COMS 4701 Artificial Intelligence
Final Project Report

Applications of Machine Learning to Facial Recognition in Video

David Lariviere
dal2103@columbia.edu

John Petrella
jep2124@columbia.edu

Jian Pan
jp2472@columbia.edu

ABSTRACT

Facial Recognition remains one of the most popular fields in computer imaging research due to its broad scope of applications in identity verification, human computer interactions, law enforcement applications and video surveillance systems. Although numerous artificial intelligence and machine learning techniques have been successfully applied to still image recognitions, recognition in video streams remains a difficult task due to the changing conditions present in the video data, such as lighting, pose, and image motion. This paper describes an application to solve the above problem with automatic image selection and normalization from video streams using OpenCV, and conducting recognition through training and testing using Machine Learning library WEKA.

1. Problem

Although seemingly obvious for the human eye, facial recognition remains a difficult yet desirable task for computer scientists. Over the past twenty years, advancements in various fields such as neuroscience, cognitive science, statistics, artificial intelligence, machine learning, computer vision and imaging have all contributed to the field. Due to its nature, facial recognition has been difficult to realize because apart from a human who can describe facial features, computers require a set of quantified features from which to identify, classify and further learn about different faces.

In recent years, several advancements in theory and techniques have appeared to address or overcome many of the important issues perplexing researchers in the field. Namely, face detection in complex backgrounds through neural-network based system [Rowley, Baluja and Kanade, 1998] and sample-learning based system [Colmenarez and Huang, 1997], the feature extraction of facial images through global descriptions based on Eigenface, KL expansion [Turk, Pentland, 1994] and local descriptors such as nose, eyes, hair regions.

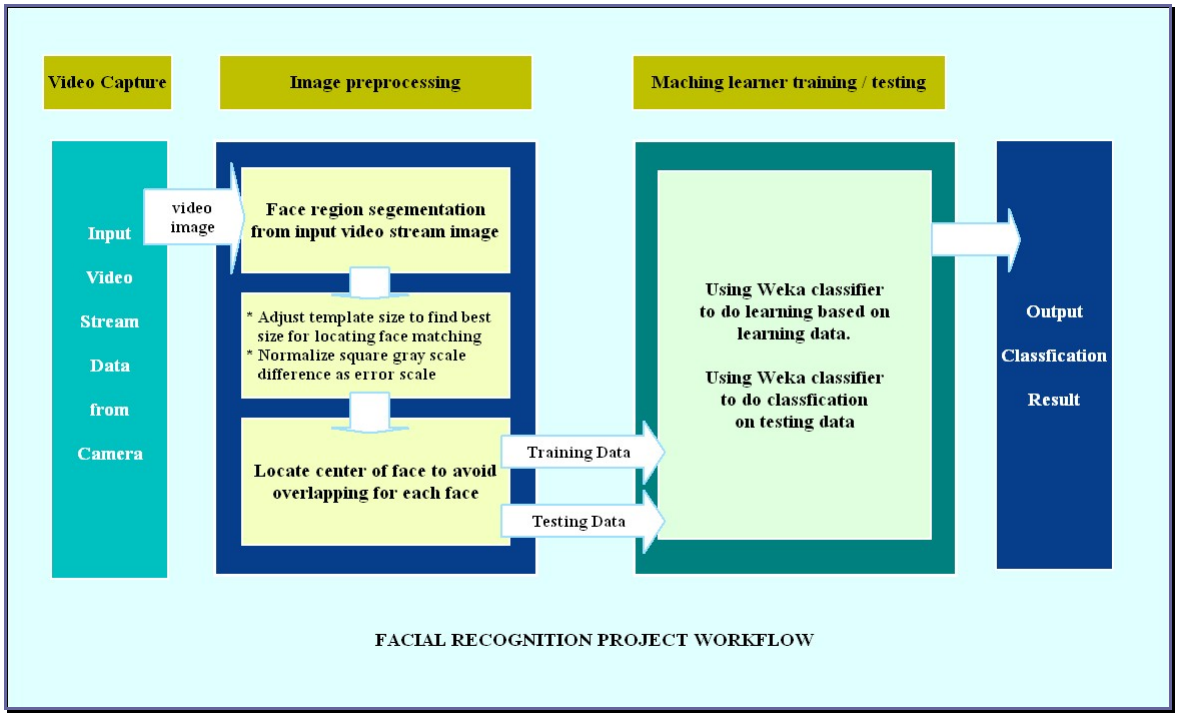
Although these methods have applied successfully to facial recognition in still images, facial recognition in sequential images (video streams) remains to be a difficult task due to several factors such as: low quality of image extracted from video, constant lighting condition change, pose difference (head turning), object motion (sideway moving, face image overlapping) and changes in sizing (close up and backing off). It is thus a demanding task to find a technique to effectively normalize all these affecting conditions in video derived images before and then send them into the existing framework of still image facial recognition.

