Multimedia Content-Based Indexing and Recognition in Digital Libraries

Steven Medina
CUNY City College

How does access to this work benefit you? Let us know!

Follow this and additional works at: http://academicworks.cuny.edu/cc_etds_theses

Part of the Computer Sciences Commons

Recommended Citation
http://academicworks.cuny.edu/cc_etds_theses/164
Multimedia Content-Based Indexing and Recognition in Digital Libraries

Steven Medina

September 13th, 2012
Recognition of digital media is more prevalent in today’s computer culture than ever before. The advent of low cost storage has created a seemingly infinite amount of metadata on the internet, as well as on local machines throughout the world. It is more important than ever to have the capability of quickly and accurately filtering this metadata to find a desired result.

Data can be searched using various criteria. For example, text data is searched by analyzing the contents of the text itself. One might execute a search using a method as simple as looking for the ASCII file name, or as complex as parsing large quantities of text and analyzing it with intelligent algorithms. However, searches are not limited to text. Images are also a searchable piece of digital media. Sometimes, an image cannot be searched by file name. Therefore, methods of analyzing images in a digital library are needed to match any input images the user may provide. Both text and image search methods will be discussed throughout this thesis.

We will begin with a discussion of the Eigenface algorithm. This algorithm has become an essential area of study for creating more advanced face/feature recognition algorithms. Before Eigenface was created, face recognition was done primarily by pinpointing key features on a face image, such as the eyes, nose, and mouth. However, this process proved to be slow and inefficient. With the creation of Eigenface, the most distinct features in a face image can be stored as Eigenvalues of a matrix that represents the face image. This method was revolutionary for its time, and will be studied in this thesis with both human and animal faces.

Our next study will be with Scale-Invariant Feature Transform (SIFT). As the name of the algorithm implies, the result of this process is invariant to scale of size, as well as rotation and noise. Unlike Eigenface, this process does not require a frontal-facing style image for both input images and library images. SIFT provides us with a modern algorithm to compare to Eigenface, giving us the ability to see how the original idea of Eigenface has
evolved into a more efficient and effective face detection algorithm.

Moving along to text-based analysis, we will explore the idea of Latent Semantic Analysis (LSA). This type of analysis is very important in search engines and various other kinds of semantic analyzers. LSA makes use of matrices and Singular Value Decomposition to sort out the most frequent and important words and phrases in any piece of text. In fact, this procedure allows us to sort through multiple pieces of text to determine which text is most relevant to our search term [6]. This kind of tool has become very powerful in today’s search engines, and is used by the general population every day.
0.0.1 ACKNOWLEDGEMENTS

I would like to thank Professor Jie Wei for his support in my thesis work and research. His classes on databases and digital libraries were an inspiration which got me interested in image processing and computer vision. Many of the things I have learned about these topics were due to his interesting classes and knowledge.

I would also like to thank Professor Douglas Troeger for his advice on thesis writing and proof reading.

Next, I would like to express my gratitude to my cousin Nestor Medina for proof reading my thesis. His advice was useful in helping me articulate my ideas in an intelligent way.

I would also like to thank my parents, who pushed me to work extremely hard both in school and in life. Their encouragement was a blessing I very much appreciate every day. They are the reason I have certainly developed a thirst for learning that could never be quenched.

Finally I would like to thank my lovely wife Yokasty. She has been nothing but supportive of my educational endeavors, and has given me more than enough motivation to go as far as I possibly can in life.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.1</td>
<td>ACKNOWLEDGEMENTS</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>BACKGROUND</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>EIGENFACE</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>SCALE-INVARIANT FEATURE TRANSFORM</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>LATENT SEMANTIC ANALYSIS</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>EIGENFACE</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>MATHEMATICAL BACKGROUND</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Fourier Series</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>THEORY OF EIGENFACE</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Average Image</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Difference Image</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Covariance Matrix</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Calculate Eigenvectors</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Obtaining Weights</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>RECOGNITION WITH EIGENFACE</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Feature Vector</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Classification</td>
<td>24</td>
</tr>
<tr>
<td>3.4</td>
<td>RESULTS (HUMAN FACE)</td>
<td>25</td>
</tr>
</tbody>
</table>
## List of Figures

1. A theoretical image with pixels labeled in terms of $p$. ........................................ 17
2. A 1-dimensional column vector of length $m \times n$. ........................................ 17
3. A visual representation of a covariance matrix. .................................................. 19
4. Image Library ...................................................................................................... 25
5. Average Image of Images from Figure 4. ......................................................... 26
6. Difference Images .............................................................................................. 27
7. Difference Images .............................................................................................. 28
8. First input image to be tested. ............................................................................ 29
9. Euclidean distances with successfully identified image. .................................. 30
10. Second input image to be tested. ................................................................. 31
11. Euclidean distances with successfully identified image (part 2). ................ 31
12. Cat Image Library ............................................................................................. 32
13. Average Image of Images from Figure 12. ...................................................... 33
14. Cat Difference Images .................................................................................... 33
15. Cat Eigenface Images ...................................................................................... 34
16. Cat input image to be tested. ........................................................................... 34
17. Results of first Eigenface test with cats. ....................................................... 35
18. Second cat input image to be tested. ............................................................. 36
19. Results of second Eigenface test with cats. .................................................. 37
20. Dog Image Library ............................................................................................ 38
Image of a dog and various levels of blurring. The upper-left most dog is the original image. The images that follow it are the five levels of blurring that follow. It should be noted that Matlab uses an algorithm that assumes image pixels outside the boundaries of the image have a 0 value. This is why a progressively darker border appears on the blurred images.
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>Input image compared with a rescaled version of it, while keeping constant length and width ratio.</td>
</tr>
<tr>
<td>42</td>
<td>Results of comparing the input image with the resized image of constant dimensional ratio.</td>
</tr>
<tr>
<td>43</td>
<td>Input image compared with a rescaled version of it, while changing the length and width ratio.</td>
</tr>
<tr>
<td>44</td>
<td>Results of comparing the input image with the resized image of non-constant dimensional ratio.</td>
</tr>
<tr>
<td>45</td>
<td>Input images of a girl. One shows her being serious, while the other shows her smiling.</td>
</tr>
<tr>
<td>46</td>
<td>Results of comparing the the two faces of different expressions.</td>
</tr>
<tr>
<td>47</td>
<td>K-Means Example.</td>
</tr>
<tr>
<td>48</td>
<td>Results of search term “computer”.</td>
</tr>
<tr>
<td>49</td>
<td>Contents of “something.htm”.</td>
</tr>
<tr>
<td>50</td>
<td>Contents of “computers.htm”.</td>
</tr>
<tr>
<td>51</td>
<td>New search results after adding weight to the page count.</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

I often walk down the street and see signs posted on poles and walls. The sign reads “Lost Pet”. The pet is usually a dog or cat. However, it is possible for other pets to be lost, such as snakes, lizards, and hamsters. Most pet owners, especially dog owners, are attached to their pets, and would appreciate any tools available to aid in the search for their lost loved one.

This is only one motivation people have for the need of advanced methods of searching digital media. Technology can certainly come into play in solving search problems with real world applications, such as the one mentioned above. For example, a digital database of lost pets would allow owners to put a picture of their beloved pet in a public repository of lost animals. The key to this repository is the ability to search for a lost pet with nothing but an image. A good Samaritan, or animal control agent, could take a picture of this pet and upload it to this public repository of lost animals. An algorithm would compare the animal’s image to all other lost animals in the database. No other information about the pet would be required, provided the pet’s owner has taken the time to place an image of his or her lost pet in the “lost pet database”. Doing this would require an algorithm that can compare features of a face to faces already in a repository. In fact, we can borrow existing algorithms that are specialized on human face recognition, and apply them to animals. Specifically, we will analyze the Eigenface algorithm, as well as the Scale-Invariant Feature Transform (SIFT).

Text is another form of digital media that is searched. Newspaper articles, cooking recipes, and fictional books are just a few forms of text media that are searched millions of times a day. There are various ways to handle text-based searching, and each method depends on the search criteria involved. This thesis will look to analyze Latent Semantic Analysis.
2 BACKGROUND

2.1 EIGENFACE

When asking an average person about their idea of face recognition, they might suggest addressing key features in a face. Some of these features can include a person’s eyes, nose, mouth, or ears. This idea was certainly the train of thought early computer vision experts had in determining if a person’s face was part of a larger set of faces. Distances were calculated between various key features in a person’s face. These values were compared with corresponding data from the larger set of images (the image database). These ideas were enhanced by adding features such as hair color and the thickness of lips.

