Emoji Translator

Simon Julien (Cole), Kyle Kolanowski (Cole), Desmond Manthy (Grooms), Alex St. Clair (Cole)

December 2018

Abstract

In a world with many non-related software companies and data libraries, conversion algorithms are vital for cross platform communication. In this project, a basic facial recognition program was created capable of discerning between ten different emoji faces. This program was implemented using an approach based off Muller, Magaia, and Herbst's paper[2] on facial recognition. Through the use of training sets generated from Facebook and Apple's emoji libraries, facial averages of eigenfaces were created using the method of SVD factorization. A new unknown set of emoji faces were then compared against these averages and thus matched into their intended emoji expression (Smiley face, laughing face, etc.) by a least squares fit. Through multiple trials, our program was found to have a seventy percent success rate in sorting these unknown emojis, from Google's library, into their proper categories.

1 Attribution

Alex St. Clair was the lead MATLAB programmer, and participated in report editing; Simon Julien created the emoji library, offered MATLAB support, and contributed to writing the report; Desmond Manthy contributed heavily to the writing and editing of the final report; and Kyle Kolanowski also contributed to the writing and editing of the report and offered MATLAB support. All partners were involved in the general outline, research, and theoretical approach to our solution.

2 Introduction

With increasing technological capabilities, facial recognition has become a more and more prevalent aspect of day to day life. What was once science fiction has now been integrated into things as simple as signing into a phone, using social media platforms, and other modes of identification. This facial recognition software can be highly complex in nature, but the basic ideological foundation from which it is built is actually rather simple. These basic building blocks of facial recognition are the focus of this report. To explore these ideas, a simple facial recognition program was created, capable of identifying emoji faces. Emoji faces were chosen because they are simple, with easily defined features, making them more feasible to categorize. While this is a simpler version of a facial recognition program, it still utilizes the core mathematics of a full-fledged program.

The creation of this emoji recognition program was modeled after the method used by Muller, Magaia, and Herbst in their paper *Singular Value Decomposition, Eigenfaces, and 3D Reconstructions, SIAM Review*[2]. This paper models the use of singular value decomposition to create eigenvectors for the covariance matrix. When applied to the pixel columns of an image, these eigenvectors create an "eigenface" or an orthogonal facial feature representation for that image. All of these eigenfaces together form an orthonormal basis for the faces in the training set. By projecting facial (or emoji) images onto this basis (or a wisely chosen subset of the vectors that form this basis), the key characteristics can be much more efficiently represented. In our tests, these representations were over two orders of magnitude smaller than the full images. Not only does this mean that a database of known emojis can be stored much more efficiently, but also it means that emojis can be compared much more efficiently. Instead of comparing each emoji image directly, it is projected onto this basis to determine a representation in terms of the eigenfaces, and then those representations are compared directly.

The databases of what these new faces are compared to are created using a training set. A training set consists of various images of the same facial expression. The program created eigenfaces using this training set, and thus created a basis to project the new faces onto. A good training set is vital for the algorithm, and the paper outlines the importance of everything from

similar facial framing to similar lighting. Because our approach is using pixel data, it should be successful even with images that include accessories such as sunglasses or tears that overlap the actual face. The referenced paper does test accessory situations (such as subjects wearing glasses) and finds success in doing so. (In our application these distinct features proved useful for identification.) Additionally, the pictures must all have the same resolution and be in black and white to reduce external variables that add unnecessary complexity to the project. The significant advantage of using emojis is precisely this: that the images are nearly identical in framing and "lighting".

The following sections detail the creation of training sets from the Google, Facebook, and Apple emoji libraries, the creation of eigenfaces from standard images, the comparison of new eigenfaces to previously created libraries, and the success rate that this facial recognition program achieved. Moreover, it offers an explanation for the varying success rates of the algorithm over three different iterations.

3 Mathematical Formulation

The below process details the mathematical steps that were taken to form an eigenface basis and ultimately a database of different emojis from a training set. The process and notation are based on that which is outlined in the referenced paper[2]. More implementation detail can be seen in the Matlab code in the appendix.