Most recent work [Viola and Jones, 2003] have introduced new techniques combined with machine learning algorithms in order to quickly locate faces within images. They introduced a new midlevel representation for images, the integral image, which allowed significantly faster image processing at varying scales over previous techniques. In order to satisfy the requirements of real time processing, they utilized a customized variant of Adaboost, in which “cascaded” classifiers were used. Various decision stumps were applied in order of priority, such that regions that were highly unlikely to correspond to faces were immediately rejected before being further processed. Only those regions that initially seemed promising were processed further. As mentioned in their paper, utilizing 38 classifiers required over 80,000 operations to classify, whereas utilizing the classifiers resulted in only 270 operations, on average, per sub-window.

2. Implementation Overview

To address the facial recognition problem for video streams, we constructed our system based on two connected functional modules: The first module implemented as a C Dynamic Link library, utilizing Intel’s OpenCV library, processes image data from live video stream feeds, identifies and segments face regions within the image, matches Captured Face Template (CFT) with Stored Face Template (SFT), does automatic resizing for CFT to maximize the correlation, filters overlapping recognized templates, and at last sends normalized CFT as output to data trainer; The second module implemented in Java code, using the WEKA machine learning library, acts as the data

trainer, which accepts image data input from the first module, in order to train or test the data and return a classification rate showing who in the image has been identified. The main program calling these modules is written in Java, providing a graphical user interface with which the user can see the live calculation data output, and the identification result integrated with the video stream.



High-level Overview

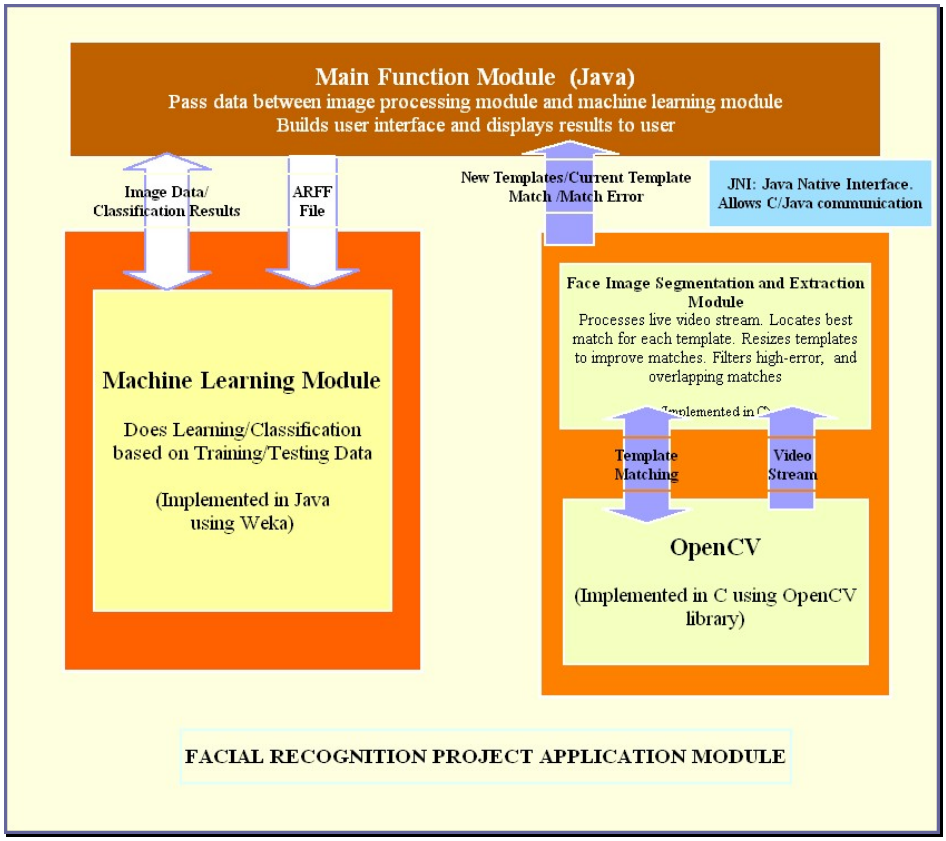
FaceRecLib.dll: Image Processing and Face Locating Module