A few years later, an approach was created where features from different pieces of the face were mapped on a template. Unfortunately, it was discovered that there are not enough unique data points in the template to accurately represent a face.

After many trials of identifying a human face by its key features, the Eigenface approach was discovered. The advantage of this approach was the fact that feature points were not needed in the original or basis images. Finding feature points is computationally expensive, especially when more features are needed for higher accuracy.

Although we typically think of humans when thinking about face recognition, this thesis will also cover effectiveness of recognizing animal faces.

2.2 SCALE-IN Variant FEATURE TRANSFORM

When one looks at an object in an image, there are certain key points a person’s eyes are attracted to in the image. One might see a picture of a person and think the person in the image has a large nose, big eyes, or bright hair. These are the “key points” of the image.

Eigenface was based on reducing the dimension of matrices using Principal Component Analysis. As mentioned in the Eigenface section of this thesis, we treat each image as a vector in a space of high dimension. A reduced subspace is produced after applying
Principal Component Analysis to the image vector. The name “Eigenface” is applied to the PCA vector. Identification of an input image is made by matching the reduced space of the input image with the reduced spaces of the image library [7].

SIFT is a method that uses keypoints to search for faces in a digital library. In the “Eigenface” Section above, I mentioned that Eigenface was created to avoid the use of keypoints in image searched. However, SIFT uses a method which is much more feature-invariant than the Eigenface method. Scale, brightness, and even noise do not affect the SIFT algorithms ability to search for images in an image library. This method will be fully analyzed, as well as its real world applications [11].

Included in our study of LSA is the K-means Algorithm. This is an algorithm centered around clustering groups of data points for the purpose of analyzing. This clustering algorithm assumes the user inputs the number of desired data clusters. In 1967, James MacQueen coined the term “k-means”. However, the idea of k-means clustering goes as far back as Hugo Steinhaus in 1957. This algorithm has become widely used not only in its original form, but is also the basis for many variations of the k-means algorithm used today.

### 2.3 LATENT SEMANTIC ANALYSIS

Text categorization has become an important part of the most widely used applications of technology. Search Engines, text filtering, and email classification are just a few applications of categorizing text-based data that are used in the daily lives of many people. However, this technology was not always efficient, as words in English, as well as all languages, can exist in different contexts. A method of classifying text is important in improving real-world technology for the advancement of the most widely used applications.

Latent Semantic Analysis (LSA) is a method that stores information about documents in a matrix, thereby allowing efficient matrix operations to analyze and categorize the data [8]. One such matrix operation is Singular Value Decomposition (SVD). SVD is capable of reducing the size of the matrix to build a relationship between the words in a way that uses
the least information possible. This kind of method is space-efficient because it can reduce large amounts of data into a fraction of the size, keeping the most important information to categorize data in an efficient way. When analyzing a library with many books, this can be a very effective method [16].
3 EIGENFACE

3.1 MATHEMATICAL BACKGROUND

Overview

The Eigenface algorithm was the first facial recognition algorithm to ignore “feature matching”. This means it does not involve searching for key features of a face, such as eyes, nose, or mouth. The process of finding key features on a face is very time consuming, and does not lead to a very accurate basis for face detection. Instead, the Eigenface algorithm uses information, similar to that of signal processing, to encode the most relevant face information into a set of basis images. This procedure will be explained in later sections [10].

One assumption the Eigenface approach makes is that face recognition is a 2-dimensional recognition problem. The reason for this is in one of the requirements of Eigenface: The images of faces are to be frontal images of the face. These frontal images give the ability for Eigenface to ignore 3-dimensional information, which reduces the complexity of the problem in comparison to the classical “find key features” approach. The idea of taking a linear combination of images and using them to represent a face in the image library is similar to the idea of a Fourier Series.

Fourier Series

The Fourier Series is named after its creator, Jean Baptiste Joseph Fourier [1]. This theory is well know in the world of signal processing, and is commonly used in image processing and computer vision.

A Fourier series is a representation of a signal as a linear combination of complex sinusoids. If a sinusoid is too complex to process, it is often broken up into a linear combination before further computation is performed. After the necessary computation is finished, it is then combined again into a single sinusoidal expression. Formally, suppose $f(x)$ is a function that is both periodic and satisfies Dirichlet’s conditions. Dirichlet’s conditions are as follows:
1. $f(x)$ is finite, single valued and its integral exists in the interval

2. $f(x)$ must have a finite number of discontinuities in any given interval

3. $f(x)$ has a finite number of extrema in that interval

The Dirichlet conditions are conditions that should be met to show that a periodic, real-valued function, where $f$ is continuous, is equal to the sum of its corresponding Fourier Series. The Fourier Series is shown below.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n(\cos(nx)) + b_n(\sin(nx))]$$  \hspace{1cm} (1)

The coefficients $a_0$, $a_n$, and $b_n$ are considered to be the Fourier coefficients. They can be solved using Euler's formula’s. The derived equations for solving these coefficients are given below.

$$a_0 = \frac{1}{\pi} \int_{c}^{c+2\pi} f(x) \, dx$$ \hspace{1cm} (2a)

$$a_n = \frac{1}{\pi} \int_{c}^{c+2\pi} f(x)\cos(nx) \, dx$$ \hspace{1cm} (2b)

$$b_n = \frac{1}{\pi} \int_{c}^{c+2\pi} f(x)\sin(nx) \, dx$$ \hspace{1cm} (2c)

### 3.2 THEORY OF EIGENFACE

#### Average Image

Using the idea of the Fourier Series, we will combine a library of face images into one image. This image will be called the *average image*. This is analogous to recombining a Fourier Series of signals into one signal. Obtaining the average image is the first step in creating the Eigenface basis.

Before computing an average image, we will assume that all images are of the same
resolution. In other words, the number of row pixels and column pixels are the same on all images in the image library. Given this assumption, an average image of a set of images is computed by adding up all corresponding pixel values in every image, and dividing each sum by the number of images in the image library [2].

In mathematical terms, let there be a set of $I$ images in an image set. Let each image be $m$ pixels high by $n$ pixels wide. For any given pixel value $p$ on each image, such as pixel $p_{xy}$, we add all values of $p_{ij}$ from each image $I$ in the image set. This process starts from pixel $p_{00}$ on each image, and continues through $p_{mn}$. A mathematical expression for the average image is given below.

$$
\text{Avg}(I) = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{m} \sum_{i=1}^{n} I_{ij}
$$

In Equation 3, $I$ represents the current image in the image library, $M$ represents the total number of images, $i$ and $j$ are the current iteration of the row-column pixel pair in an image with $m$ rows and $n$ columns.

Equation 3 can be rewritten to include only one summation for clarity. We will remove the summations iterating $i$ and $j$ and rewrite it as

$$
\text{Avg}(I) = \frac{1}{M} \sum_{i=1}^{M} I_{i}
$$

where $i$ is the current image in the image library being iterated.

In order to correlate the idea of “adding images” with the math behind it, we will envision each image as a matrix. Each image can be represented by a matrix of pixel values. Doing simple matrix addition, we can add any image matrix to any other image matrix. These image matrices are added together until each image in the image library has been considered. Once this sum of image matrices is obtained, it is divided by the number of images in the image library. This is simply the matrix version of finding the mean of a set of numbers [2]. Shown below is an image that illustrates how an image should be considered when iterating.
through pixel values.

![Figure 1: A theoretical image with pixels labeled in terms of $p$.](image)

As can be seen in Figure 1, the image is simply a matrix of pixel values. However, for computational purposes, each image will be displayed as a 1-dimensional column vector rather than a 2-dimensional image matrix. An example of this is shown in the image below.

