Recognizing Linear Algebra's Potential: The Eigenface Method of Facial Recognition

Lauren Cooke, Alex Dyer, Ashley Herrera, Annika Huprikar

Math 22a: Vector Calculus and Linear Algebra I

December 14, 2020

0 Cover Letter

The writing process has been extremely rewarding, and we hope that shines through in the final result. Our research process began with the list of suggested topics within the Project Guidelines on Canvas. The topic of Facial Recognition is broad and spans a variety of topics and subjects. From neural networks and artificial intelligence to optics and surveillance, the myriad of components within this space provided plenty of room for exploration. Out of all of the different methods for facial recognition, the one that stood out most for us was the aptly named *Eigenface Approach*. The cornerstone of this method is the work of Matthew Turk and Alex Pentland. In addition to this original source, we have been lucky that many other people have shared our interest in the subject and published more information on the topic. From textbooks to published lectures to YouTube videos, we have drawn our knowledge on the topic from a diverse range of sources.

Transitioning from the initial discovery phase into writing the paper, we divided the workload by different topics. Lauren proved the Spectral Theorem and implemented the code that the examples utilize. Alex described the motivations, broad overview of treating images as matrices and vectors, and the pros/cons. Ashley focused on the applications of eigenfaces as well as the Principal Component Analysis section. Annika provided the technical side overview and description of spanning sets. Over time, we began to work progressively more out of our respective roles. Especially after we received our draft feedback, each one of us has been able to provide our distinct perspectives to address the deficits of the former submission. We have all revised each other's work and can easily call this paper in its entirely the product of all of our efforts. The end result is a paper where every member has contributed in a substantial way to every section, explanation, and example.

Concerning moving from the draft to the final copy, the feedback we received has been instrumental to polishing and revising. The bevy of feedback spanned the entirety of the work and helped us make improvements in every corner of the paper. In reviewing the feedback, we identified several key areas of revision: transitions, confusion surrounding the covariance matrix, inconsistent formatting, thrown-in definitions, non-integrated figures, inconsistent notation, and a lack of justification for why we include the spectral theorem. The transitions involved a lack of subsections and section titles that were not the most descriptive. In terms of formatting, adding subsections has been the most instrumental change in beautifying the final copy. By going through the feedback, we added appropriate subsections wherever the paper's structure began to become too confusing. With the covariance matrix, we made sure to define the key terms of covariance and the covariance matrix. We added the entirety of section 3.1 in order to define and explain these terms within the context of facial recognition. In order to address inconsistent formatting, we made sure to be consistent with how we referred to different variables. Since we drew our information from a diverse body of literature, making our notations consistent has been especially important. Regarding thrown-in definitions, the subsection titles helped with this regard. Also, we made sure to add prose explaining their inclusion and implications. Integrating the figures was particularly interesting, because we were able to play with warping text and with how latex handles figures. We made sure to include the figures at points in the texts that they were most needed in addition to cutting a graph that was superfluous. We hope that the revised figures will be effective in breaking up large blocks up texts and helping the reader best understand more abstract topics. Lastly, we hope to have driven home the idea that the Spectral Theorem undergirds the use of PCA. In other words, the theorem provides a justification for how we know that PCA will work when applied to facial image vectors.

As always, we have been committed to the Harvard College Honor Code in all of the work that has culminated in this project.

We hope you enjoy reading about the linear algebra basis for facial recognition. If any aspect of the paper is particularly intriguing, please feel free to refer to the linked bibliography for further reading. Also, our code can be found at this Jupyter Notebook. Following the link will prompt you to download a zip file that contains the code. The code goes through many of the same steps as this written project and finds unique ways to implement the results that we have found. If you have the chance, we definitely recommend that you at least take a peek!

1 Eigenfaces Overview

1.1 Motivations

Take a look at the image on the right of the page. If you see a colon and parenthesis smiling back at you, you are not alone. Over the course of a lifetime, the human mind intuitively recognizes millions of faces in the objects and people around it. Although scientists do not completely understand the mental mechanisms that allow us to identify faces so well, researchers still question whether we can construct a computational system that can recognize faces on par with the human mind. We often use the terms recognition and verification interchangeably in our day to day lives, but the difference in definition of these two terms will help us understand our goals. While recognition simply checks whether or not an image contains a face, verification goes further by identifying to whom the face belongs. In this construction process, systems must learn to recognize faces before they can begin verifying them. As a result, we will be focusing on recognition.

:)

So, what exactly defines a face, and how can we teach a computer to recognize them? A heuristic approach may attempt to recognize faces much like a human would, checking to see if an image has two eyes, a nose in between, and a mouth right below. Another approach would be to look at the image as a whole, comparing the sequence of colors between an unknown image and pictures of faces to determine if this new image is similar enough. Early exploration of this process comes from the work of Matthew Turk and Alex Pentland, who used linear algebra to assign eigenvectors to these known and unknown faces, which they aptly named **eigenfaces**. As we explore this method for facial analysis, we hope not only to examine how machines may learn to detect faces, but also to explore some of our subconscious mental processes.