1. Gather n grayscale emoji images of resolution $m = p * q$ for use in the training set
2. Form each emoji image into a column vector of size m, call it $\mathbf{f_j}$ for $j = 1...n$
3. Calculate the average face: $\mathbf{a} = \frac{1}{n} \sum_{j=1}^{n} \mathbf{f_j}$
4. For each face, calculate the deviation, $\mathbf{x_j}$, from the average face: $\mathbf{x_j} = \mathbf{f_j} - \mathbf{a}$
5. Form the full training set matrix: $X = \frac{1}{n}[\mathbf{x_1}\ \mathbf{x_2}\ ...\ \mathbf{x_n}]$
6. Calculate the SVD decomposition of X: $X = UDV^T$
7. Show the normalized singular values (along the diagonal of D) to get a sense for the decay rate and to determine the number of singular values

(and later eigenvalues) to use. Based on this, use v eigenfaces in the basis (the paper uses 150 for faces, though we only generate and use 20).

8. Use the first v eigenvectors from the covariance matrix $C = XX^T$. The columns of U are these eigenvectors. The vectors, $\mathbf{u_j}$, form a basis for the column space of X and are the eigenfaces.

9. Thus, we create the orthonormal basis we will use: $U_v = [\mathbf{u_1}\ \mathbf{u_2}\ ...\ \mathbf{u_v}]$

10. Now, we can compare faces (emoji) directly using their eigenface representations, $\mathbf{y_j}$: $U_v\mathbf{y_j} = \mathbf{f_j} - \mathbf{a} \Rightarrow \mathbf{y_j} = U_v^T(\mathbf{f_j} - \mathbf{a})$ given that U_v is orthonormal.

11. To compare faces, simply find the error between each representation, and match the emoji with the least error. Thus, for all representations in the database, y_j, minimize $e = |y_{new} - y_j|$ over $j = 1, 2, ...n$.

4 Examples and Numerical Results

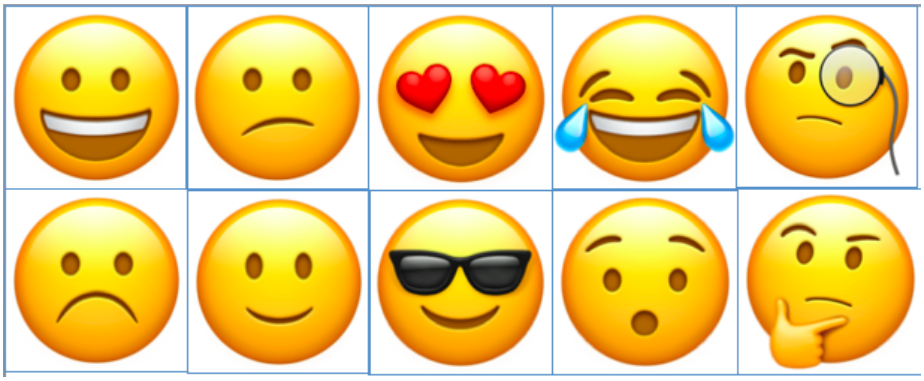

Figure 1: Relevant emojis: Above is a visual of our 10 chosen emoji expressions illustrated through the Apple software design.

Emoji images were first obtained for the following emotion types: big smile, confused, heart eyes, laughing, monocle, frowning, regular smile, sunglasses, surprised, and thinking respectively to what is depicted in Figure 1. For the analysis the best fitting result was determined by assigning a confidence value to each iteration of comparison. The highest resulting confidence value outputted from our least squares fit was then determined to be the program's final answer to the matching emoji.

These emojis initially started at different resolutions. They were then modified into consistent resolutions of 120 x 120 so their corresponding RGB matrices would be compatible for analysis.

Apple and Facebook emoji libraries are now used to create a spacial basis for the first step of our algorithm. This means that the combination of these company's software types and different emotions have been "overlayed" 20 times to combine into our critical basis shown in Figure 2.

Figure 2: Eigenfaces: the above images show the eigenfaces created from Apple and Facebook libraries to span our emoji emotion space as a basis.

Each image is an orthonormal vector and together they span the column space of X, which is the full training set. The eigenfaces are displayed from left to right and top to bottom in descending order of singular value magnitude. The singular values are the values along the diagonal of D, and a normalized plot of the decay of the singular values is shown in Figure 3.