![Figure 2: A 1-dimensional column vector of length $m \times n$.](image)

Therefore, the 1-dimensional vector will be of length $m \times n$, which is the total number of pixels in the image. This means that the average image will also be stored in the same 1-dimensional column vector format. The reason for storing the image this way will become clear in subsequent sections.
**Difference Image**

A method is needed to extract the most distinguishing features in the average image. The average image is an image that combines everything that is common about each face in the image library. These common features need to be removed, leaving only the distinguishing characteristics of each image in the image library. In terms of math, this is done by subtracting the average image from each image in the image library. If this is thought about logically, it becomes clear that features that are common to all images in the image library are removed. This leaves only the most distinguishing features of each image. The procedure for obtaining each difference image is shown below mathematically.

\[ Diff(I) = I - \text{Avg}(I) \] (5)

**Covariance Matrix**

The covariance defines the amount of change two variables have together. This can be shown mathematically as

\[ \text{COV}(A, B) = E[(A - E[A])(B - E[B])] \] (6)

which is equivalent to

\[ \text{COV}(A, B) = E[AB] - E[A] \cdot E[B] \] (7)

The value \( E[A] \) represents the expected value, or mean, of \( A \). It should be noted that the covariance of \( A \) with itself equals the variance of \( A \). Considering the definition of the expected value being equivalent to the mean of \( A \), the entire definition of covariance can be rewritten once again as

\[ \text{COV}(A, B) = \sum_{i=1}^{M} \frac{(a_i - \bar{a})(b_i - \bar{b})}{M} \] (8)
When applied to a matrix, the covariance matrix is a matrix of the covariance of two variables. Let us assume that we have a vector \( A \) that represents a difference image. In this case, \( A \) would be a column vector. The covariance matrix of this vector would be such that each \((i, j)\) entry would be the covariance

\[
COV(A_i, A_j) = E[(A_i - E[A_i])(A_j - E[A_j])].
\]  

(9)

This is simply a matrix of covariance equations. This can easily be applied to our goal of finding the Eigenface basis, as we need a way of finding variance between each difference image. Since each difference image can be represented by an \( n \times m \) matrix, which in turn can be represented by a column vector, the definition in equation 9 will fit the needs of Eigenface. Visually, this can be seen below.

![Figure 3: A visual representation of a covariance matrix.](image)

Formally, a mathematical representation of Figure 3 can be represented as
\[ \text{COV}(A_i, A_j) = E[(A - E[A])(A - E[A])^T]. \] (10)

Equation 10 shows that the covariance matrix can be represented by multiplying the difference of a vector and its average by the transpose of itself. In other words, we can multiply the difference image by its transpose and take its expected value. Mathematically, this is represented in the following equation.

\[ \text{COV}(A, B) = \frac{1}{M} \sum_{i=1}^{M} \text{Diff}(I_i) \cdot \text{Diff}(I_i)^T \] (11)

Equation 11 can be further simplified by creating a 2-dimensional matrix out of the 1-dimensional difference images. Let \( D \) be a 2-dimensional matrix where each column of \( D \) is a difference image. Then, \( D \) can be represented as

\[ D = \{ \text{Diff}(I_1), \text{Diff}(I_2), ..., \text{Diff}(I_i) \} \] (12)

Using our definition for \( D \), a representation of the covariance matrix is shown below.

\[ C = D \cdot D^T \] (13)

Assuming each difference image is represented by a \( N \times 1 \) matrix, and \( D \) is an \( N \times M \) matrix (where \( M \) represents the number of images), then by rule of multiplying matrices, \( C \) is an \( N^2 \times N^2 \) matrix.

**Calculate Eigenvectors**

Up until now, we have mathematically found a matrix to find the covariance between each difference image. Now it is time to find the Eigenvectors of the covariance matrix. But first, we discuss some background of Eigenvectors.

When a matrix acts on a vector, the direction is often, but not always, changed with
the magnitude. The vectors whose direction are not changed are called *Eigenvectors*. The magnitude by which the vector has changed is called an *Eigenvalues*. Note that the only change of direction an Eigenvector can have is the reverse direction [9].

Finding the Eigenvectors of the covariance matrix \( C \) will yield \( N^2 \) Eigenvectors, each with an \( N^2 \) dimension. Practically, even a powerful home or business computer would run out of memory before the Eigenvectors are calculated. Therefore, an alternate method must be used to achieve the desired Eigenvectors and Eigenvalues [3].

Rather than concentrating on the product \( D \times D^T \), we will look at \( D^T \times D \). We will call the product of \( D^T \times D \) \( L'' \). \( D^T \times D \) gives an \( M \times M \) matrix. Finding the Eigenvectors returns \( M \) Eigenvectors, each of dimension \( M \). This is a very large difference in size and computational power required by a computer. We will call these Eigenvectors \( v_i \).

For reference, the general definition of an Eigenvector states that a non-zero vector \( x \) is an Eigenvector of a linear transformation if it satisfies the Eigenvalue equation

\[
Ax = \lambda x
\]

(14)

where \( A \) is a matrix, \( x \) is a vector, and \( \lambda \) is an Eigenvalue. When applying this to our covariance matrix, we have

\[
D^T D v_i = \lambda v_i.
\]

(15)

Multiplying both sides by \( D \) gives us

\[
DD^T D v_i = \lambda D v_i.
\]

(16)

Therefore, the Eigenvectors of \( C = DD^T \) are \( D v_i \). Using \( v_i \) gives us the \( M \) largest vectors of \( DD^T \). These Eigenvectors represent the Eigenface image set we will use, with one more technique, to search for faces that are similar to the ones in our original image library. We will call the set of Eigenfaces \( \mu \).
**Obtaining Weights**

Now that the Eigenfaces have been obtained, we can use them to represent the difference images as a linear combination of Eigenvectors. This is done by multiplying each Eigenvector by some weight, then adding all the results together. The equation for this, shown below, is

\[
Diff_i = \sum_{j=1}^{K} w_j \mu_j
\]  

(17)

where \( w \) represents the weight of \( \mu \). The weights can be computed with the following formula.

\[
w_j = \mu_j^T \cdot Diff_i
\]

(18)

The operation shown in equation 18 projects images of known individuals onto the face space [3]. This means that for each difference image, \( M \) weights must be computed. These \( M \) weights are stored in a weight vector, which we will call \( \Omega_i \). This weight vector, also known as the basis vector, is illustrated below.

\[
\Omega = \begin{pmatrix}
w_1 \\
w_2 \\
... \\
w_k
\end{pmatrix}
\]

We have successfully built a set of basis vectors to use when finding a face in the image library.
3.3 RECOGNITION WITH EIGENFACE

Feature Vector

The idea behind recognizing an image from our image library is based on the basis vectors that were found when creating the Eigenface basis. When an image is introduced into an Eigenface recognition program, a basis vector must be created using this new image. This new basis vector will be compared to all the basis vectors that have been previously found.

Let us call the input image $\Gamma$. To create a new basis vector out of the input image, we must subtract the average image from gamma. The average image that is used is the same average image that was created when obtaining the Eigenface basis. As with the Eigenface basis, this subtraction gives us a difference image. For quick reference, the equation for this is given below.

$$Diff = \Gamma - Avg$$  \hspace{1cm} (19)

Now that we have the difference image of the input image, the weights of this vector can be computed using $\mu$, the set of Eigenvectors that have already been found. For reference, equation 18 is repeated below.

$$W_i = \mu_i^T \cdot Diff$$  \hspace{1cm} (20)

Now, as with the weights found in the Eigenface basis, we will create a weight vector as a basis vector for the input image.

$$\Omega = \begin{pmatrix} W_1 \\ W_2 \\ \vdots \\ W_K \end{pmatrix}$$
Classification

With out new basis vector, we can compare the input image to the images in the library. This is done by measuring distances between the input basis vector and the basis vectors for the images in the library. In general, a distance measure is computed by

$$\delta = \min \parallel \Omega - \Omega_i \parallel$$

(21)

where each weight vector in the image library is subtracted from the new weight vector. However, simply looking at the smallest distance between input basis vector and library basis vector is not enough. The minimum instance between these two could still be very large. In this case, the input image being recognized is not in the image library at all. Therefore, a threshold value $\Theta$ must be selected heuristically that will determine if the input image is in the image library. If the minimum $\delta$ value is less than this threshold $\Theta$, then this image is in the image library, and is also matched to the image that it shares its minimum $\delta$ value with.

We now have the idea of using a distance measure to compare the input and library basis images for face recognition. However, what kind of distance measure should be used?