1.2 The Process

To recognize faces using the **eigenface approach**, we collect a set S of face images that we will use as the 'standard' for what a face looks like. Given that these images are our model faces, we then want to represent what they have in common to get a better understanding of what a face is. To do this, we create our average face a by averaging the pixel information from each image to a single picture. We then generate a **face space** to quantify just how different these face images are using a smaller collection C of these face images, particularly the images that contribute the most to possible image variations. We can then reconstruct each face image in set S by taking the linear combination of the face images in C. This process of reconstructing faces within the face space is identical to the process of identifying a vector within an eigenspace. For example, if we form a face space using only 4 face vectors, then we can represent any face within this space using a combination of these face vectors. If the face image has rounder cheeks, then we assign more influence (in the form of a higher weight) to the face vector that corresponds with this feature. When looking at a new image, we must compare it with the average face as well as the weighted combinations of the face images in the set S to determine its distribution of image variation. This analysis will determine whether the image is a known face, an unknown face, or not a face altogether (Turk and Pentland, 73).

We can boil this process down into a series of high-level steps:
Initializing the facial recognition system

1. Compile a set S of p face images, known as the **training set**. Each image corresponds to a matrix of size $n \times m$. Each image is then transformed into a column vector in $(n \times m)$ space.

2. Transform the data represented by these images into a covariance matrix that is $nm \times nm$
   - **Covariance** is a measure of the relationship between two variables (more information in 3.1). The **covariance matrix** is a square, symmetric matrix that provides the covariance of two elements of a vector.

3. Calculate the p eigenvalues and eigenvectors of this matrix, where the eigenvectors are referred to as **eigenfaces**. Note: this step will require working in a smaller-dimensional $p \times p$ subspace (as opposed to $nm \times nm$)

4. Using Principal Component Analysis, create a basis for the face space by choosing a set of eigenvectors whose associated eigenvalues are the largest, representing the most variation in the training set of images. This basis will consist of p' vectors, where $p' \leq p$.

- **Principal Component Analysis** is a method that suppresses repetitive information by reducing the number of dimensions that are necessary to represent a given data set, thus calling the necessary components the principal elements of a data set.

5. Project the face images onto the resulting face space, as by calculating the distribution in p-dimensional weight space for each known individual face image

Recognizing outside face images

1. Project the outside image onto the face space by projecting it onto each eigenface and getting the corresponding eigenface components/weights

2. Calculate the distance between the external image and face classes of eigenfaces (face classes calculation will be explained later), and use this calculation to classify the image as a face, known or unknown, or not a face (Turk and Pentland, 73).

2 Making Sense of Image Matrices

2.1 Face Averaging Process

To calculate the average face image, a, we want to add up the individual entries for each training vector T_i from the training set of images, $\{T_1, T_2, \ldots, T_p\}$, before dividing by the total number of images p. Put formally, the average face of this set is defined by

$$a = \left(\frac{1}{p}\right) \sum_{q=1}^{p} T_q$$

Let us now apply this process onto a set of example training images. The example images we will use are in color, meaning each entry of the $n \times m$ matrix is a **pixel** composed of three values for red, green, and blue. Thus each matrix representing an image has dimensions $(n \times m \times 3)$. This dimensionality means that each entry in the $n \times m$ matrix consists of three values, as opposed to the one for black-and-white images. For our proofs, we use black-and-white images with $n \times m$ matrices for the sake of brevity. This choice will not affect the applicability of our results to color images, but it is important to note in the construction of our eigenface algorithm. If you are curious about our work with color images, consider checking out our code!

$$\begin{bmatrix} (26,0,67) & \cdots & (38,12,115) \\ \vdots & \ddots & \vdots \\ (100,231,25) & \cdots & (201,45,123) \end{bmatrix}, \ldots, \begin{bmatrix} (128,38,65) & \cdots & (188,236,141) \\ \vdots & \ddots & \vdots \\ (91,117,110) & \cdots & (165,58,80) \end{bmatrix} \Longrightarrow \begin{bmatrix} \frac{(26+\cdots+128,0+\cdots+38,67+\cdots+65)}{6} & \cdots & \frac{(38+\cdots+188,12+\cdots+236,115+\cdots+141)}{6} \\ \vdots & \ddots & \vdots \\ \frac{(100+\cdots+91,231+\cdots+117,25+\cdots+110)}{6} & \cdots & \frac{(201+\cdots+165,45+\cdots+58,123+\cdots+80)}{6} \end{bmatrix}$$

Figure 1: Training Set and its relationship to the average face vector. the six images on the left represent the training images, and the image on the right is that of the average face image

2.2 Face Normalization Process

Using the average face, we can find out how much each training image differs by the equation $r_i = T_i - a$. This difference matrix r_i is known as the **normalized image**.