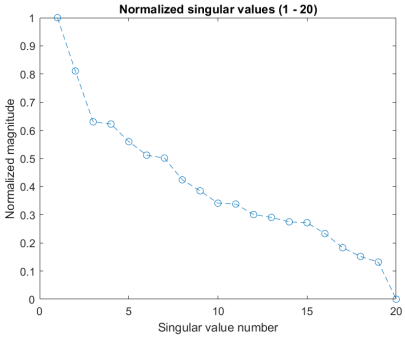

Figure 3: Decay of singular values as found in the D matrix

The singular values decay somewhat exponentially. It can be seen that the final singular value is effectively zero. It corresponds to the final eigenface (shown previously), which contains no useful information other than perhaps a facial outline. The eigenfaces as represented in U are floating point values with various ranges. So, in order to display them, we normalized each image to span the 0-255 value range of an 8-bit unsigned integer so that they can clearly be shown in a greyscale image. We took the first (and only) 20 results of this and set them to be the span of our future space.

Given this basis of eigenfaces, we were able to create the U_v matrix by projecting each face onto the eigenface space. Given this, we calculated the representation for each of the 20 emojis used in the training set: for each of the 10 emojis, the Apple and Facebook versions were used in the training set.

Given this database of emoji representations, we were ready to begin optimizing our emoji matching algorithm to successfully translate an emoji from an unknown library (in our case, the Google one) into the intended corresponding emotion. We began with an algorithm that we reference as our "naive" results. This simply found the difference between the unknown emoji image representation and our basis library and assigned that a confidence value. The optimal confidence value was then picked as the algorithm's best prediction of a correct match. Counterintuitively, the confidence interval is simply the error value, so a lower confidence value represents a better match. The first confirmation of the algorithm's success was demonstrated by a significant success-rate of 7/10 (70%) in correctly identifying the emotions in this "naive" case.

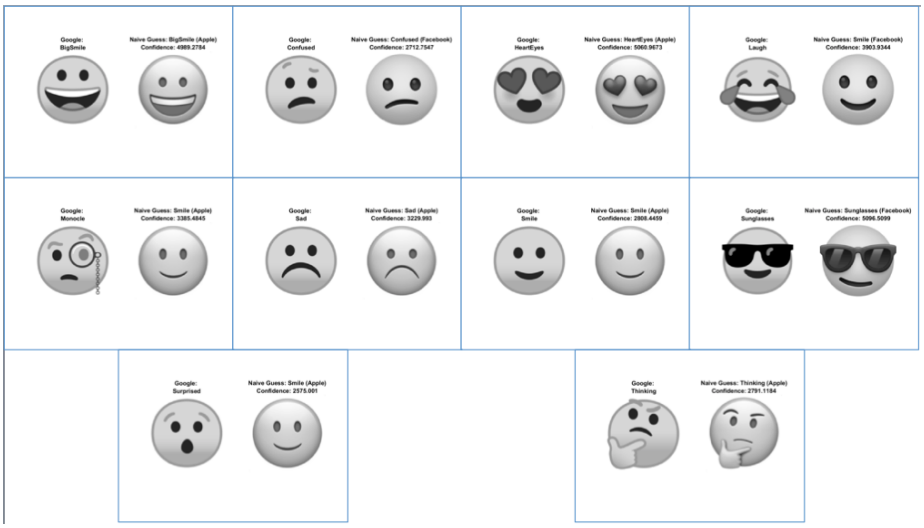

Figure 4: Naive Method of Matching: By this method, the facial recognition program was able to match seven out of ten of the emojis with their proper category. The failed cases were the laughing emoji, the monocle emoji, and the surprised emoji. Note that two of the three failed cases have accessories that extend beyond that original shape of the emoji face.

In an attempt to improve the success rate of our algorithm, we performed the same method of math optimization, while only including 19 of the 20 eigenfaces for our basis of comparison. This was intended to allow our basis to be less restrictive and potentially find a stronger match with low confidence value results. As seen above in Figure 2, the 20th eigenface contains no useful information. Although the exact confidence values did vary slightly (some in the wanted direction), we ended up with the same exact matches and failing cases, as shown in Figure 6.