- ***Face region segmentation function****:* before each training session for face recognition starts, the user will select (by drawing a square box) a face image from the video stream and store it as the Stored Face Template (SFT). In subsequent capturing phases, captured images are first turned into gray scale, for each input image from the video stream, the program select every sub-region within the input image in equal size to SFT, and subtract it with the SFT, if the error is below a certain threshold, it's then identified as a face region and also a match with the SFT.
- ***Captured Face Template (CFT) resizing function:*** in cases where the faces in the testing input image is closer or further to the camera compared to the training data, the face region in the testing images maybe substantially bigger or smaller than the SFT. Therefore the face finding module will iterate through different resized templates in order to maximize the match. Without doing thus, the face finder would likely lose the face being tracked once it surpassed a certain size.
- ***Template size search pruning using Markovian characteristics***: in order to effectively iterate through the possible template sizes without doing drastically increasing the computational cost, we utilized a first order Markov assumption with respect to the best scale of the template in sequential video frames images, that is, the face image from the next frame will not differ significantly in size from the image in the previous frame. Given that, even if the people in the image are fast approaching the camera or quickly backing off, the application can adjust the template size dynamically, while only trying a few other sizes slightly larger and smaller than the size used before.

- ***Pruning Overlapping Matches:***: the application can effectively separate and capture multiple faces within the same image. However, overlapping issues must be considered when two people begin to move closer to each other and eventually their faces overlap somewhat. Also, there is the possibility that there are more templates in the system being searched for than there are people in the current scene, allowing fro the possibility that a person is matched by more than one template---in these cases, the system must select which face to extract and consider a match. In our application, we implemented a judging mechanism, keeping both templates for both faces if overlap does not occur. If an overlap does occur, then the face that is matched the best is chosen. Implementation of this function makes our application robust against overlapping situations, either due to fewer people in the scene than tracked templates, or to overlap of individuals.
- ***Image normalization function***: because lighting conditions may change for different test data, we try to normalize our image according to different backgrounds and lighting conditions to be resilient to such lighting condition changes.

