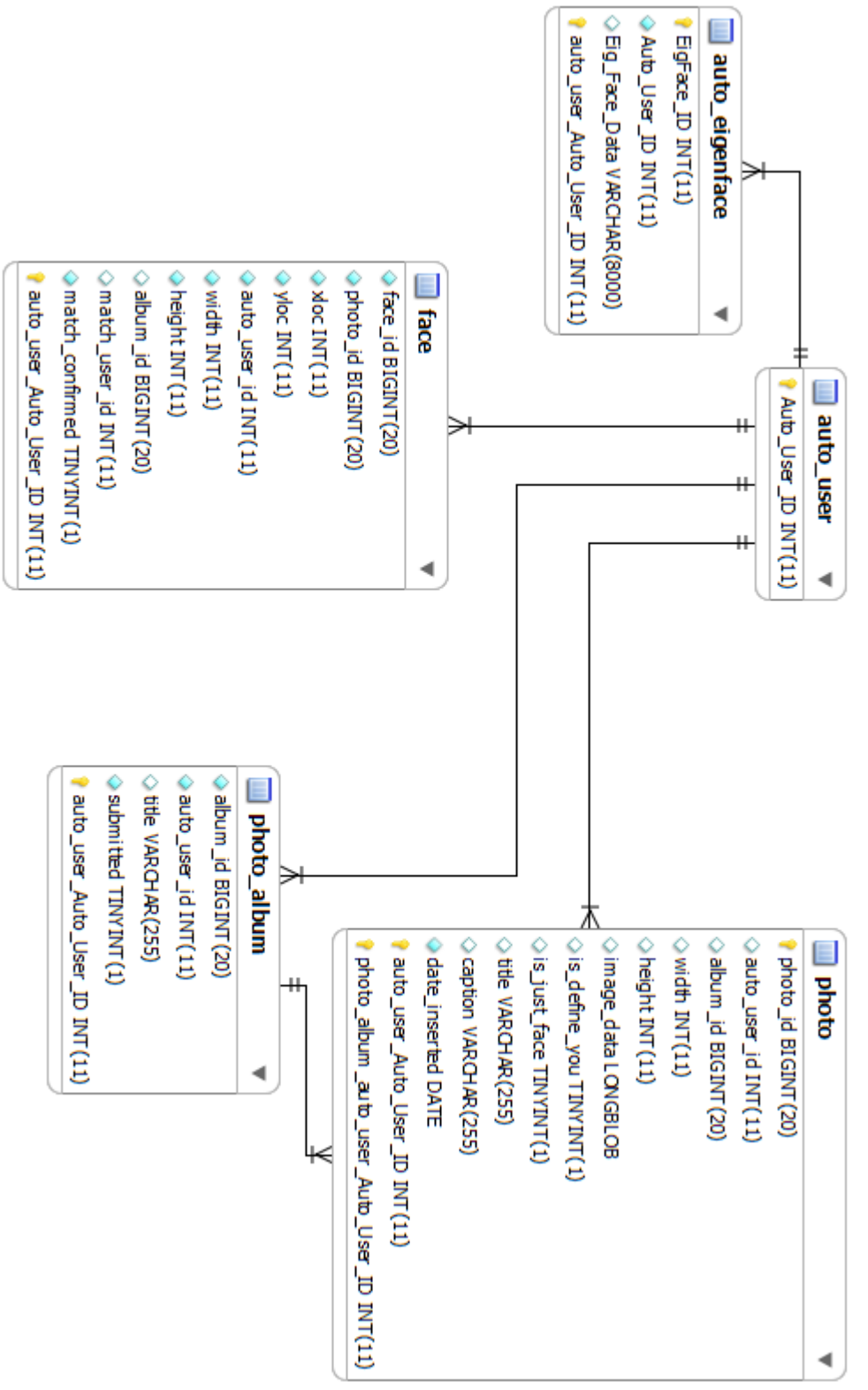

Figure 11 - AutoTagIt Database Model

Appendix C – Building the Cascaded Classifier of Haar-like features with OpenCV

Building a face detector with OpenCV:

The OpenCV package includes a toolset for creating face detectors using the principles found in the Viola-Jones method. The toolset consists of two executable files for creating general object detectors. One, "opencv_createsamples.exe" is a tool that allows you to manipulate a set of positive examples, in our case face examples, into a vector file of uniformly sized images. This vector file (.vec) is a binary file that is used by the second tool in the set "opencv_haartraining.exe" that performs the actual training of the haar classifier. The final product of the training is an XML file. Once created correctly, the XML files can be used as an argument to OpenCV's haarDetectObject() function.