For purposes of this research, we will use Euclidean Distance to determine the distance between the basis vectors in question. This distance is given by

$$\parallel x - y \parallel_\delta = \sqrt{|x_i - y_i|^2}.$$  

(22)
3.4 RESULTS (HUMAN FACE)

Before face detection with animals is tested, we first need a base case to compare to. Since face detection was developed with the intent of detecting human faces, it would be a good idea to use human face detection results as the data we compare all animal face detection results to. Therefore, the results of the Eigenface algorithm will be analyzed with human faces before the results of the animal faces are computed.

Image Library

For human face analysis, a library of 15 human faces were used. Each human face that was used had a white background to keep non-facial features from skewing the data in the Eigenvectors. This allows all feature vectors to be made up of data that are facial features without background noise.

The 15 images in the image library are shown below.

As shown in Figure 4, the faces that are chosen are of different sex, color, race, and hair style to give a large variance between the faces. This was done to create a testing environment that covers all major face features.
Average Image

As was mentioned in the Theory section, equation 3 finds the average image of all the images in the image library. The average image will contain features from all images in our image library. In the case of the image library in Figure 4, the average image is shown below.

![Average Image of Images from Figure 4.](image)

As can be seen in Figure 5, the average of all the faces in the image library, if displayed, produce an image that is “ghost-like”. Comparing the average image to the image library, we can see small details from almost every image in Figure 4. The long hair in the average image appears transparent, or “weaker”, because only a small percentage of the faces in the image library have long hair. The solid portions of the image is generally in the face area, where all the images in Figure 4 share that space. It is important to note that all the images in Figure 4 must line up in almost exactly the same spot in references to location to their own image boundaries. This allows key features in each image, such as eyes, nose, and mouth, to line up with each other.

Difference Images

Now that the average image has been obtained, we can use it to find the difference images. Remember, the average image is the average of all the images in the image library, with the assumption that the key features in the image library are lined up. This ensures
that all common features in the image library are stored in the average image. Subtracting
the average image from each image in the image library will result in a matrix (which, in
turn, can be displayed as an image) of all the unique features of each image in the image
library. Displayed below are the resulting difference images from the image library.

![Difference Images](image-url)

Figure 6: Difference Images

Figure 6 shows the resulting images when the average image is subtracted from each
image in the image library. These images have a metallic look. In general, the lighter parts
of each face are features of the face that are unique to that image. The darker areas of
each image are areas that are common to each image. For example, the background of each
difference image is very dark. This is because each of the original images in the image library
had a white background. However, girls with long hair show brightly colored hair. This is
because a small percentage of the images have faces with longer hair, making this feature
unique.

**Eigenfaces**

As noted in the *Theory* section, the difference images are used to find the covariance
matrix, followed by the Eigenvectors. These Eigenvectors are used as our Eigenface basis
for detecting an input image to compare to our image library. This library of Eigenfaces can
also be considered as our projection onto the face space. Theoretically, the input image to
be detected should lie near the face space. Remember that the face space is simply a set of images that look like faces. The original image library is sometimes considered the “face space” because it is assumed that all the images in the image library all have a face-like appearance. Shown below is the set of Eigenfaces computed from the image library shown in Figure 4.

![Eigenfaces](image)

**Figure 7: Difference Images**

These faces, like the difference image faces, are metallic in appearance. These faces will be multiplied with the difference images to find the set of weights associated with each Eigenface. It should be noted that the first Eigenface is not an image with zero-valued pixels. The values in that image, which in turn means the values in that particular Eigenface matrix, have values between zero and four. These are legitimate matrix values that are used to find the matching library image the same way all the other Eigenfaces do.

**Face Detection**

Now that we have our Eigenface basis, we can input an image to be analyzed for a match to the original image library. The input image will be analyzed for its own set of feature vectors. These vectors will be combined to make up our basis vector. Once this is done, the input basis vector will be compared to the basis vector that was found for the Eigenfaces of the image library.
Testing of the Eigenface algorithm will be done in two general phases. First, the input image will be an image that is exactly the same as one of the images in the image library. The second phase will use an image that is similar to an image in the image library. This can be an image of a person in the original image library with a different expression.

We will first start off with an image that is exactly the same as one of the image in the image library. This test case is the most basic test case, as there are no details that make the input image different from any of the original image library images. This image is shown below.

![Figure 8: First input image to be tested.](image)

The image in Figure 8 was successfully identified. However, it is not interesting to point out that the image was found. It is important to note the data behind finding the image. Specifically, the distances of the input images’ basis vector to the basis vectors of the image library. These distances are mapped out in the Figure below alongside the successfully identified image. It should be noted that the images were labeled “image1” through “image 15”. The graph shown in the Figure below show the relation of the images on the x-axis to their Euclidean distances on the y-axis.
As is seen on the graph in Figure 9, image 6 (on the x-axis) has the smallest euclidean distance. This graph highlights the key to determining the matching image in the image library to the input image with distance measures. This result is expected, especially since the input image is an image taken directly from the original image library.

Now that the Eigenface algorithm has worked successfully for the most basic case, we will now analyze the algorithm when a face is input that is similar, but not exactly the same as a face in the original image library. Shown below is the new input image.
The image in Figure 10 is similar to one of the faces in the image library. However, it is not exactly the same, as the face expression is very different (she is smiling). Therefore, it will provide a good test for the Eigenface algorithm. The resulting detected face is shown in the Figure below.

Figure 11: Eucledian distances with successfully identified image (part 2).

The image shown in Figure 11 is the correct image, as it is the same girl without the smile. The graph shown in Figure 11 shows that the fourth image in the image library had the smallest Eucledian distance. Therefore, it was selected as the matching image to the input image. The fourth image in the image library happens to be the image that was chosen (the matching girl without the smile).
3.5 RESULTS (CAT FACE)

Image Library

Eigenface with animals will be based on the same assumptions about the image library. This means that all faces in the image library are lined up in a way so that the eyes, nose, and mouth of each animal are approximately in the same position in the image. Also, each image in the image library should be the same dimensions.

For animal face detection, we will begin with cats. Cats are a common household pet, and are prone to wandering off from their owners. The 10 images in the cat image library are shown below.

![Cat Image Library](image)

Figure 12: Cat Image Library

As shown in Figure 12, the cat faces are of various colors, ages, and types. Also, the cats have different amounts of fur, different sized eyes, and differently shaped ears. Regardless of these differences, these key features still line up with each other in relation to the boundaries of each image.

Average Image

Finding the average image for the cat image library is done the same way as in the human image library. The resulting average image is shown below.
Figure 13: Average Image of Images from Figure 12.

Difference Image

We now have the average image of the cats. This image can be used to find the difference images needed, just as in the human face recognition.

Figure 14: Cat Difference Images

Figure 14 shows the resulting difference images for the cats. Just as in the human difference images, the cat difference images have a very metallic appearance. It is interesting to note that there are more cats that have unique features in the general area of their face, rather than specific features such as eyes or mouth. This is because a cats’ face can vary a lot more than a human face, especially in terms of color and shape. Remember that unique features in the difference image are identified by their brighter colors.
Eigenfaces

The Eigenfaces for the cats were found the same way the human Eigenfaces were found. The difference images were used to find the covariance matrix, which was then used to create an Eigenface basis.

![Cat Eigenface Images](image)

Figure 15: Cat Eigenface Images

Similar to the human Eigenfaces, the cat Eigenfaces have a metallic-like look (with the exception of the first Eigenface). These Eigenfaces are then used to build our basic vector to determine if any input image is in the set of library images. First, we need to select a cat to search the image library for.

![Cat input image to be tested](image)

Figure 16: Cat input image to be tested.

Figure 16 shows the image that will be used to test the Eigenface algorithm for cat faces. Just like the procedure for testing the Eigenface algorithm for human faces, we will first use an image that is taken directly from the original image library as the input image. This will be used as our trivial case. The results are shown below.
As expected, the correct image was selected as the matching image to the input. As can be seen in the graph to the left, the Euclidean distance for fourth vector in the Eigenface basis was significantly smaller than that of the other nine vectors. In fact, this seems to be the trend with detecting all of the cat faces. This is because the variation between cat faces is greater due to their various fur types and colors. However, this large disparity in Euclidean distances is only the case for the input image being the same as one of the images in the image library.