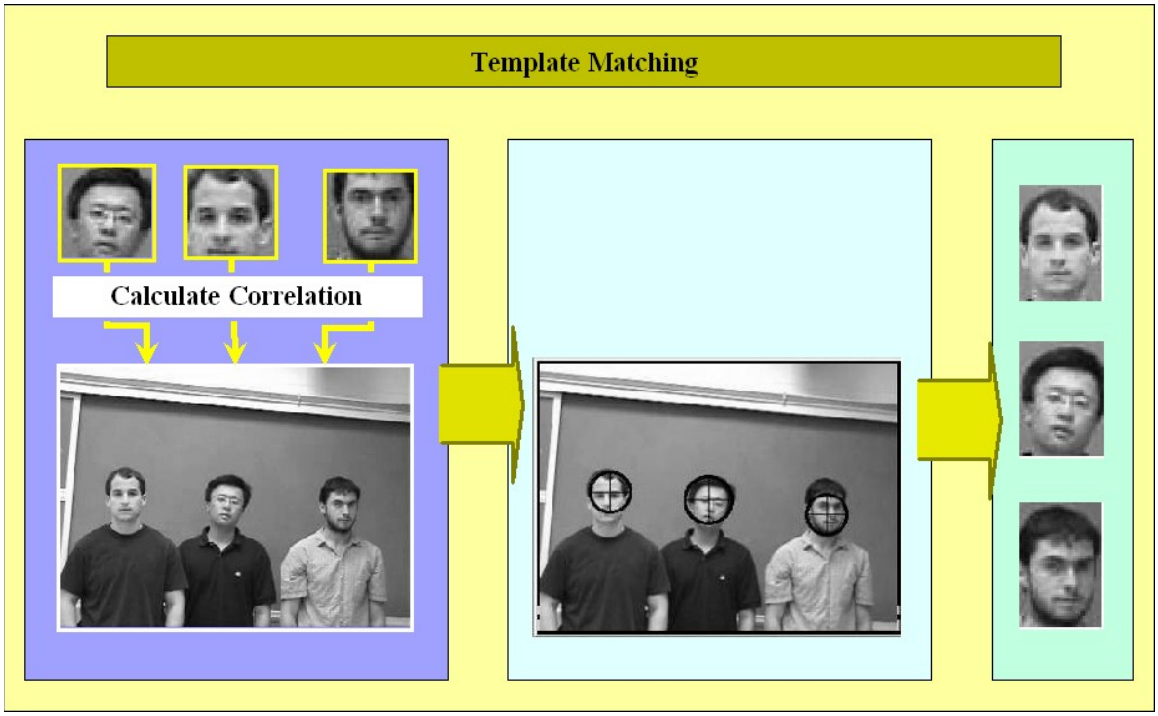
Java Program:

- **FaceRec:** The primary class responsible for handling events and connecting all of the other modules (both java and C-based) together.
- **Wekacaller:** The primary goal of our application was to generate a system capable of operating on real time video streams. As such, the wekacaller class was implemented in order to interface with the WEKA APIs in order to train and test various machine learning algorithms within the program, rather than have to resort to other external software.
- ***ARFFWriter***: In the case where one wants to perform extensive testing and analysis of different machine learning algorithms, on the fly classification using the WEKA APIs is not desired.