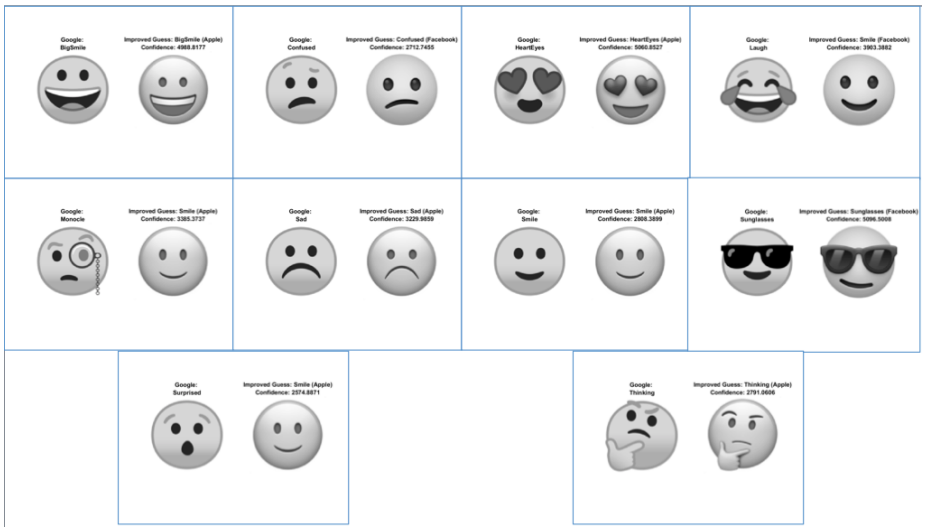

Figure 5: Improved Matching Results: Again we have found the program was able to match seven out of ten of the emojis with their proper category. The failed cases were still the laughing emoji, the monocle emoji, and the surprised emoji. The assigned confidence values are also shown, which are slightly different than the "naive" matching method's confidence values.

Again with the hopes of improving the algorithm's accuracy, the same method of mathematical optimization was attempted with the change that each eigenface was "weighted" or multiplied by singular values. The reasoning behind this was that multiplying by these singular values would properly take into account the importance of each eigenface and likewise make the corresponding confidence value more distinct. In theory, this would make it easier to extract the relevant facial features given the importance of each eigenface. However, in practice this resulted in only five of the ten faces being properly matched, dropping the algorithm to a fifty percent success rate. This made it evident, that "weighting" the eigenfaces did not in fact makes them more easily identified. In fact it did just the opposite for the algorithm. Ultimately this method resulted in our algorithm's lowest performing scenario.

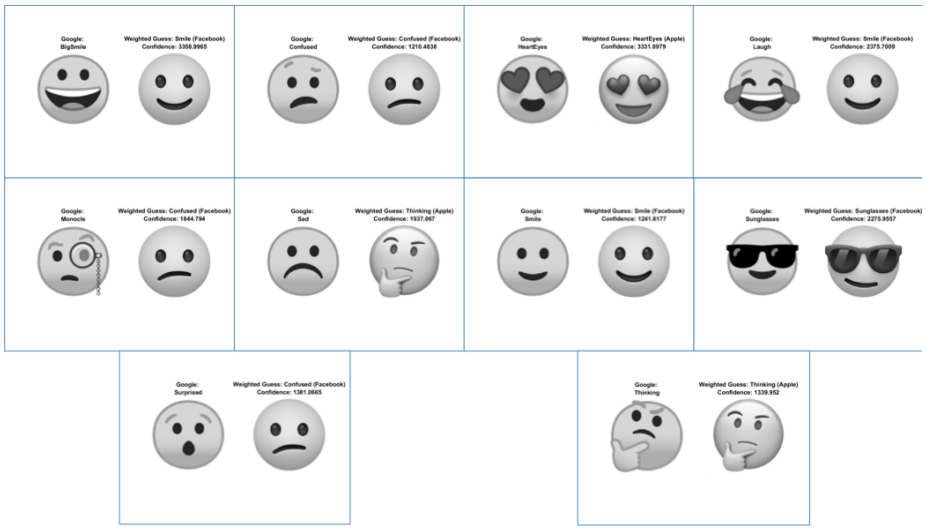

Figure 6: Weighted: By weighting each eigenface by singular value, the facial recognition program was able to match five out of ten of the emojis with their proper category. Using this method yielded poorer result than either the naive or improved methods.