Now that we have seen the Eigenface algorithm work for a trivial case with cats, we will try a case where the input image is similar to, but not exactly the same as, one of the cat faces in the image library. This will test the Eigenface algorithm for effectiveness as a means of matching an image of a pet found in the street with an image already in the database. Shown below is the input image that will be used.
The input image shown in Figure 18 is the same as the seventh image shown in Figure 12. However, the cat in the input image has its eyes closed, its head tilted at a slightly different angle, and its ears pointed in a different direction. At minimum, this will most likely decrease the disparity between the Euclidean distances in the graph shown in Figure 17. This is something we do not want. However, we must remember that the first test case for detecting a cat image using Eigenface was an ideal case. The smaller disparity does not mean the algorithm will incorrectly identify the matching image.

Using the Eigenface algorithm, the results showed that the correct cat image was identified. Even though the input image had some details that were different from its matching image in the original image library, there were enough details stored in the Eigenface basis to differentiate which cat belonged to the one that was input. The results for this test are shown below.
As was predicted, the disparity between the Euclidean distances was reduced. This is because searching for an image that is not exactly the same as any of the images in the image library is not the ideal case. However, this result proves that searching for a cat can be done using the Eigenface algorithm.

### 3.6 RESULTS (DOG FACE)

Now that Eigenface detection on cat faces has proven feasible, the algorithm will be tested on dog faces. Dog faces have an even larger variation of color, shape, and “fuzzyness” than cat faces. This makes testing dog faces worthwhile for obtaining new data.
Dog Image Library and Average Images

For this test, dog faces will be used in the image library. These faces will be of different dog breeds and colors, making for a good sample in testing the Eigenface algorithm. The selected dog faces in the image library are shown below.

![Dog Image Library](image)

Figure 20: Dog Image Library

Finding the average image for the dog image library is done the same way as in the human image library and the cat image library. The resulting average image is shown below.

![Average Image](image)

Figure 21: Average Image of Images from Figure 20.

Although the average image in Figure 21 is still “ghost-like”, it has a vague representation of a dog. The differences in the dogs in the image library are so large that the “ghost-dog” begins to lose its definition. However, we can still use it to find the difference images.
Dog Difference Images

The differences images that are based on the dog average image are very similar to the difference images that were created for the human and cat faces.

![Dog Difference Images](image)

Figure 22: Difference Images

The brighter colors in the metallic-like differences images still represent features that are more unique to that particular dog. Because the shapes of the dogs faces vary so much, the location of the unique features vary as well. Most dogs seem to have their unique features highlighted in the general center of their faces. However, some only have their eyes and nose recognized as truly unique. The background of every difference image is black due to the background in the original image library being white in every image. As with human and cat faces, the darker colors in the difference images represent parts that are not unique to that face.

Eigenfaces

The Eigenfaces for the dogs were found the same way the human and cat Eigenfaces were found. The difference images were used to find the covariance matrix, which was then used to create an Eigenface basis.
Figure 23: Dog Eigenface Images

The results shown in Figure 23 are expected, and they are simply another set of metallic-like dog images. However, meeting the basic expectations of the algorithm is a good sign for the Eigenface algorithm to correctly identify the input dog images.

We now test the algorithm by selecting a dog to use as our input image. Just like the human and cat cases, we will select an input image directly from the original image library as an ideal case.

Figure 24: Dog input image to be tested.

Following the mathematical steps of the Eigenface algorithm, the Eigenfaces in Figure 24 are used to find the Eigenface basis to compare with the input images’ basis vector. The result of this procedure is shown in the Figure below.
As shown in Figure 25, the ideal case is identified correctly. This ideal case allows us to move on to a less trivial case of dog face recognition.

The image below is similar to one of the dogs in the original dog image library. Specifically, it should match up with the 8th image in the image library. The dog in the input image has its head tilted differently. To make this test case more interesting, the 2nd image in the image library is also a white dog of similar style.
Figure 26: Second dog input image to be tested.

With the naked eye, one can see the differences between the input image and the second image in the image library. The ears of the dog in the “second image” are longer, and the nose sticks out more. These differences are easy for a human to interpret. However, the computer needs to interpret this with the Eigenface algorithm. The results of this identification are shown below.

Figure 27: Results of second Eigenface test with dogs.
Looking at Figure 27, the correct face was indeed selected as the matching image. However, the graph on the left side of the image shows interesting results. Remember, the second image, although incorrect, looked similar to the input image with the naked eye. It turns out that the Eigenface algorithm agreed with this. The second image happened to have the second lowest Euclidean distance of all the image. The correctly matched image only (relatively) narrowly beat out the second library image for smallest Euclidean distance. This shows that not only are the identified images correct, but the images that were deemed “not matching” were also ordered correctly in terms of how closely they matched. The Eigenface algorithm agrees with the naked eye.
3.7 CONCLUSION

The Eigenface algorithm correctly identified all humans and animals that were tested. The test cases that were described in this thesis are small in comparison to the number of missing animals that exist in any given town or city. However, the results show that the Eigenface algorithm can be used as a basis for detecting animal faces, specifically common household pets, on a large-scale basis. It is possible that the Eigenface algorithm will not be sufficient for a large scale animal database as it is now. Modifications have to be made to dedicate it to finding animal faces. A missing animal database might have to be broken up into smaller geographical regions to decrease the probability that two animals look almost exactly the same, thereby increasing the chance of a correct match. An animal image database should also be separated into species. For example, dogs and cats should be stored in different animal databases. Regardless of the modifications that need to be made for this to work on a large scale, the results in this test prove that using Eigenface as the basis algorithm to find missing animals is feasible in a real world situation.
4 SCALE-INVARIANT FEATURE TRANSFORM (SIFT)

4.1 MATHEMATICAL BACKGROUND

Overview

There are several ideas taken from Linear algebra that are used in SIFT. Both Laplace and Gauss have mathematical techniques that assist in finding the Scale-Invariant feature transform.

Gaussian Filtering

When thinking of Gaussian Distribution, one often thinks of the classic “bell curve”. This is a distribution of magnitudes where there is a central peak, with a gradual decrease in magnitude (where “magnitude” represents the unit of magnitude for any particular Gaussian application).

The idea of Gaussian filtering is based on convolution. When convolving a signal with a “convolution kernel”, the shape of the signal will become smooth, with less peaks and valleys.

![A bell curve.](image)

Figure 28: A bell curve.

These types of distributions are used as filters for many kinds of signals. In the case of computer vision and image processing, the “signal” is an image. Normally, a one-dimensional Gaussian Filter is represented with the following equation.
\[ g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{x^2}{2\sigma^2}} \]  

\[ \sigma \text{ represents the standard deviation. However, an image is not considered to be one-dimensional. Images have a length and a width, or an x-axis and y-axis. Therefore, the equation for a Gaussian Filter must be modified to apply to two-dimensional signals.} \]

\[ g(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \]  

The application behind Gaussian filtering will be explained in the sections that follow.

**Laplace Operator**

The Laplace Operator is simply the second derivative of a “signal” of any dimension. Once again, our signal is a two-dimensional image. The only stipulation is that the signal must have a second-order differential. This idea is expressed mathematically as

\[ \Delta(f) = \sum_{i=1}^{n} \frac{\delta^2 f}{\delta x_i^2} \]  

The equation above is a general n-dimensional case. Since we are currently dealing with images, a 2-dimensional case is needed for our Laplace operator.

\[ \Delta(f) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2} \]
4.2 THEORY OF SIFT

Scale Space

Scale Invariant Feature Transform is a more powerful image recognition tool than Eigenface. SIFT allows recognition of images regardless of scale, noise, and lighting. These are three image qualities that Eigenface does not specialize in. This allows images taken from different angles and environmental conditions to be compared to each other for detection purposes [4].

To start this procedure, we need to construct a Gaussian scale space. In image processing, the word “Gaussian” is usually synonymous with “blurring”. Blurring an image is done to eliminate (or at least highly reduce) any noise that may be in the image. The greater the blue, the more noise eliminated from the image. However, this comes at the cost of reducing the image quality. Too much blur can render an image unrecognizable.