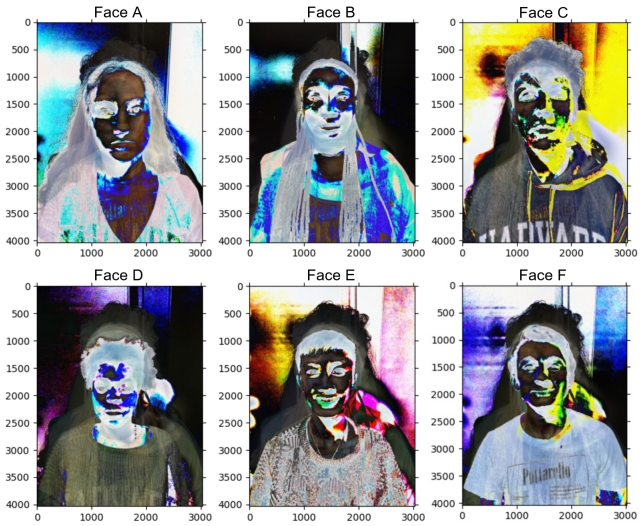

Figure 2: Normalized Faces: From left to right, images are labeled A-F.

Subtracting the average face from our original faces, we get the unnaturally colored normalized faces in figure 2. Regions that appear lighter in the pictures represent areas that deviate starkly from the average image, while dark regions are those that are highly similar between an image and the average. Applying this information to Face A, we can see that there is a large splotch of white and blue where her shirt formerly was. This coloration indicates how the red shirt deviates strongly from the clothing of the average image, which we can see tends to be lighter hoodies and t-shirts. On the contrary, an example of dark coloration in Face A is the face minus the areas around the eyes. Despite the minor differences that allow us to see and differentiate faces, we can look at the average face and see that large swathes of each of these normalized faces are conserved between individuals, such as the cheeks, forehead, and chin. As a result of these similarities, the face regions are dark. This similarity between faces will form the basis for facial recognition!

The matrix subtraction from the equation $r_i = T_i - a$ is shown below for the first training image. If either the face averaging or normalization are particularly interesting to you, be sure to explore the included code!

$$r_1 = \begin{bmatrix} (26,0,67) & \dots & (38,12,115) \\ \vdots & \ddots & \vdots \\ (100,231,25) & \dots & (201,45,123) \end{bmatrix} - \begin{bmatrix} \frac{(26+\cdots+128,0+\cdots+38,67+\cdots+65)}{6} & \dots & \frac{(38+\cdots+188,12+\cdots+236,115+\cdots+141)}{6} \\ \vdots & \ddots & \vdots \\ \frac{(100+\cdots+91,231+\cdots+117,25+\cdots+110)}{6} & \dots & \frac{(201+\cdots+165,45+\cdots+58,123+\cdots+80)}{6} \end{bmatrix}$$

$$= \begin{bmatrix} (26,0,67) - \frac{(26+\cdots+128,0+\cdots+38,67+\cdots+65)}{6} & \dots & (38,12,115) - \frac{(38+\cdots+188,12+\cdots+236,115+\cdots+141)}{6} \\ \vdots & \ddots & \vdots \\ (100,231,25) - \frac{(100+\cdots+91,231+\cdots+117,25+\cdots+110)}{6} & \dots & (201,45,123) - \frac{(201+\cdots+165,45+\cdots+58,123+\cdots+80)}{6} \end{bmatrix}$$

Before we move forward, dealing with image matrices can become clunky very quickly as the math above has shown. It thus becomes useful to represent images not as matrices but instead as vectors. To do this, we must convert the the $n \times m$ matrices to corresponding vectors that have exist in $\mathbb{R}^{nxm}$ space. To make sense of this process, let us consider each pixel to be like a tile. In order to represent the image as a vector, we would have to stack each tile one on top of the other, stacking them from left to right and then from bottom to top. The final result is no longer an $n \times m$ image but a vector (or tower of tiles) of size (or height) of the product of n and m. From here on out in the paper, we will be referring to the vector form of images whenever we discuss images, which includes the training images T and the normalized images r.

3 Transformation of Image Vector Data

3.1 Covariance Matrix

First, recall that **variance** is a measure of the spread between an individual data value and the mean data value of a data set while **covariance** is a measure of the relationship between two variables. If the covariance has a large magnitude, then the two variables have a strong relationship. If the covariance has a small value, then the two variables have a weak relationship. A positive covariance indicates that the two variables move in the same direction, whereas a negative covariance indicates that the two variables move in opposite directions. The **covariance matrix** is a square, symmetric matrix that provides the covariance of two elements of a vector. With more than two elements, the covariance matrix more closely resembles the form of a multiplication table: the entry at the ith row and jth column (or vice versa with ith column and jth row due to symmetry) reveals the covariance value corresponding to the ith and jth features (Cheng). For example, suppose we wanted to find out how often three food items were paired together at parties: bread, hotdogs, and beef patties. The corresponding covariance matrix would be a 3×3 symmetric matrix. Let us assign bread to row and column 1, hot dogs to 2, and beef patties to 3. If we wanted to see how often party guests choose bread with hot dogs, we could look at entry 12 or 21. We would expect this value to be both large and positive (how else are you supposed to eat your hot dog if not with a bun). If we wanted to see the covariance between hot dogs and beef patties, we could examine entries 23 or 32. We would expect the value to be negative because most people grab either a hot dog or a hamburger.