Functional Overview

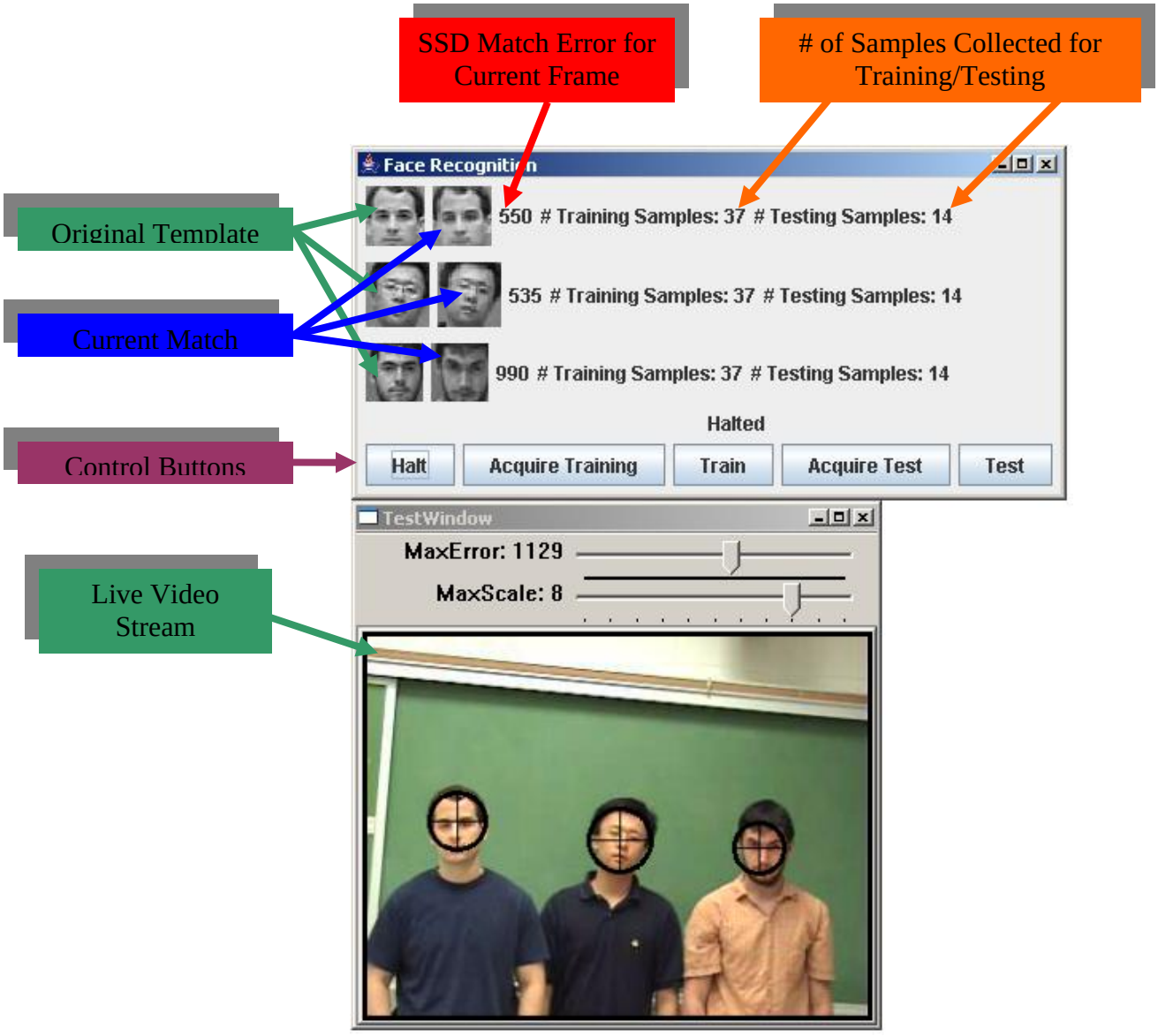
Template Matching:



Locating a desired template (small subimage) within another image is a common task in computer vision. Numerous methods exist. One of the simplest, known as Sum of Squares (SSD), involves calculating the difference between the template and the image. To do so, one places the template in the top left corner of the image, and then subtracts the template value for each pixel from the current image's value. Adding up all of these errors provides the value of the match for the top left most pixel. By shifting the template to the right, and performing the same calculation, one calculates the SSD for the next pixel. Continuing this process for all pixels (except for those near the borders where the entire template would not lie above actual image pixels) provides a two dimensional function over the area of the image. Finding the pixel with the lowest value (in the case of SSD) is equivocal to finding the part of the image that best matches the template. OpenCV provides two functions, cvMatchTemplate and cvMinMaxLoc, for calculating and locating the best match for a template within another image. This information is then used by the FaceRecLib for all of the more advanced processing and normalization.

In order for WEKA to correctly classify the image data representing the faces, it was necessary to normalize the data. For this reason the resizing of templates was introduced in order to first locate the best scale of the template which matches the current image. Then the portions of the current frame which best match the templates are extracted, resized to 32x32 pixel images, and then sent to the java module for processing by WEKA. Without the introduction of the resizing algorithm, if a person were to move twice as close to the camera, the template extracted for WEKA (assuming a match could even been found at all), would have only the center of their enlarged face, little resembling the original template. By introducing the adaptive scaling, the extracted faces are of much more consistent size than otherwise would've been the case.