However, we do not do this procedure only once. The scale space is made up of the difference of Gaussians, also known ad the DoG. Several different versions of Gaussians are taken, each with different sized kernels. The DoG is simply the difference between two of these filtered, or “blurred”, images. The difference in these DoG images is the scale used to filter the image. As mentioned before, the larger the kernel, the more blurred the image becomes. The mathematical expression for a single Gaussian blur operation is shown below [12].

\[ L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \] (27)

The variables \(x\) and \(y\) in equation 28 represent the location of the pixel being processed, \(G\) represents a Gaussian kernel, \(I\) represents an input image, and \(\sigma\) represents the Gaussian kernel. Building on this equation, the DoG can be represented as

\[ D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \] (28)
where k represents the scale factor of the convolution kernel. This multiplies the width of
the Gaussian being used as a filter.

With each level of DoG, the previously filtered image is filtered again. For example, let
$I$ represent the input image. Let $L_0$ represent the Gaussian-filtered version of $I$. $L_1$ would
then represent the Gaussian-filtered version of $L_0$, $L_2$ would represent the Gaussian filtered
version of $L_1$, etc. The difference between two adjacent filtered images is considered to be
a Difference of Gaussian. Therefore, if $N$ filtered images are produced, there can be up to
$N - 1$ DoG images.

Figure 29: Image of a dog and various levels of blurring. The upper-left most dog is the
original image. The images that follow it are the five levels of blurring that follow. It should
be noted that Matlab uses an algorithm that assumes image pixels outside the boundaries of
the image have a 0 value. This is why a progressively darker border appears on the blurred
images.

This process of progressive blurring is not done just one time. The original image is
resized by half several time, and the blurring process is repeated. Each DoG set of resized
images is called an Octave. The original image of the first octave is double the size of the
input image into the SIFT algorithm. The image of each subsequent octave is half the size
of the first octave.
Laplacian of Gaussian Images

When blurring an image, the result is an image where the natural edges and corners remain, while the noisy pixels get blended in with their surrounding pixels. This is optimal for finding key edges and corners in the blurred image.

To find corners and edges, we need to examine each pixel in the image. Then, we look at all the surrounding pixels of the current pixel, and see if there are any large color differences. These large color differences represent either edges or corners. One of the most common techniques of finding edges is called the “Hough Transform”. However, this process is computationally intensive, as it uses the idea of finding the second-order derivative (hence the term “Laplacian”). Therefore, we must find another way to replicate the same results, but with a less computationally intensive method.

Figure 30: Difference images based on the five blurred images from Figure 29

The method we will use to approximate the Laplacian of Gaussians is the Difference of Gaussians. The difference images between each level of blurred is computed. DoG is a very fast approximation, as it turns what would normally be a second-order derivative into a
simple subtraction. The result of a difference image between two variously blurred images is an outline of the most prominent edges and corners.

The images in Figure 30 look very dark. However, upon close inspection, a faint outline of dog can be seen. These images will be the basis for finding the key points, which will be explained subsequently.

**Key Point Detection**

What is a keypoint? Mathematically, a keypoint will be defined as the local maxima and minima of a given part of the difference images. However, the maxima and minima will not be restricted to the image we are processing. It will involve the scale space itself. This means that our maxima/minima set will be based on each pixel’s surrounding pixels, as well as the pixels (and surrounding pixels) on the images above and below it on the scale space. This idea might be hard to visualize. Therefore, Figure 31 is shown to illustrate this point [5][20].

![Figure 31: A representation of finding the local maxima and minima in the scale space.](image)

The red-shaded square represents the current pixel being processed. The surrounding blue squares, both in the current image and the scale-spaced surrounding images, are the pixels we are comparing the current pixel to. If the current (red) pixel is the smallest pixel, it is considered a local minimum. If it is the largest pixel, it is considered a local maximum. If it is considered either a local minimum or maximum, we call this a “local extrema”.

50
Since we started with five Gaussian-blurred images, which gave us four difference images, we can create two sets of maxima/minima images. We will use the second-level difference image, compared with the images above and below it in the scale space, to create our first extrema-detected image. The same will be done with the third-level difference image, as it can be compared with the second and fourth-level difference images to find the local extrema.

Are all the keypoints that we have useful? No. There are two kinds of keypoints that need to be eliminated. These are “low-contrast” keypoints and “edge pixel” keypoints.

“Low-contrast” keypoints are keypoints that have an intensity below a certain user-defined threshold. For example, the programmer of this process might want to exclude all keypoints that have a pixel intensity less than 20. This means that the difference in color on either side of the side or corner that lies on this keypoint is small. For example, the extrema detection algorithm may have considered a shadow that lies on an object to be an edge. This is not true, as either side of the edge of the shadow represents the same object. Therefore, this would be considered a “false edge”. These types of false keypoints need to be eliminated.

Keypoints that are localized on an edge are also considered to be poor candidates for keypoints. This is because an edge can be quite long, and we would have to consider every point on that edge. Therefore, keypoints representing the end of the edge, or its corners, would be better candidates for keypoints. This can easily eliminate many keypoints, making it easier to do further processing after this step.

In order to find keypoint edges, the gradient of the keypoint must be calculated in two perpendicular directions. There are three conditions to consider when eliminating edges.

- **Two Large Gradients** - This means we are at a corner keypoint. We do not want to eliminate these from our set of keypoints. The reason why we know this is a corner keypoint is because taking perpendicular gradients resulted in two large gradients, which means two directions of large contrasting colors. Let us take the example of
looking at a black square on a white background. When we are at a corner, there is a large contrast of color between the corner and two of the perpendicular directions.

- **One Large and One Small Gradient** - In this case, one gradient has a high contrast, and one has a low contrast. This represents an edge pixel. Looking at the image of the black box again, we can see that there is a high contrast of pixel intensity in only one direction when considering an edge pixel. These pixels are eliminated.

- **Two Small Gradients** - When there are two small gradients, the current pixel being considered is not a corner pixel or an edge pixel. Referring to the black box example again, these pixels would either lie in the middle of the black box, or in the middle of the background. These pixels are also eliminated.

![Figure 32: A black square. The large contrast between the black square and the white background make it easier to visualize the gradients of corner pixels.](image)

**Key Point Orientation**

What is the orientation of a pixel? Each keypoint pixel has a direction in which the gradient is the most prominent. It is rare for an image to have a perfect black square in front of a white background. Therefore, keypoints, especially corner pixels, may have strong gradients in more than one direction. We must determine what direction of each keypoint candidate has the strongest gradient in order to find that particular candidate’s orientation.
In order to find the orientation, we will use a “voting method” similar to that used in a Hough Transform. The magnitude and direction (orientation) are calculated for each remaining keypoint in the image. The formulas for these values are as follows.

\[
m(x, y) = \sqrt{[L(x + 1, y) - L(x - 1, y)]^2 + [L(x, y + 1) - L(x, y - 1)]^2}
\]

(29)

\[
\theta(x, y) = \tan^{-1} \frac{L(x, y + 1) - L(x, y - 1)}{L(x + 1, y) - L(x - 1, y)}
\]

(30)

For each keypoint, we round the value of orientation to fit in one of 36 “bins”. This means each keypoint will have an orientation in a bin representing 0-10 degrees, 11-20 degrees, 21-30 degrees, and so on. Each time a direction value falls into a particular bin, we add 1 to the value of the bin, indicating a “vote”. These votes will then be placed in a histogram, indicating the amount of direction values matching each bin.

Figure 33: An example histogram. Any values past 80 percent of the maximum value become new keypoints.
Looking at Figure 33, the largest frequency of angels lies in bin number 1, which would represent 0-10 degrees. However, there are several other frequency values in bins 8, 14, 28, and 33 that are above 80 percent of the maximum frequency. Therefore, the pixels that apply to these orientations will be new keypoints.

**Keypoint Fingerprinting**

Each and every person on earth has a unique fingerprint. Investigators often use this fact to identify an individual who has committed a crime. We need our keypoints to have this kind of unique identification in order to make them useful in comparing our current image with any other image that may have similar features.

In order to create a fingerprint for each keypoint, we create a 16x16 grid around each candidate keypoint. This 16x16 grid will be divided into 16 4x4 grids. In each 4x4 grid, orientations are computed, and placed into an orientation histogram. The ranges of angles will be from 0-44, 45-90, and so on. This, of course, is simply dividing 360 degrees into 8 parts, creating a 8-bin histogram.

To backtrack, we will have a 16x16 grid for each candidate keypoint. The grid is divided into 16 4x4 squares, computing an 8-bin histogram. This will give us a total of 16 4x4 regions times 8 bins = 16x8 = 128 values for each keypoint. In other words, each keypoint is identified by a 128 element vector. The possibility of having two keypoints with exactly the same 128-element vector in the same image is virtually zero. Therefore, these vectors provide use with a good unique identifier.