To train a more robust classifer, a rather large set of sample images must be obtained. The number used in this study was 941. The location of each known positive face sample in all images must be documented in a text file in the following format. On each row of the text file there must be at least six arguments. 1) The name of the image file. 2) The number of face examples for this row. 3) The X coordinate of the top left corner of a bounding box around the face. 4) The corresponding Y coordinate to the top left corner. 5) The width of the bounding box. 6) The height of the bounding box. Subsequent arguments repeat the format of the last four.

Example file ... AllFlipped.txt

train0001.jpg 1 120 56 56 56
train0002.jpg 2 121 77 38 33 130 135 180 175
train0003.jpg 1 78 85 62 56

Along with a large set of positive object samples an even larger set of negative samples is needed. More than 4,000 negative samples were used to create the haar classifier used by this study. A second text file is needed for the negative samples that simply lists the names of the image files to be used and any necessary folder structering.

Example file ... NegativeSamples.txt

nightimages/night1132.jpg
nightimages/night1133.jpg
nightimages/night1134.jpg
sceneryimages/scenery0001.jpg
sceneryimages/scenery0002.jpg

The Opencv_createsamples.exe is an overlaoded executable whos functionality is based on a set of keyword arguments. There are four uses for this executable. First by specifying the -data and -vec arguments, the program will create an image vector file from many images. Second, if the -img, -bg, and -vec arguments are supplied then then program will create many samples from one by applying a configurable set of distortions to copies of the original image. The third use is to create test samples where faces are randomly distorted and inserted into negative sample images. This funciton is called by

supplying the -info, -bg, and -img arguments. The last and simplest function is to display the images stored in a vector file which is achieved by supplying only the -vec argument.

The following is a list of arguments and their associated meanings.
-img <path> - path to an image to be used to create samples from one
-vec <path> - path to the image vector to be created or read from
-info <path> - path to the text file listing positive image sample locations
-bg <path> - path to the text file listing negative image samples
-maxidev <int> - Maximum intesity deviation when applying distortions Range: 0-255
-maxxangle <float>- Maximum rotation on the x-axis in radians
-maxyangle <float>- Maximum rotation on the y-axis in radians
-maxzangle <float>- Maximum rotation on the z-axis in radians
-w <int>- The width of the resulting images in the vector file :: -show display width
-h <int>- The height of the resulting images in the vector file :: -show display height
-show -

An example of the command to create a vector file of trainging samples from a list of positive sample images is as follows. Note that it may be necessary to supply the full path to .txt and .vec files.

C:\> opencv_createsamples -info AllFlipped.txt -vec AllFlipped.vec -maxidev
0 -w 24 -h 24 -show

The opencv_haartraining.exe program has its own set of arguments that need to be supplied to generate a classifier.

-data - the file name to be used for the .xml haar-like feature classifier generated
-vec - the image vector file generated from the previously mentiond execuatble
-bg - the list of negatives used for training via the Adaboost algorithm
-nstages - maximum number of rounds for training as described in the Adaboost algorithm
-nsplits - maximun number tree splits
-minhitrate - minimum positive hit rate required to move to next round of training
-maxfalsealarm - maximum rate of false positive detection before moving to next round
-npos - number of positive samples
-nneg - number of negative samples
-w face image width (must correspond with width in .vec file)
-h face image height (must correspond with height in .vec file)
-nonsym - specified if training images are no symmetrical. Used to reduce time for training.
-mem - amount of memory in mega byte units to be used for training
-mode <ALL,BASIC> All uses all haar features including diagonals.
BASIC uses only upright feature set.

The following is an example of the command used for training. Note that it may be necessary to supply the full path to .txt and .vec files.

C:\>opencv_haartraining -data Flipped1882 -vec AllFlipped.vec -bg negatives.txt -nstages 20 -nsplits 2 -minhitrate 0.999 -maxfalsealarm 0.5 -npos 1882 -nneg 4000 -w 24 -h 24 -nonsym -mem 1024 -mode ALL

Invaluable documentation on using the OpenCV Haartraining tools can be found at Naotoshi Seo's tutorial: http://note.sonots.com/SciSoftware/haartraining.html
Naotoshi Seo has aslo created a tool that was used in this study during the process of generating a face detector called MergeVec.exe This tool takes two image vector files and combines them into one file to be used by the Haartrainning tool.

The collection of images used for training were gathered over a few different sources. The restriction on gathering images is inherent in Facebook's privacy architecture where only a "Friend" of a user is allowed to view that user's photos. While some images were collected in Facebook from a personal list of friends, the majority of training images were obtained from a combination of Flickr.com and the Labeled Faces in the Wild (LFW) data set. The users of Flickr.com previously started a community gallery of face images that closely match the definition of a face defined for this study. The LFW data set features only images of forward facing persons with a majority exhibiting sufficient lighting.