Through these three variations on the algorithm, it was found that with both a "naive" and "Improved" comparison, the program could reach a seventy percent match rate. More, the "weighted" trial revealed that the error comparison solution provided enough of a difference for the algorithm to recognize faces without being magnified or weighted. However, no trial was able to exceed a seventy percent match rate which brought up the question of what limiting factor was preventing a one hundred percent match rate. Based on these results, it seems that certain emoji faces were recognized across all three trials, and likewise certain emojis were missed across all three trials. On speculation, it would appear that there are certain traits/characteristics of emoji faces that are more influential or easily identified by the algorithm in matching them. For instance, emojis with significant features apart from the eyes and mouth such as tear drops, hearts, and sunglasses were never misidentified. However, emojis without notable features apart from the eyes and mouth were often mixed up. If this were to be continued further, it may be worth considering adjusting the algorithm to "focus" more directly on the eyes and mouth of the emoji face rather than the entire generic circular shape. This in theory, would allow for emoji faces without notable features to be sorted more accurately.

Finally, take for example the difference between monocle emojis as shown in Figure 7. Aside from the fact that each contains a monocle, they share little in common. We tried to match the Google version to either of the two others, but the Google mouth is much to thick to be matched with Apple, and likewise the Google eyebrows are too curved and thin to be matched with Facebook. It does, in fact, make sense for the Google version to match with Facebook's confused emoji, as the weighted algorithm did.

(a) Apple (b) Facebook (c) Google

Figure 7: Different monocle emoji

5 Discussion and Conclusions

Based on the results, the "naive" and "improved" methods of matching the unknown library's faces to those of the training set were quite successful. Each of these matching parameters successfully predicted seventy percent of the emotion types from the Google emoji library. An attempt at improving these results by incorporating a weighted matching method was unsuccessful, yielding only a fifty percent success rate.

Some faces were consistently chosen correctly among the different matching methods. The following emojis were predicted correctly for all three methods: thinking, glasses, smiling, heart eyes, and confused. This outcome is most likely a result of these faces' distinct appearances. It is easy to understand that the program should be less likely to get confused when a significant portion of an emoji's face is blocked out by a shape that is only common to the training set for the emotion and the unidentified emoji alone. This is especially the case for the thinking, glasses, and heart eyes emojis. Each of these emotion types has a unique shape that occupies a substantial amount of the image (a hand stroking a nonexistent beard, dark shades, and hearts covering what would normally be eyes). The consistency in predicting the smiling and confused faces are less clear. These emojis may have been consistently predicted correctly due to decreased

variance between the different emoji libraries.

Other emojis in the Google library were consistently predicted incorrectly. These included the following faces: monocle, surprised, and laughter. Interestingly, each of these emojis were predicted to be the smiley face. The reason that the software preferred the smiley face for these emojis is unclear. The program may have emphasized other features in its assessment which humans might overlook. For example, the monocle and surprised emojis from Google both demonstrate shading on the lower portions of their face which closely resembles the shading on the Apple smiley face.

In future work, it may be possible to better refine the matching methods to achieve greater effectiveness of the software since only a simple error comparison was used. However, the best way to improve the results would be to use a larger training set. Only 20 emojis (ten expressions from two companies) were used in the training set. The paper[2] that outlined the method we used required 600 images (200 individuals with three photos each) for its training set. As such, we only used 20 eigenfaces for our basis, where the paper had 150. A notable difference, naturally, is that the actual faces in the paper were higher resolution, and faces have more subtle variation than emojis.

For only using 20 eigenfaces in its basis, our algorithm performed remarkably well. Many of the mismatches are likely the result of the emojis being too different from each other for the algorithm to match them. This is an error in the training set more so than anything else. If a more diverse group of emojis were used, the algorithm could more easily find a correct match. However, this would result in an increase in storage size for the facial representations. With only 20 eigenfaces, we only need 20 variables with double precision to represent each face (floating precision would almost certainly suffice as well). So, considering the very small size of each representation, the algorithm is very effective.