Formally, the covariance matrix C is

$$C = \frac{1}{p}\sum_{q=1}^{p}(r_q r_q^T) = AA^T$$

where the matrix $A = \begin{bmatrix} r_1 & r_2 & r_3 & \dots & r_p \end{bmatrix}$.

Namely, the covariance matrix contains the information representing all of the ways in which the images are different from each other. The eigenvectors of the covariance matrix inform us of the directions of the data, and their corresponding eigenvalues indicate how spread out the data is in that particular direction. Thus, we will be able to represent any given face from the training set as a linear combination of the covariance matrix's eigenvectors. Taken another way, the eigenvectors of the covariance matrix allow us to reverse engineer each of the faces of the training set.

Since each image vector has dimension nm, the covariance matrix is $nm \times nm$. Putting this value into perspective, the covariance matrix corresponding to 1080p images would be in $\mathbb{R}^{2073600}$ space. Our images are even larger than 1080p, so finding eigenvectors for the entirety of this space would be extremely difficult and inefficient due to the sheer magnitude of our vector space.

Instead, we can use what we know about covariance, which is that certain eigenvalues will have greater degrees of variance in the image than others. By ignoring the smaller-valued eigenvalues, we lose minimal amounts of information while making computation significantly easier. So, we want to use the eigenavalues and corresponding images with the greatest variance, as they collect the most "uniqueness" of the vectors. To do this, we will use principal component analysis.

3.2 Principal Component Analysis

Principal Component Analysis (PCA) suppresses redundant information by reducing the number of dimensions, called the principal components, needed to represent a given data set. For example, take a data set that consists of five solid yellow images of varying brightness. In this case, the two potential varying elements are color and shade. Given the similarity in color, the principal component would then be shade. Reduction of these images using PCA could, for example, transform them into five pictures of varying shades of gray. This instance of PCA reduces the dimensions from two (color and shade) to one (shade only).

We must now compute the eigenvectors u_i of AA^T, the covariance matrix. However, the $nm \times nm$ matrix AA^T is extremely large, so it would be difficult to find its eigenvectors, u_i. Therefore, we will compute the eigenvectors v_i of the $p \times p$ matrix A^TA where $A^TAv_i = \mu_i v_i$. This is because all of the eigenvalues of A^TA are also eigenvalues of AA^T, and their eigenvectors are related as follows: $u_i = Av_i$. We know this because

$A^TAv_i = \mu_i v_i$. Multiplying through by A produces $AA^TAv_i = \mu_i Av_i$. So, $CAv_i = \mu_i Av_i$. Because $Av_i = u_i$, $Cu_i = \mu_i u_i$. The nm eigenvalues of A^TA and their corresponding eigenvectors are the largest eigenvalues of AA^T and their corresponding eigenvectors. We must now normalize the u_i eigenvectors, and keep the largest eigenvalues with their corresponding eigenvectors. The vectors v_i are the linear combinations of the p training set of images of faces to create eigenfaces u_s where $u_s = \Sigma(v_{sk}r_k)$ for $s = \{1, 2, \ldots, p\}$. Moreover, the dimension of the face space, as spanned by the eigenfaces, can actually be equated to p', where $p' \leq p$ since only the eigenfaces associated with eigenvalues representing the most variance are needed to form the basis for the face space, which is the subspace spanned by the eigenfaces (Turk and Pentland, 75).

3.3 Spectral Theorem

Before we can conduct PCA, we must answer an underlying question: can we guarantee that we will find an orthonormal basis of eigenvectors (or eigenfaces in this instance)? The Spectral Theorem, or eigendecomposition, tells us that we can. Recall, from our discussion in class, that the Spectral Theorem can examine an $n \times m$ matrix B by studying the $n \times n$ matrix BB^T. Before proving and using this theorem, lets look back at a few definitions from class:

Definition (Lay 397): A matrix A is **symmetric** if $A = A^T$.
Definition (Lay 399): A matrix A is **orthogonally diagonalizable** if $\exists$ an orthogonal matrix P and a diagonal matrix D such that $A = PDP^T$.

Spectral Theorem (Lay 399): Let A be an $n \times n$ symmetric matrix.

1. A has n real eigenvalues, counting multiplicities.
2. A is orthogonally diagonalizable.
3. The dimension of the eigenspace for each eigenvalue λ equals the multiplicity of λ as a root of the characteristic equation.
4. The eigenspaces are mutually orthogonal, in the sense that eigenvectors corresponding to different eigenvalues are orthogonal.

Proof: Suppose A is an $n \times n$ symmetric matrix.

1. We first claim that A has n real eigenvalues, counting multiplicities.
   (a) *Lemma:* If $Ax = \lambda x$ for some nonzero vector $x \in \mathbb{C}^n$, then λ is real

   Suppose $Ax = \lambda x$ for some nonzero vector $x \in \mathbb{C}^n$. $\overline{x}^T Ax = \overline{x}^T(\lambda x)$ by the definition of Ax, and $\overline{x}^T(\lambda x) = \lambda \overline{x}^T x$ by linearity. For any complex number f, $\overline{f}f$, or the multiplication of a complex number with its complex conjugate, produces a positive real number by definition, meaning that $\overline{x}^T x$ must be a positive real number.