To locate the positions of the faces within the images, the IrfanView [28] image processing program was used. While other programs such as Microsoft Paint could suffice, IrfanView provides an easy to use area selection feature that displays the starting coordinates and the width and height of the area selected. These values were presented in just the format needed to build the -info description file used as an argument to the opencv_createsamples program. Irfanview also proved handy when the need presented itself to crop, rotate, resize, rename or convert the format of images in bulk.

With guidance from Naotoshi Seo's tutorial and the toolset previously presented, a number of Haar classifiers were created. The most impressive was genereated from the original set of 941 face images. With the opencv_createsamples utility a set of 5,000 face images was obtained by making 50 modified copies of 100 faces. The detection rate of this classifier performed much more poorly than the first. The decline in performance could easily be attributed to the lack of variance among the set of faces to train on. A third classifier performed just under the first which was a 50-50 split of face images including the entire original set and an equal number of faces selected those generated for the 5,000.

After testing the previous three detectors, a fourth was created after the realization that the orginal set could be doubled by flipping the images about the vertical axis and adjusting the description file accordingly. A special command line utility, ImDim.exe [29] created by Guillaume Dargaud allows a user to quickly find the widths of a set of JPG images. The set of widths of the original images was used to modify the description file for training. Credit should go to the developers at Microsoft for making this an easy task with Microsoft Excel where a text file could be imported and split into columns. We can use the following to mark the location of a face in a flipped image.

$\phi_i^{k'}\{x',y,w,h\}$

Where $\phi_i^{k'}$ is the location of face i in the image k flipped about the vertical axis, and x' is defined as follows.

$x' = W^k - x_i + w_i$

W^k is the width of image k and w is the width of face i. For image k, (x,y) are the top-left pixel coordinate of face i and w,h are the width and height of that face.

Appendix D – Web Application Screenshots

Figure 12 - Welcome Screen.