It is best to normalize each vector before storing them. We normalize by dividing the vector by the square root of the sum of its squares. It should be noted that these vectors are commonly referred to as “feature vectors” [18].
Using for Image Comparison

The theory of Scale Invariant Feature Transform is now complete. It can now be used to compare different images with each other. For a single comparison, both input images have their feature vectors computed. Then, each feature vector of image 1 has their features compared with all the features of image 2. However, given two images being compared, with image 1 having \( m \) features and image 2 having \( n \) features, there are \( mn \) possible feature pairs. Rather than analyzing all of them on the fly, we will first create an error margin for each feature pair. These error margins will then be ordered, and we will only process (compare) feature pairs above a certain threshold. After all, it would be silly to compare two features that have almost 0 percent compatibility. To compute the error, we use:

\[
\text{error}_{mn} = \sum_{i=0}^{m} \sum_{i=0}^{n} |v_1 - v_2|
\]

(31)

where \( v_1 \) and \( v_2 \) are the feature vectors of image 1 and image 2.
4.3 RESULTS

Using a stock image of a face from the internet, we will test different properties of SIFT. The input image that will be used is shown below.

![Stock image from the internet that will be used to test SIFT.](image)

Figure 34: Stock image from the internet that will be used to test SIFT.

Rotation

In our first test, we will rotate the original image by 30 degrees. The Eigenface algorithm would have failed a rotational test. However, we will see if SIFT can do better. Shown below are the two images side-by-side for a pre-processing comparison.

![Input image rotated by 30 degrees.](image)

Figure 35: Input image rotated by 30 degrees.

Now that we have a clear understand of what is being tested, the SIFT algorithm will be applied for this test. Remember, the test should work independent of rotation, as we are
simply computing feature vectors of the image.

Figure 36: The results of comparing the input image with the rotated image.

Figure 38 shows the two images with many lines connecting from one image to the other. These lines are connecting the feature vectors that are similar in both images. As can be seen, there are many such feature vectors that exist. This is a good thing, as this means the images are considered to be a match. Specifically, there are 760 keypoints in the first image, 733 keypoints in the second image, and 562 matching keypoints between the two images. This overwhelming number of matching keypoints suggests that there is an object in image 1 that can be mapped to image two. Specifically, the two faces are the same. The matching features shown in Figure 38 show that the SIFT algorithm works to match images independent of rotation. This is a strong feature of Scale Invariant Feature Transform that the Eigenface algorithm lacks.

**Brightness**

Our second test will be a test on various lighting levels. The input image has been brightened to create an obvious difference in brightness between the two images. Both images are shown below for comparison.
Figure 37: Input image with a significant brightness difference.

The image on the right of Figure 37 has been brightened enough to exploit any errors the SIFT algorithm might have with differing brightness levels. The results of this test is shown below.

Figure 38: The results of comparing the input image with the brighter image.

As can be seen in Figure 38, there are many keypoints found, indicating a match between objects in the images. Image 1 had the same 760 keypoints detected, while image 2 had 552 keypoints found. The drop in number of keypoints detected by SIFT can be expected because brightening the image cause some edges and corners to disappear. However, this did not affect the number of keypoints matched. There were still 508 matching keypoints between the images. That is about 67 percent of the keypoints found in the first image, and
92 percent of the keypoints found in the second image.

Noise

In our third attempt to break the SIFT algorithm, the input image has Gaussian noise added to it. We will now see if adding noise to an image will affect the SIFT algorithm’s ability to detect a matching object. The noisy image is shown below, with a side-by-side comparison with the original input image.

Figure 39: Input image with a Gaussian noisy image.

The first step in the Scale invariant Feature Transform is to blur the image. This step is made to reduce the affect of noise on the algorithm’s ability to detect matching objects and images. This test will determine if blurring the images will work.

Once again, the SIFT algorithm has proved itself useful. Although the noisy image would normally create a large contrast between many of the pixels in the image, blurring the image before finding gradients reduced the probability of finding false positive matches on keypoints. The second image had 637 detected keypoints. It should be noted that the number of matching keypoints was reduced to 109. However, this is still enough to determine that there are one or more matching objects in both images.
Scaling

When the size of an image is increased, pixels are often interpolated with the surrounding pixels to either fill in blank spaces of a larger image. Conversely, decreasing the size of the image causes the information of many pixels to be forced into a single pixel. This is certainly a change worth testing the SIFT algorithm over.

This procedure will test image scaling when the ratio of the image is kept constant. Although the resized image will be larger, the ratio of width to height will be the same. We will then test with a resized image with different scaling. Because this procedure tests scaling, it is important to note the sizes of each image for each test.

- The input image has a 400x400 resolution.
- The resized image with same ratio has a 600x600 resolution.
- The resized image with different ratio has a 600x400 resolution.
Shown below is a side-by-side comparison of the first set of images being tested.

Figure 41: Input image compared with a rescaled version of it, while keeping constant length and width ratio.

This situation presents another opportunity for the Gaussian filter to blend interpolated pixels to reduce the amount of noise in the image. Pixels that were interpolated into the larger image do not exist in the smaller image. The results for this test are shown below in Figure 42.

Figure 42: Results of comparing the input image with the resized image of constant dimensional ratio.

As shown in the results, the algorithm worked regardless of the scaling of the image. The amount of keypoints in the larger image were increased; it had 977 keypoints compared to
the input images 760. There were 548 matching keypoints between the two images. This is a very large amount of keypoints, making the SIFT algorithm very strong against image scaling with constant ratio.

Now, the input image will be tested against a larger resized version of it.

Figure 43: Input image compared with a rescaled version of it, while changing the length and width ratio.

It is reasonable to think that there is a larger probability of this scaling with different ratios causing the SIFT to fail. However, the results shown in Figure 44 show otherwise.

Figure 44: Results of comparing the input image with the resized image of non-constant dimensional ratio.

Judging by Figure 44, it seems that there is a clear matching of objects from one image to the other. It should be noted that, although a large number of keypoints were matched, there was a large drop in the number of keypoints matched for scaling of different ratio.
There were 640 detected keypoints to the input images 760, and only 182 keypoints were matched. This is a large drop from the 548 matched keypoints of constant-ratio scaling.

**Face Detection with Different Images**

We already know that Eigenface is a great algorithm for detecting faces. Given the limitations that each face must be front facing and centered, the algorithm performs quite well. However, how does the SIFT algorithm do in a similar situation? To test this, we will use two images of a girl; one image gives her a serious face, and the other image shows her smiling. The two images are shown in Figure 45.

![Figure 45: Input images of a girl. One shows her being serious, while the other shows her smiling.](image)

As can be seen in Figure 46, there are very few matching keypoints between the two
images using the SIFT algorithm. The change in facial expression simply changes the subtle keypoints that each image uniquely has. There were only 8 matching keypoints between the two images.

4.4 CONCLUSION

The Scale-Invariant Feature Transform is useful for detecting images that have undergone a form of manipulation. On the internet, it is possible to find the same image in different sizes, rotations, crops, and brightnesses. SIFT is a great tool for finding matching images when a text label of the image is not an option to include in search parameters.

When comparing faces with different expressions, the basic SIFT algorithm fails to match the two images with confidence. Eigenface performed much better at this task. However, this does not take away from SIFT’s ability to be invariant on so many image alterations, such as scaling, luminosity, rotation, and noise.
5 LATENT SEMANTIC ANALYSIS

5.1 MATHEMATICAL BACKGROUND

LSA makes heavy use of complex matrix operations. The two main matrix operations used in latent semantic analysis are Eigenvectors/Eigenvalues and Singular Value Decomposition (SVD). Since Eigenvectors were already discussed in the “Eigenface” section, we will talk exclusively about SVD.

Singular Value Decomposition

SVD is a matrix procedure that reduces the dimensional size of the matrix while keeping the most relevant information. The result is a matrix that reduces/eliminates the noise surrounding the data in the original matrix [17]. The information it keeps are usually patterns. This is especially useful when large matrices need to be processed further.

The equation for SVD can be expressed as shown below.

\[ A = USV^T \] (32)

If \( A \) is an \( nxm \) matrix, then the dimensions of the variables on the right side of the equation are as follows.