   We also claim that $\overline{\overline{x}^T Ax} = \overline{x}^T Ax$. $\overline{\overline{x}^T Ax} = x^T \overline{Ax}$ because $\overline{x}^T = \overline{x^T}$. A is a real matrix, meaning that $\overline{A} = A$ and $x^T\overline{Ax} = x^T A\overline{x}$. Then, $x^T A\overline{x} = (x^T A\overline{x})^T$ because $x^T A\overline{x}$ is a scalar and transposing a scalar does not change it. For two given matrices B, C, $(BC)^T = C^T B^T$, so $(x^T A\overline{x})^T = \overline{x}^T A^T x$. $A^T = A$ because it is an $n \times n$ symmetric matrix, so $\overline{x}^T A^T x = \overline{x}^T Ax$. Therefore, $\overline{\overline{x}^T Ax} = \overline{x}^T Ax$, and complex conjugates are only equal when they are real. Thus, $\overline{x}^T Ax$ is real.

   Because both $\overline{x}^T x$ and $\overline{x}^T Ax$ are real and $\overline{x}^T Ax = \lambda \overline{x}^T x$, λ is by definition the quotient of two real values and therefore must also be real.

   A must have n eigenvalues, including multiplicities, because A is a symmetric $n \times n$ matrix where $\det(\lambda I_n - A)$ is of degree n, meaning the characteristic polynomial has n roots; and by lemma 1a we know that said eigenvalues must be real. Therefore, A has n real eigenvalues, counting multiplicities.

2. We then claim that A is orthogonally diagonalizable. We know that A is symmetric, and by the definition of symmetric $A = A^T$. By the definition of orthgonally diagonalizable, we claim that $\forall$ A, $\exists$ a diagonal matrix D and an orthogonal matrix S such that $A = SDS^T$ (recall that $SDS^T = SDS^{-1}$ in this case, by definition of an orthogonal matrix). We prove our claim by induction over n:

(a) base case ($n = 1$): Suppose our orthogonal matrix $S = [1]$. Then, $A = SAS^T = 1 \times A \times 1 = A$, and A is our diagonal matrix D. Thus, A is orthogonally diagonalizable.

(b) Inductive step: Suppose our result holds $\forall$ $(n-1) \times (n-1)$ matrices. From our work in part 1, we know that all the eigenvalues of A are real. Let λ be some real eigenvalue of A and suppose v is an eigenvector corresponding to eigenvalue λ. By the definition of eigenvalues and eigenvectors and knowing that λ is real, we know that $\det(A - \lambda I) = 0$ and $(A - \lambda I)x = 0 \implies Ax = \lambda x$ has a nontrivial solution $x = v$. Now we define the unit vector $v_0 = \frac{v}{||v||}$ by the definition of a unit vector. Thus, v_0 is a unit eigenvector of A.

We can then expand this set of unit vector $\{v_1\}$ to be an orthonormal basis of $\mathbb{R}^n$ that we will call $\{v_1, \ldots, v_n\}$. We can do this using one of the following processes depending on what A looks like:

i. If A is 1×1, then $\{v_1\}$ is already an orthonormal basis of $\mathbb{R}^1$ and no expansion is needed and $\mathcal{V} = \{v_1\}$.

ii. Recall, from part a, we know that A has n real eigenvalues counting multiplicities. For $n > 1$, suppose λ_1 is an eigenvalue for A with the correspondong unit eigenvector v_1. To form our basis, take v_1 and the standard basis for $\mathbb{R}^n$, which we will denote $\mathcal{B} = \{b_1, \ldots, b_n\}$. We apply the Gram-Schmidt algorithm from class on the set of vectors $\{v_1, b_1, \ldots, b_n\}$. Because v_1 is listed first and all vectors in $\mathcal{B}$ are linearly independent and already span $\mathbb{R}^n$, one of the vectors, b_k where $1 \leq k \leq n$, in $\mathcal{B}$ will be a linear combination of the vectors already in the set (including v_1) and will become the zero vector, meaning it can be removed from the basis. Knowing that v is already normalized, by the Gram-Schmidt algorithm, we produce an orthonormal basis $\mathcal{V} = \{v_1, \ldots, v_n\}$ for $\mathbb{R}^n$ where $v_1 = v_1$.

From this basis, $\mathcal{V}$, we form an $n \times n$ matrix $P = \begin{bmatrix} v_1 & \ldots & v_n \end{bmatrix}$ where $v_1, \ldots, v_n$ are column vectors. By our construction, P is orthogonal, meaning $P^{-1} = P^T$ from our work in class. By definition $AP = AP$, and because $P^{-1} = P^T$, $P^{-1}AP = P^TAP$. Recall that A is symmetric. By the definition of a symmetric matrix, P^TAP is symmetric because $(P^TAP)^T = P^TA^T(P^T)^T = P^TAP$. We then use matrix multiplication to find the first column of this matrix:

$$\begin{aligned} P^TAP \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} &= P^TAv_1 = P^T\lambda v_1 = \lambda P^T v_1 = \lambda \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix}^T v_1 \\ &= \lambda \begin{bmatrix} v_1^T \\ v_2^T v_1 \\ \vdots \\ v_n^T \end{bmatrix} v_1 = \lambda \begin{bmatrix} v_1^T v_1 \\ v_2^T v_1 \\ \vdots \\ v_n^T v_1 \end{bmatrix} = \lambda \begin{bmatrix} \langle v_1, v_1 \rangle \\ \langle v_2, v_1 \rangle \\ \vdots \\ \langle v_n, v_1 \rangle \end{bmatrix} = \lambda \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \lambda \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{aligned} \tag{1}$$