User Interface:



The user first uses single left-click and drags to trace out rectangles on the video stream in order to extract the desired templates. Then the underlying face recognition algorithms will locate the templates in subsequent frames and mark them with crosshairs. The window which displays the live video stream has two scrollbars (MaxError and MaxScale). MaxError refers to the maximum difference (normalized per pixel) between the template and the best match which will still be considered a true match. MaxScale corresponds to the maximum allowed difference in scale between the original template and resized versions that will be allowed. This was added to prevent the face finding algorithm from constantly shrinking the template which results in increasingly better matches which correspond to very few pixels and a poor match. Lastly, since the OpenCV-based GUI tools used on the C side of the module do not support floating value trackbars, an integer bar of 0-10 is used for max scale, where 8 corresponds to 10-8=2, referring to +/- 20% scale difference allowed.

3. Results

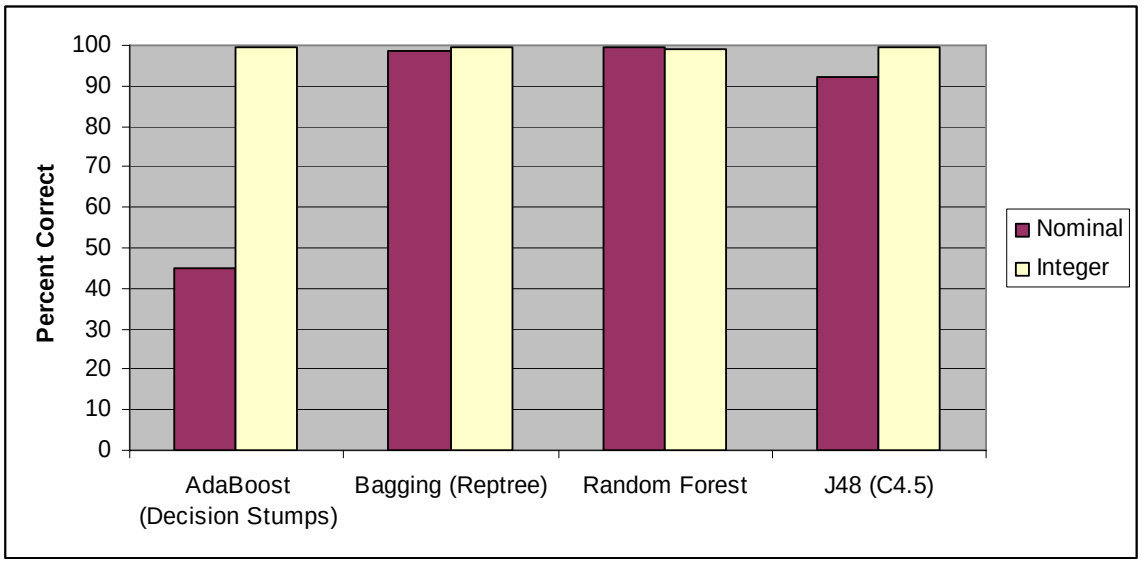
Machine Learning with WEKA:

Using the previously described software solution, hundreds of templates, representing 3 separate individuals were extracted from a video stream, labeled, and written to an ARFF file for a thorough analysis using various machine learning algorithms utilizing the WEKA Experimenter.

3.1 Nominal versus Integer Attributes

The first set of experiments was designed to compare representing pixel values as nominal versus integers attribute data types, as defined by the WEKA ARFF file format. Nominal attributes were defined as having a value from the set (0,1,…,254,255), whereas integer attributes were simply labeled as INTEGER. Treating pixel values as part of a set of nominal attributes corresponds to a scenario with absolutely no noise, in which pixel values of the same static point in space are expected to remain constant, given constant illumination. For the evaluation, four machine learning algorithms were selected: AdaBoost (utilizing Decision Stumps), Bagging (utilizing REPTrees), Random Forest, and C4.5. The results are presented below:

	Nominal Attribute		Integer Attribute	
Algorithm	Average	STD	Average	STD
AdaBoost (Decision Stumps)	45.15	1.36	99.54	0.3
Bagging (Reptree)	98.4	1.05	99.66	0.7
Random Forest	99.58	0.33	99.16	0.92
J48 (C4.5)	92.24	2.51	99.62	0.31