References

[1] Meyer, C (2000), *Matrix Analysis and Applications*, Society for Industrial and Applied Mathematics,

[2] Muller, N., Magaia, L., Herbst, B. M. (2004), *Singular Value Decomposition, Eigenfaces, and 3D Reconstructions*, SIAM Review Vol. 46, No. 3, pp 518-545.

[3] "Google." Google Emoji Library, Aug. 2018, emojipedia.org/google/.

[4] "Apple." Apple Emoji Library, Oct. 2018, emojipedia.org/google/.

[5] "List of Emoticons for Facebook." Symbols Emoticons, www.symbols-n-emoticons.com/p/facebook-emoticons-list.html.

Appendix

MATLAB Code

5.0.1 Main

```
%% Perform the tedious work of loading images and calculating the basis
[emoji,names] = read_images();
[a,X,U,D,V,U_v] = create_basis(emoji,10,2,14400);

%% Show the singular values
singular_values = zeros(20,1);
for itr = 1:20
    singular_values(itr) = D(itr,itr);
end

% normalize singular values
singular_values = singular_values / max(singular_values);

figure(1)
plot(singular_values, 'o--')
xlabel("Singular value number");
ylabel("Normalized magnitude");
title("Normalized singular values (1 - 20)");
```

```
%% Show the eigenfaces
eigenface = uint8(zeros(120,120));
for itr = 1:20
    figure(itr+100)
    eigenface = reshape(uint8((U(:,itr)-min(U(:,itr)))*(255/max(U(:,itr)))),[120,120]);
    imshow(eigenface);
    imwrite(eigenface,['eigenfaces/eigenface',num2str(itr),'.png']);
end

%% Create library of eigenface representations (using apple, faceboook)
library = cell(10,2);
for itr1 = 1:10
    for itr2 = 1:2
        % y = U_v' * (f - a)
        library{itr1,itr2} = U_v'*(double(emoji{itr1,itr2})-a);
    end
end

%% Match the Google emoji to the ones used to form the basis
for itr1 = 1:10
    find_match_naive(emoji{itr1,3}, names{itr1}, U_v, a, library, emoji, names);
end

%% Ignore the final eigenface in an attempt to improve results
for itr1 = 1:10
    find_match_improved(emoji{itr1,3}, names{itr1}, U_v, a, library, emoji, names);
end

%% Use a weighted match method in an attempt to improve results
for itr1 = 1:10
    find_match_weighted(emoji{itr1,3}, names{itr1}, U_v, a, singular_values, library, emoji, names);
end
```

5.0.2 Naive Matching

```
function confidence = find_match_naive(unknown, unknown_name, U_v, a, library, images, names)
%find_match uses least squares to determine a match for the input face

best_match = 10^10; % arbitrarily large (bad) guess
image_representation = U_v'*(double(unknown)-a);
guess_row = 0;
guess_col = 0;

[rows,cols] = size(library);

for itr1 = 1:rows
    for itr2 = 1:cols
        current_match = norm(image_representation - library{itr1,itr2});
        if (current_match < best_match)
            guess_row = itr1;
            guess_col = itr2;
            best_match = current_match;
        end
    end
end

confidence = best_match;

figure()
subplot(1,2,1)
imshow(reshape(unknown,[120,120]));
title(["Google: ",unknown_name]);
subplot(1,2,2)
imshow(reshape(images{guess_row,guess_col},[120,120]));
if (guess_col == 1)
    title({['Naive Guess: ',names{guess_row},' (Apple)'],['Confidence: ',num2str(confidence)]});
else
    title({['Naive Guess: ',names{guess_row},' (Facebook)'],['Confidence: ',num2str(confidence)]});
end
saveas(gcf,['naive_matches/naive_',unknown_name,'.png']);

end
```