Thus, because $P^{-1}AP$ is symmetric, the first row of $P^{-1}AP = \begin{bmatrix} \lambda & 0 & \ldots & 0 \end{bmatrix}$. Therefore, we can define $P^{-1}AP = \begin{bmatrix} \lambda & 0 \\ 0 & B_{n-1} \end{bmatrix}$ where B_{n-1} is an $(n-1) \times (n-1)$ symmetric matrix. By our induction hypothesis, B is orthogonally diagonalizable, meaning that $B_{n-1} = GDG^{-1} \implies D = G^{-1}B_{n-1}G = G^TB_{n-1}G$ for a diagonal matrix D and an orthogonal matrix G s.t B_{n-1}, D, G are all of size $(n-1) \times (n-1)$.

For the orthogonal matrix G, define matrix $F = \begin{bmatrix} 1 & 0 \\ 0 & G^{-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & G^T \end{bmatrix} = F^T$. Now let $S = PF$. P, F are orthogonal, meaning that $SS^T = (PF)(PF)^T = P(FF^T)P^T = I_n$ and S is also orthogonal. Thus, by plugging in definitions and using our previous work:

$$
\begin{aligned}
S^{-1}AS &= (F^{-1}P^{-1})A(PF) \\
&= F^{-1}(P^{-1}AP)F \\
&= F^{-1}\begin{bmatrix}\lambda & 0 \\ 0 & B_{n-1}\end{bmatrix}F \\
&= \begin{bmatrix}1 & 0 \\ 0 & G^{-1}\end{bmatrix}\begin{bmatrix}\lambda & 0 \\ 0 & B_{n-1}\end{bmatrix}\begin{bmatrix}1 & 0 \\ 0 & G\end{bmatrix} \\
&= \begin{bmatrix}\lambda & 0 \\ 0 & G^{-1}B_{n-1}\end{bmatrix}\begin{bmatrix}1 & 0 \\ 0 & G\end{bmatrix} \\
&= \begin{bmatrix}\lambda & 0 \\ 0 & G^{-1}B_{n-1}G\end{bmatrix} \\
&= \begin{bmatrix}\lambda & 0 \\ 0 & D\end{bmatrix}
\end{aligned}
\tag{2}
$$

Thus, $A = S\begin{bmatrix}\lambda & 0 \\ 0 & D\end{bmatrix}S^{-1} = S\begin{bmatrix}\lambda & 0 \\ 0 & D\end{bmatrix}S^T = SDS^T$ where S is orthogonal and D is diagonal. By the definition of orthogonal diagonalizability, A is orthogonally diagonalizable for $n \times n$ matrices.

So, by induction, this result holds for all square matrices and A is orthogonally diagonalizable.

3. We then claim that the dimension of the eigenspace for each eigenvalue λ equals the multiplicity of λ as a root of the characteristic equation.

   Recall from part b that A is orthogonally diagonalizable, meaning $A = PDP^{-1}$ s.t matrix P is orthogonal and matrix D is diagonal. Let λ be an eigenvalue of A with algebraic multiplicity m. I then claim that I can construct D to be $\begin{bmatrix}\lambda I_{m\times m} & 0 \\ 0 & B_{(n-m)\times(n-m)}\end{bmatrix}$, where $B_{(n-m)\times(n-m)}$ is a diagonal matrix where all entries $b_{ij} \neq \lambda$ s.t $i \leq n-m$ and $j \leq n-m$.

   Let E_λ be the eigenspace with respect to eigenvalue λ.

$$
\begin{aligned}
E_\lambda &= \{x : Ax = \lambda x\} \\
&= F^{-1}(P^{-1}AP)F \\
&= \{x : PDP^{-1}x = \lambda x\} \text{ by our work from part 2} \\
&= \{x : PDP^{-1}x = \lambda PP^{-1}x\} \\
&= \{x : PDP^{-1}x - \lambda PP^{-1}x = 0\} \\
&= \{x : (PDP^{-1} - P\lambda IP^{-1})x = 0\} \\
&= \{x : P(D - \lambda I)P^{-1}x = 0\} \\
&= \{x : (D - \lambda I)P^{-1}x = 0\} \\
&= \{x : \begin{bmatrix}0 & 0 \\ 0 & B_{(n-m)\times(n-m)} - \lambda I\end{bmatrix}P^{-1}x = 0\}
\end{aligned}
\tag{3}
$$

   By the definition of rank, $\begin{bmatrix}0 & 0 \\ 0 & B_{(n-m)\times(n-m)} - \lambda I\end{bmatrix}$ has rank $n-m$, and the rank-nullity theorem from class states that the nullity of this matrix is $n-(n-m) = m$. Recall that we had defined m to be the algebraic multiplicity of A. Therefore, $\dim(E_\lambda) = \text{Nul}(A - \lambda I) = m$ and the dimension of the eigenspace for each eigenvalue λ equals the multiplicity of λ as a root of the characteristic equation.