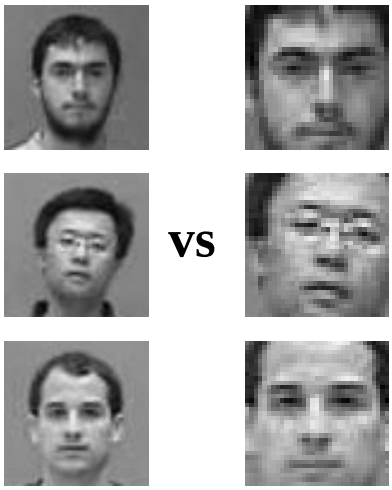


For three of the four algorithms evaluated, a modest but noticeable improvement is observed when using integer-valued pixel attributes. For Adaboost, however, a drastic difference existed. Adaboost performed horribly when using nominal attributes. Even after accounting for the standard deviation, the classifier produced by Adaboost actually performed worse than purely random guessing. When utilizing integer attributes,

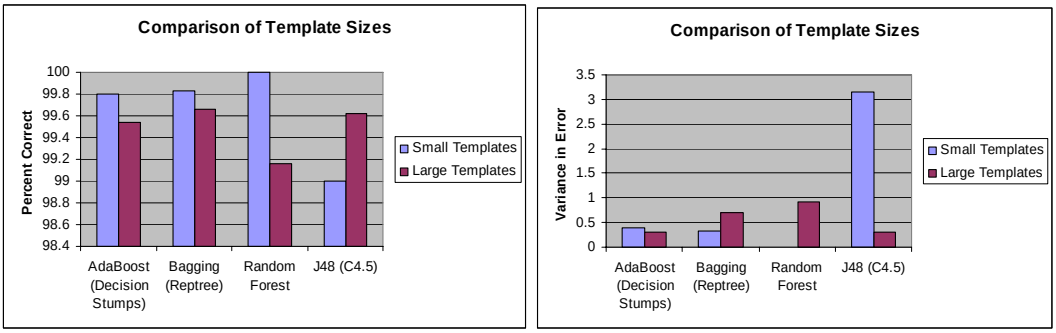
however, Adaboost worked as well as the other three algorithms. The explanation is most likely the general sensitivity of boosting algorithms, like Adaboost, to noise. By utilizing nominal attributes, a pixel value changing from 127 to 128 is as significant as a pixel changing from 127 to 255. In other words, the underlying algorithms have no intuition that pixel brightness of 128 is much closer to 127 than 255. In the presence of noise, in which pixel values fluctuate slightly, the boosting algorithms are therefore quite susceptible to the noise and perform quite poorly. For subsequent tests, pixel values were treated as INTEGER type values.

3.2 Area of Face Extracted

The second set of tests was designed to compare the performance of machine learning algorithms on data in which only the center of the face was present versus having the entire head and some of the background visible. For the purposes of normalization, square templates were always used. The human face being an oval, however, created the problem of choosing between extracting a larger template, in order to enclose the entire face, versus extracting a smaller template, and thereby losing some portions of the face (hair and jaw). Also note that all image data, before being processed by WEKA, is normalized to 32x32 pixel images. The same learning algorithms as before were applied to two datasets containing templates similar to the images displayed to the right. The results of the comparison are presented below:



	Small templates		Large Templates	
AdaBoost (Decision Stumps)	99.8	0.39	99.54	0.3
Bagging (Reptree)	99.83	0.32	99.66	0.7
Random Forest	100	0	99.16	0.92
J48 (C4.5)	99	3.14	99.62	0.31



Comparison of % Correct Classification/Standard Deviation

An initial view of the percent correct classification reveals that with three of the four machine learning classifiers, smaller templates were superior, with the exception of C4.5. The tables and graphs presented above, however, due not contain the complete results from the 10-fold by 10-run cross validation, which were excluded due to their length. As seen from second chart above, graphing the variance in error, C4.5 has a significantly

higher variance than all of the others. Upon closer examination of the full cross-validation results, it was observed that the lower scoring and increased variance of C4.5 was almost entirely the result of a single fold among the 10 selected for cross validation. The vast majority of classifications for each fold in each run were 100% correct classification. In every run, however, the ninth fold was classified correctly 90.91% of the time. The only other value contributing to C4.5's error rate and variance was 2nd run's classification of the sixth fold, with a success rate of 81.82%. In short, virtually all of the errors were the result of a single data fold.