5.0.3 Improved Matching

```
function confidence = find_match_improved(unknown, unknown_name, U_v, a, library, images, names)
%find_match uses least squares to determine a match for the input face

best_match = 10^10; % arbitrarily large (bad) guess
image_representation = U_v'*(double(unknown)-a);
guess_row = 0;
guess_col = 0;

[rows,cols] = size(library);

for itr1 = 1:rows
    for itr2 = 1:cols
        current_match = norm(image_representation(1:19) - library{itr1,itr2}(1:19));
        if (current_match < best_match)
            guess_row = itr1;
            guess_col = itr2;
            best_match = current_match;
        end
    end
end

confidence = best_match;

figure()
title("Improved match algorithm");
subplot(1,2,1)
imshow(reshape(unknown,[120,120]));
title(["Google: ",unknown_name]);
subplot(1,2,2)
imshow(reshape(images{guess_row,guess_col},[120,120]));
if (guess_col == 1)
    title({['Improved Guess: ',names{guess_row},' (Apple)'],['Confidence: ',num2str(confidence)]});
else
    title({['Improved Guess: ',names{guess_row},' (Facebook)'],['Confidence: ',num2str(confidence)]}
end
saveas(gcf,['improved_matches/improved_',unknown_name,'.png']);

end
```

5.0.4 Weighted Matching

```
function confidence = find_match_weighted(unknown, unknown_name, U_v, a, singular_values, library, i
%find_match uses least squares to determine a match for the input face

best_match = 10^10; % arbitrarily large (bad) guess
image_representation = U_v'*(double(unknown)-a);
guess_row = 0;
guess_col = 0;

[rows,cols] = size(library);

for itr1 = 1:rows
    for itr2 = 1:cols
        current_match = norm(singular_values.*(image_representation - library{itr1,itr2}));
        if (current_match < best_match)
            guess_row = itr1;
            guess_col = itr2;
            best_match = current_match;
        end
    end
end

confidence = best_match;

figure()
title("Single-value-weighted match algorithm");
subplot(1,2,1)
imshow(reshape(unknown,[120,120]));
title(["Google: ",unknown_name]);
subplot(1,2,2)
imshow(reshape(images{guess_row,guess_col},[120,120]));
if (guess_col == 1)
    title({['Weighted Guess: ',names{guess_row},' (Apple)'],['Confidence: ',num2str(confidence)]});
else
    title({['Weighted Guess: ',names{guess_row},' (Facebook)'],['Confidence: ',num2str(confidence)]}
end
saveas(gcf,['weighted_matches/weighted_',unknown_name,'.png']);

end
```