4. We finally claim that the eigenspaces are mutually orthogonal, in the sense that eigenvectors corresponding to different eigenvalues are orthogonal. Recall that A is a symmetric $n \times n$ matrix. Then, by Lay theorem 7.1, any two eigenvectors from different eigenspaces are orthogonal (Lay 397-8).

Now that we know we will find an orthonormal basis and can perform PCA, we can shift our attention to recognizing faces from unknown images.

4 Recognizing Outside Face Images

To recognize new faces, a different sequence of steps must be followed. Given an image F, we consider both the distance between F and each eigenface, represented as a face class, and the distance between F and the face space in general. We will represent F as a linear combination of the eigenfaces that form the basis for the face space by projecting F onto each eigenface u_k, from $k = 1$ to p'. After calculating the euclidean distances, E_k, for each eigenface and the face space, we recognize that F can exist as one of three possibilities (Turk and Pentland 75-6):

1. Near face space, near a face class (F is a known face) *Note: This is depicted in Figure 3.*
2. Near face space, not near a face class (F is an unknown face)
3. Not near face space, irrespective of whether near a face class (F is not a face)

4.1 Near Face Class Analysis

As with each eigenface, we can calculate the difference r_F between F and the average face of the training set a, where $r_F = F - a$ (see Figure 3). We need to express F as a linear combination of the p' eigenfaces, so the weights are

$$W_F^T = \begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_p' \end{bmatrix}, \text{ where } w_k = (u_k)^T r_F$$

w_k represents the magnitude of the scalar projection of F onto the eigenface u_k, as by the projection formula, since $(u_k)^T r_F = (u_k) \cdot r_F$. Note that the eigenfaces are orthonormal vectors, and as such, the denominator of the projection formula is just 1 in calculating w_k.

To determine which face class best matches with F, we need to find the minimum distance between the weights W_F and W_k, where W_k is calculated by averaging the eigenface weight vectors for each specific known individual in the training set. Thus, $(E_k)^2 = \|W_F - W_k\|^2$. A face belongs to face class k when $(E_k)^2$ is less than some heuristically-determined threshold θ_E (Turk and Pentland, 75-6).

- If $E_k^2 \leq \theta_E$, then this face will fall into either scenarios 1 or 3, depending on the relationship between F and the face space.
- If $E_k^2 > \theta_E$, this face will fall into either scenarios 2 or 3; if image does end up being recognized as a face, a new face class can be created as by adding F to the training set.

4.2 Near Face Space Analysis or Not a Face

Recall from above that $r_F = F - a$ represents the difference between F and the average face a, so r_F is the mean-adjusted input image. We now can express F as a projection onto the face space as $proj_F = \Sigma(w_k u_k)$ from $k = 1$ to p'. The distance between F and the face space is the the squared distance $d^2 = \|r_F - proj_F\|^2$, and is compared to some heuristically-determined threshold θ_D (Turk and Pentland, 75-6).

- If $d^2 \leq \theta_D$, then this image is near the face space, and it is thus concluded that F is an image of a face, falling into either scenario 1 or 2 above (if $E_k^2 \leq \theta_E$ then F is a known face as in scenario 1, otherwise an unknown face as in scenario 2).
- If $d^2 > \theta_D$, then F is not an image, thus resolving F to scenario 3.

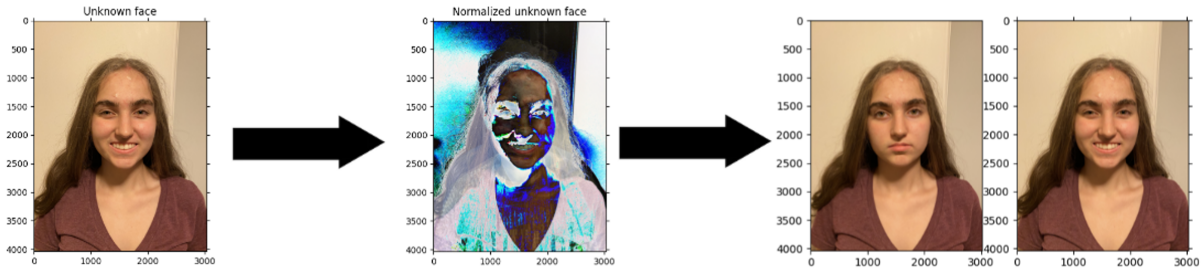

Figure 3: Normalizing an unknown face and matching it to a known face

5 Final Thoughts

5.1 Pros

The eigenface method looks at every pixel in an image as opposed to trying to group certain sections to form different parts of the face (such as the nose, mouth, etc.). By calculating distance, the eigenface approach provides a simple and effective solution to the problem of facial recognition (Coro). This simplicity becomes especially apparent when recognizing new faces and comparing this method with other facial recognition approaches. Looking at new faces, the eigenface approach can recognize if a given image both contains a face and whether said face is similar to images in the training set. If the image is recognized as a face but not one from the training set, a different algorithm can add the image to the training set and rerun the PCA, becoming more accurate for more nuanced variations in faces. Other facial recognition methods like neural networks require significantly more training images and computational power to work effectively, including examples of incorrect or non-face images. Unlike eigenfaces, recursive neural networks depend on many repetitions of the training process to find the best facial identification algorithm (Jaiswal et al.).