- \( U \) is an \( mxm \) matrix
- \( S \) is an \( mxn \) matrix
- \( V \) is an \( nxn \) matrix

Given matrix \( A \), we first use this matrix to find the Eigenvalues and Eigenvectors of \( AA^T \), which make up the columns of \( U \), and \( A^T A \), which make up the columns of \( V \). The square roots of the Eigenvalues of both \( AA^T \) and \( A^T A \) are used to find the singular values of \( S \). The arrangement of \( S \) is a diagonal matrix with its values stored in decreasing order.
K-Means Clustering

Clustering is an important aspect of semantic analysis. It involves grouping data points into various “clusters” to organize and classify data any way the user needs. This algorithm takes user-defined input before it can work. The user of the algorithm needs to specify the desired number of centroids, as well as the initial location for each centroids. The initial location for these centroids should be placed as far apart from each other as possible, as well as between the minimum and maximum values of the data itself[15].

After the number of clusters, as well as their initial locations, have been input by the user, the algorithm begins to work. Each datapoint finds the centroid it is closest to. This process means a measure of distance must be defined. One common distance, which will be used in this thesis, is Euclidean Distance, which is defined in equation 22 and will be repeated and modified below as follows [13][14]:

$$\sum_{i=1}^{N} \sqrt{(x_i - y_i)^2}$$ (33)

After each data point finds the centroid it belongs to, each centroid recalculates itself based on the datapoints each of them own. This is done by taking the average of all the points belonging to each centroid. The value of this average will become the new position of each centroid.

After each new centroid is computed, the entire process is repeated. Each data point groups with the centroid that is closest to it. After each data point belongs to a centroid, the average of the data points in each centroid is computed, and the new centroids are found. This process continues until the centroid values from the previous iteration are equal to the centroid values from the current iteration.

Shown below is a series of images showing a program running the K-Means Algorithm.
In the example shown in Figure 47, ten data points are entered by the user, followed by the number of centroids. The program then randomly selects three initial centroids. In the first iteration of calculations, the program assigns each data point to a centroid. The average of the data points for each centroid is then taken, and assigned as the new value for each centroid. In the second iteration, the process is repeated, and the new values of the centroids after the second iteration are the same as the values of the centroids after the first iteration. The program then ends, and the values of the centroids are finalized.
5.2 THEORY OF LATENT SEMANTIC ANALYSIS

LSA analyzes text data to find the hidden patterns and underlying meanings of these documents [19]. For every piece of text, and for every word in these texts, there is a vector with elements corresponding to these concepts that can express them. In other words, we are trying to find the semantic relationship between words and the text data that contains them. Using SVD is necessary because words can have different meanings depending on the context in which they are used.

The first thing that must be done before any complex matrix operations are performed is to create a histogram representing a word count. The number of words of each document is counted and temporarily stored. The exception to this rule is to discount any common words that have little to no meaning. Some examples would be “the”, “it”, “a”, “and”, and so on. This is because these words will almost always occur more than any other words in a large document. However, they also have no meaning in any relevant search. The resulting histogram is used to create a matrix. The rows of this matrix represent the 10 highest counted words. The columns represent the documents that the words were read from. Let us call this matrix $A$. Now that matrix $A$ has been obtained, we can manipulate this matrix to give us all the information we want about the text documents we are examining.

If matrix $A$ is an $mxn$ matrix, then we can multiply it by the transpose of it self. We will do this twice. The first multiplication will be $A^T A$, which will give us an $m x m$ dimension matrix. We will call this matrix $M$. The second will be $AA^T$, which will give us an $nxn$ dimension matrix. We will call this matrix $N$.

What is the significance of both these matrices? We will start with matrix $M$. $M$ is a matrix consisting of the number of words that documents $i$ and $j$ have in common. In other words, the value of $M_{ij}$ is the number of words that documents have in common. Given $x$ documents, with each document labeled as $d_x$, if documents $d_2$ and $d_5$ have 3 words in common, then $M_{2,5} = 3$.

Matrix $N$ gives the opposite relationship. This matrix stores the documents that search
terms $a$ and $b$ have in common. Therefore, if search terms $a$ and $b$ are both found in document $d_i$, then $N_{a,b} = i$.

Our goal is to use the information we have gathered so far for singular value decomposition. The SVD will give us the best match for any input search term to a given document. Remember, the formula for SVD is

$$A = USV^T$$ (34)

Our goal will be to find diagonal matrix $S$. At this point, we only have matrix $A$. However, matrices $U$ and $V$ can be found from matrices $M$ and $N$. In order to find matrix $U$, we use the Eigenvectors of matrix $M$ as the columns of $U$. We can then use the Eigenvectors of $M$ as the columns of $V$.

Now, we must find matrix $S$. To do this, we take the square roots of the Eigenvalues of matrix $N$. These Eigenvalues produce a diagonal matrix that happen to be in decreasing order. Depending on the number of search terms and documents being searched, this matrix can get very large. However, we can heuristically reduce the size of the matrix. One good method is to keep the number of terms equal to the square root of the number of terms in the matrix. For example, if matrix $S$ is a diagonal matrix with 100 terms, keeping the top 10 results would be enough to accurately determine which documents contain the closest match for the user query. This also reduces the size of matrices $U$ to 10 columns, and $N$ to have 10 rows. We will call these new reduced-size matrices $U'$, $S'$ and $V'$. Computationally, this will be very useful.

The search terms can now be represented by $U'S'$, and the documents can be expressed as $S'V'T'$. With this data, the search terms can be expressed as the centroid of the search term vectors. We will call this value $c$. The documents must now be ranked in relation to the search terms. Given what we have found, we can compute these ranks with the following equation:
\[
\frac{d_i \cdot q}{|d_i||q|}
\] (35)

The distance between each document and the query term from the computed centroid value. The document with the smallest distance to the query centroid value is the highest ranked document. This document would then be returned to the user as the document with the closest match to his query.

In this thesis, we will study the results of this algorithm for use in ranking web pages. To do this, we will rank webpages based on their links to other web pages, as well as the text they contain.

### 5.3 RESULTS

For the purpose of this thesis, a custom built search engine was developed to test the idea of latent-semantic analysis. The search engine parses html files for key words, and ranks each “web page” by user-input keyword. The names of each html file are also relevant to the content of the files themselves. The names of the html files are as follows.

- cell_phones_link.htm
- colorful_girl.htm
- computer.htm
- computers.htm
- computers_link.htm
- copiers_link.htm
- flash_drives_link.htm
- hard_drives_link.htm
- moniters_link.htm
- motherboards_link.htm
- networking_equipment_link.htm
The first search term we will try will be “computer”. Judging by the names of the html files, the term computer will be in several web pages. However, this will make a good first test.

Figure 48: Results of search term “computer”.
As shown in Figure 48, the search results contain HTML files related to computers and technology. However, they are shown in the order of closeness to the input search term. According to the search, the web page “something.htm” was the number one search term. The contents of this file reveals the reason for the high rank of this page.

Figure 49: Contents of “something.htm”.

The file “something.htm” contains the word “computer” repeated 13 times. This certainly gives a good reason for “something.htm” to have a high rank as far as closeness to the search term. This, of course, was designed to give an exaggerated result. The second highest ranked page to the search term is “computers.htm”. This is shown below.
The page in Figure 50 shows a web page that is more likely to show up with the search term “computer” on the actual internet. This is because there are more factors included in real life web-based searching than looking up the keyword using LSA. For example, one might also consider the number of times a web site has been visited, or the general page rank given to a web site by the search administrator. For example, if we put more weight onto the number of times a website is visited, the search results for the same keyword change.

With weight being added to the page count, the search results now change. The web page “something.htm” is now eliminated from the search results entirely. At the same time,
the page “computers.htm” is now the highest ranking web page. This is more applicable to real life than simply using LSA by itself.

5.4 CONCLUSION

Latent Semantic Analysis is the basis for many forms of text based searches. Google’s famous Pagerank algorithm uses a more complex form of LSA. This algorithm makes advanced search engines possible, forever changing the way we look up information. Research to find information that used to take hours can now be done in a matter of minutes thanks
to creative text-based search algorithms such as LSA.
6 APPENDIX

6.1 REFERENCES


17. Deerwester, Scott, Susan T. Dumais, George W. Furnas,