5.0.5 Reading Images

```
function [emoji,names] = read_images()
%read_images returns a cell matrix of emoji vectors after reading the files

emoji = cell(10,3);
names = cell(10,1);

% BigSmile
emoji(1,1) = {reshape(rgb2gray(imread('emojis/BigSmile/aBigSmile.png')),[14400,1])};
emoji(1,2) = {reshape(rgb2gray(imread('emojis/BigSmile/fBigSmile.png')),[14400,1])};
emoji(1,3) = {reshape(rgb2gray(imread('emojis/BigSmile/gBigSmile.png')),[14400,1])};
names(1) = {'BigSmile'};

% Confused
emoji(2,1) = {reshape(rgb2gray(imread('emojis/Confused/aConfused.png')),[14400,1])};
emoji(2,2) = {reshape(rgb2gray(imread('emojis/Confused/fConfused.png')),[14400,1])};
emoji(2,3) = {reshape(rgb2gray(imread('emojis/Confused/gConfused.png')),[14400,1])};
names(2) = {'Confused'};

% HeartEyes
emoji(3,1) = {reshape(rgb2gray(imread('emojis/HeartEyes/aHeartEyes.png')),[14400,1])};
emoji(3,2) = {reshape(rgb2gray(imread('emojis/HeartEyes/fHeartEyes.png')),[14400,1])};
emoji(3,3) = {reshape(rgb2gray(imread('emojis/HeartEyes/gHeartEyes.png')),[14400,1])};
names(3) = {'HeartEyes'};

% Laugh
emoji(4,1) = {reshape(rgb2gray(imread('emojis/Laugh/aLaugh.png')),[14400,1])};
emoji(4,2) = {reshape(rgb2gray(imread('emojis/Laugh/fLaugh.png')),[14400,1])};
emoji(4,3) = {reshape(rgb2gray(imread('emojis/Laugh/gLaugh.png')),[14400,1])};
names(4) = {'Laugh'};

% Monocle
emoji(5,1) = {reshape(rgb2gray(imread('emojis/Monocle/aMonocle.png')),[14400,1])};
emoji(5,2) = {reshape(rgb2gray(imread('emojis/Monocle/fMonocle.png')),[14400,1])};
emoji(5,3) = {reshape(rgb2gray(imread('emojis/Monocle/gMonocle.png')),[14400,1])};
names(5) = {'Monocle'};

% Sad
emoji(6,1) = {reshape(rgb2gray(imread('emojis/Sad/aSad.png')),[14400,1])};
emoji(6,2) = {reshape(rgb2gray(imread('emojis/Sad/fSad.png')),[14400,1])};
emoji(6,3) = {reshape(rgb2gray(imread('emojis/Sad/gSad.png')),[14400,1])};
names(6) = {'Sad'};
```

```
% Smile
emoji(7,1) = {reshape(rgb2gray(imread('emojis/Smile/aSmile.png')),[14400,1])};
emoji(7,2) = {reshape(rgb2gray(imread('emojis/Smile/fSmile.png')),[14400,1])};
emoji(7,3) = {reshape(rgb2gray(imread('emojis/Smile/gSmile.png')),[14400,1])};
names(7) = {'Smile'};

% Sunglasses
emoji(8,1) = {reshape(rgb2gray(imread('emojis/Sunglasses/aSunglasses.png')),[14400,1])};
emoji(8,2) = {reshape(rgb2gray(imread('emojis/Sunglasses/fSunglasses.png')),[14400,1])};
emoji(8,3) = {reshape(rgb2gray(imread('emojis/Sunglasses/gSunglasses.png')),[14400,1])};
names(8) = {'Sunglasses'};

% Surprised
emoji(9,1) = {reshape(rgb2gray(imread('emojis/Surprised/aSurprised.png')),[14400,1])};
emoji(9,2) = {reshape(rgb2gray(imread('emojis/Surprised/fSurprised.png')),[14400,1])};
emoji(9,3) = {reshape(rgb2gray(imread('emojis/Surprised/gSurprised.png')),[14400,1])};
names(9) = {'Surprised'};

% Thinking
emoji(10,1) = {reshape(rgb2gray(imread('emojis/Thinking/aThinking.png')),[14400,1])};
emoji(10,2) = {reshape(rgb2gray(imread('emojis/Thinking/fThinking.png')),[14400,1])};
emoji(10,3) = {reshape(rgb2gray(imread('emojis/Thinking/gThinking.png')),[14400,1])};
names(10) = {'Thinking'};

end
```

5.0.6 Forming Basis

```
function [a,X,U,D,V,U_v] = create_basis(images, num_subjects, num_sets, sample_length)
% create_basis given a training set of images, create a basis for the
% eigenface space
%
% Inputs:
%   images - cell array of uint8 image vectors
%   num_subjects - number of unique subjects (num rows in images)
%   num_sets - number of examples of each subject (num cols in images)
%   sample_length - length of each image vector
%
% Outputs:
%   a - average face as determined by taking a simple average
%   X - matrix of deviations for each face
%   U - U matrix from SVD of X
%   D - D matrix from SVD of X
%   V - V matrix from SVD of X
%   U_v - basis matrix using first 20 eigenfaces (vectors) from U

% initialize empty average vector
a = zeros(sample_length,1);

% sum all of the emoji
for itr1 = 1:num_subjects
    for itr2 = 1:num_sets
        a = a + double(images{itr1,itr2}); % a has double precision
    end
end

% calculate the average
a = a / (num_subjects * num_sets);

% empty deviations matrix
X = zeros(sample_length,num_subjects*num_sets);

% calculate the deviation vectors
for itr1 = 1:num_subjects
    for itr2 = 1:num_sets
        X(:,itr1 + (itr2-1)*num_subjects) = double(images{itr1,itr2}) - a;
    end
end
```

```
% X = (1/n) * X
X = X / (num_subjects * num_sets);

% SVD factorization
[U,D,V] = svd(X);

% now, use only the first 20 eigenfaces to form the basis
U_v = U(:,1:20);

end
```