5.2 Cons

The eigenface method depends on standardized lighting, facial positions, and background in the facial images that are used in both training and recognition to minimize the unnecessary differences between images. If the variation between images is stronger through lighting differences than actual facial feature differences, then the PCA will focus on lighting rather than actual facial differences (Turk and Pentland 79-81). For example, night vision goggles allow us to see different objects in the dark by focusing in on low levels of light. If too much light floods the goggles, we can no longer see. In the same way, if the variation in lighting becomes too great in a set of images, we go 'blind' and can no longer distinguish between faces because our principal components become light-centric.

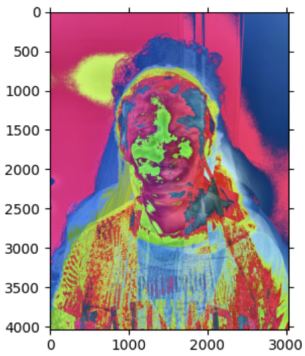

Figure 4: Normalized Face under Variable Lighting

Namely, the eigenface vectors and unknown image vectors will exist in a dimensional space that is three times the space of equivalent black-and-white faces because color uses individual RGB values. When using distance between the image and the face space, there is thus an increased likelihood that an image of a face will be classified as a non-face because of distance arising from color variation and an increased likelihood that a given non-facial image will be classified as one because of similarity in color. This places more emphasis on building a robust average face model to avoid these distance issues (Reddy 3357-3358). For more information about possible error with our average face, check out our jupyter notebook annotations in our code. Much like lighting variation, a solution to this background issue is to use uniform conditions. This solution, however, greatly reduces the practical applications of eigenface as a means of facial recognition, considering most real life images aren't taken in a studio. An alternative solution to the issue of color is conversion of the image to black-and-white, which also reduces the dimensionality of the eigenface vectors (Turk and Pentland 80).

5.3 Applications

There are various real-world applications of eigenfaces for facial recognition technology. Primarily, the method is utilized by the police and the military to find criminals and missing people. When analyzing surveillance footage, officers can utilize eigenfaces and principal component analysis to identify people they are searching for, thus enabling them to track people that threaten society's well-being. Various police stations across the United States have been able to find as well as identify criminals who have robbed stores. Facial recognition technology is also practical in identity verification and biometric authentication systems. Furthermore, facial recognition can be utilized in image and film processing (Wills 12). Eigenface analysis has been a major breakthrough that researchers are continuing to advance in order to identify more specific features in images, such as gender and facial expressions. We hope that you enjoyed learning about facial recognition and found it to be as fascinating as we did.

6 Sources (Each Entry is Hyperlinked)

Cheng, Jer Ming, "Eigenfaces for Dummies." UCLA, 2011.

Coro, Jason, "EECS 598 Fall 2014 Foundations of Computer Vision." University of Michigan, 2014.

"Dimensionality Reduction using PCA." Scikit Learn.

Gentle, James E. "Chapter 3. Basic Properties of Matrices." *Matrix Algebra: Theory, Computations, and Applications in Statistics*, by James E. Gentle, Springer, 2017, pp. 105–127.

Jaiswal, Sushma, Dr. Sarita Singh Bhadauria, Dr. Rakesh Singh Jadon, "Comparison between Face Recognition Algorithm-Eigenfaces, Fisherfaces and Elastic Bunch Graph Matching." Journal of Global Research in Computer Sciences.

Karamizadeh, Sasan et al, "An Overview of Principal Component Analysis." Journal of Signal and Information Processing, 2013.

Laity, Joel, "Principal Component Analysis: Pictures, Code and Proofs." 2018.

Lay, David C., et al. *Linear Algebra and Its Applications.* 5th ed., Pearson, 2016.

Lewis, J.P., "Three Derivations of Principal Components."

Mei, Mike, "Principal Component Analysis." University of Chicago.

Reddy, K. Sunil Manohar, "Comparison of Various Face Recognition Algorithms." IJARSET.

Sierra, Brandon Luis, "Comparing and Improving Facial Recognition Method" (2017). Electronic Theses, Projects, and Dissertations. 575.

Turk, Matthew and Alex Pentland, "Eigenfaces for Recognition." MIT.

Turk, Matthew and Alex Pentland, "Face Recognition Using Eigenfaces." MIT.

Wills, Hunter, "Linear Algebra for Computer Vision." Buzzard, 2014.

Yu, Chen, "Linear Algebra and Facial Recognition." Indiana University.