With the error result in C4.5 explained, it is now clear that utilizing templates which focus on a smaller region of the face appear to perform better than larger templates. The result intuitively makes sense. In general, the significant features that tend to differentiate people are clustered near the center of the face, not on the boundaries. Further, heavy utilization of features on the periphery of the face would likely reduce results on a larger data set, because these portions of the face are more likely to change (hairstyles, motion of the jaw while talking, etc).

3.3 Varying Algorithm's Parameters

The final set of experiments was intended to examine the impact of varying some of the machine learning algorithm's parameters: namely the number of iterations and underlying classifier, in the case of Adaboost, and the number of trees, in the case of random forest.

The results for 10, 50, and 100 iterations for Adaboost (using Decision Stumps) are tabulated below:

Adaboost	Average	Std. Dev
10 iterations	99.57	0.82
50 iterations	99.66	0.75
100 iterations	99.66	0.75

Based on the results, it is clear that increasing the number of iterations beyond 10 improves the correct classification. There was no gain in using 100 over 50 iterations, suggesting that the best value is fifty or less.

Changing the number of forests had no effect on Random Forest; using either 10 or 100 trees resulted in 100% correct classification. Lastly, changing the underlying classifier for Adaboost to Random Forest gave the same 100% correct classification result.

4. Discussion

A face tracking library was implemented designed to track multiple faces in an image, adapting to changing sizes and positions within the image. The implemented algorithms worked relatively well, capable of tracking three individuals for a period of more than 2 minutes, including when the individuals rotated and turned their heads, altered facial expressions, swapped positions thus temporarily overlapping, and moving forward and backwards with respect to the camera, thus changing their perceived size. The module was able to provide normalized consistent data susceptible to meaningful processing by WEKA.

With data in hand, a series of machine learning algorithms were applied using different methods of data classification, template sizes, and alternative parameters. Through experimentation, it has been found that using integer pixel values of small sized templates enclosing the face with Random Forest clearly produces the best classifier for the case of differentiating between the three individuals.

5. Future work

Numerous other algorithms and methods were considered, but due to time constraints were not yet implemented. The work of Viola and Jones seemed most promising, but due to its need for a custom implementation of cascaded Adaboost, it could not be entirely replicated. Based on our results with Random Forests, however, we believe that an extension of Viola and Jones would likely benefit from using Random Forests as the underlying classifier for Adaboost, as opposed to Decision stumps. In order to maintain similar processing speeds, however, it is likely that the initial Forests used consist of relatively short trees.

In the realm of face extraction from a video stream where the same face is being tracked, rather than just located, additional AI algorithms would likely increase the tracking rate. Kalman filtering, traditionally used in tracking of objects overtime based on their previous path and sensory input from Gaussian-noise effected sensors, would likely improve the template tracking. More specifically, one could use the probability map for the face's location in the next frame as a weight factor with which to scale the values calculated from the SSD, thus tipping the scale between similar matches towards those near the face's previous known location. Further, the usage of extracting multiple templates, as well as extracting the previous matches to use as additional templates is predicted to likely improve tracking significantly.

Reference:

W. Zhao, R. Chellappa, A. Rosenfeld, P.J. Phillips, Face Recognition: A Literature Survey, ACM Computing Surveys, 2003, pp. 399-458

Turk, M. A. & Pentland, A. P. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience, 3(1)*, 71-86.

K. Sung and T. Poggio, "Example-based Learning for View-based Human Face Detection," A.I. Memo 1521, MIT A.I. Laboratory, 1994

T.K. Leung, M.C. Burl, and P. Perona, "Finding Faces in Cluttered Scene using Random Labeled Graph Matching," in Proceedings, International Conference on Computer Vision, pp. 637 – 644, 1995

P. Viola and M. Jones, "Robust Real-Time Face Detection." International Journal of Computer Vision, 2004.