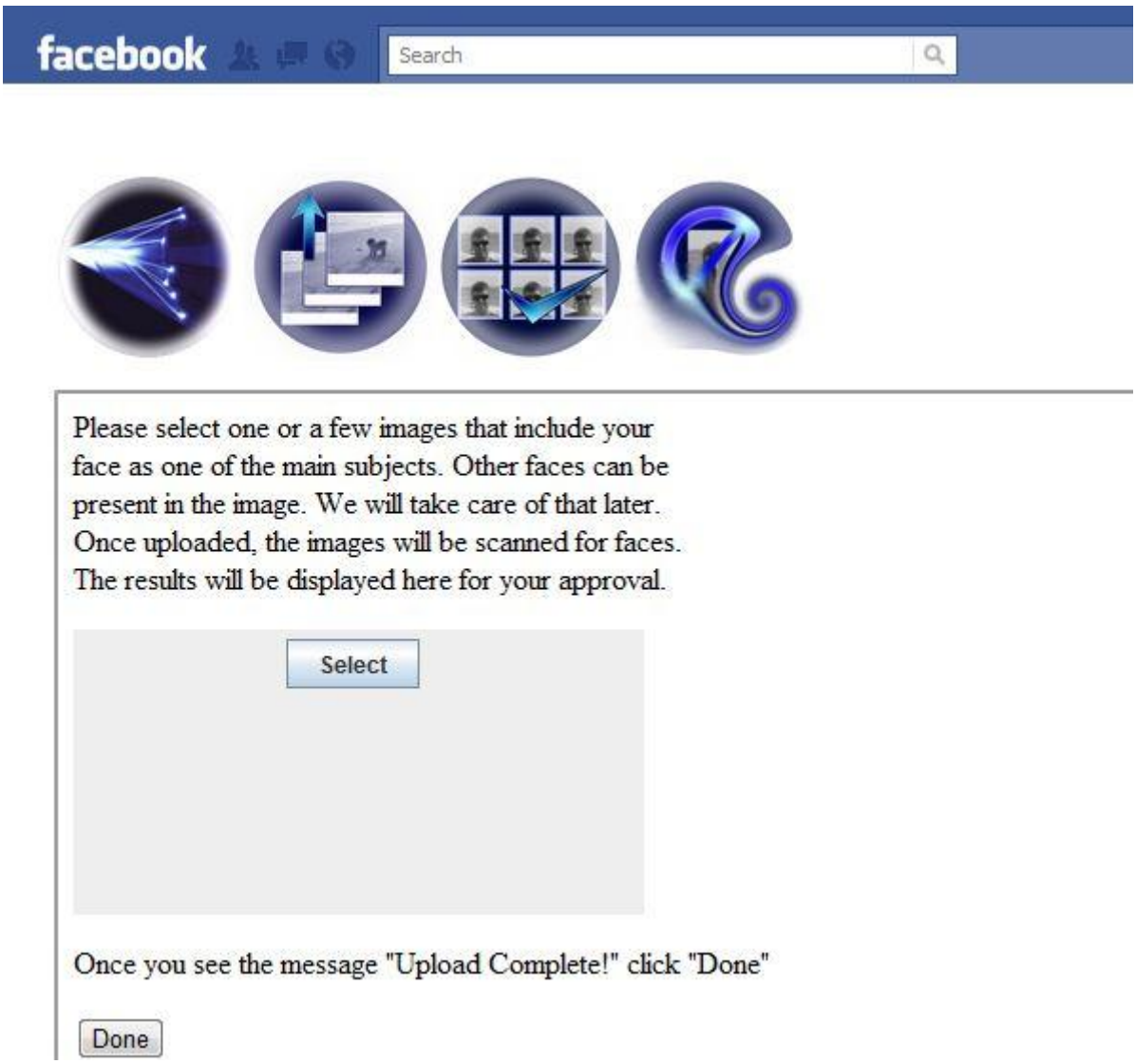

Figure 13 - "Define You page".

Figure 14 - Selecting images for the Define You page.

Figure 15 - User face verification for the Define You page.

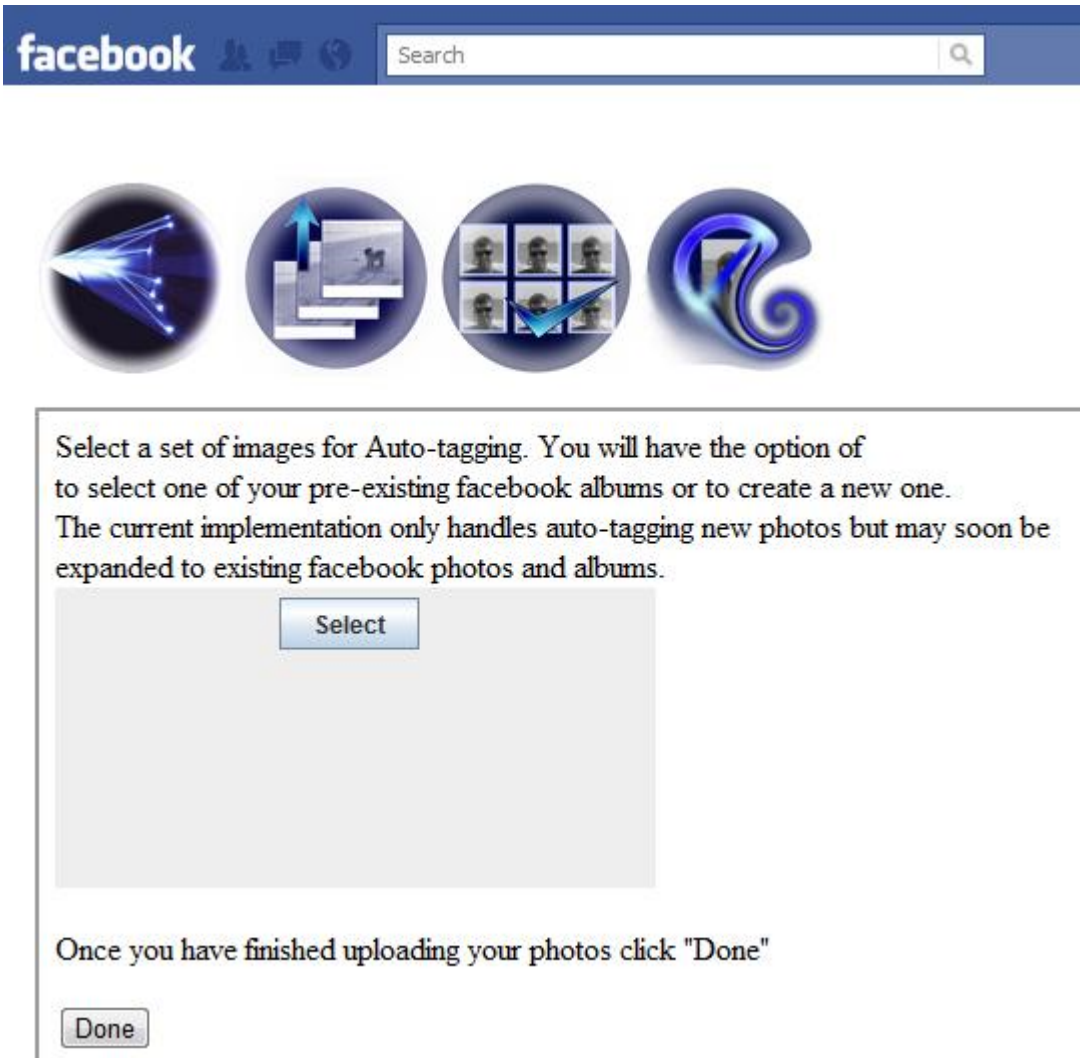

Figure 16 - The Upload Images page.

Figure 17 - Submit to Facebook page.

Figure 18 - Remove Me page.