Emerging Research Surrounding Power Consumption and Performance Issues in Utility Computing

Ganesh Chandra Deka
Regional Vocational Training Institute (RVTI) for Women, India

G.M. Siddesh
M S Ramaiah Institute of Technology, Bangalore, India

K. G. Srinivasa
M S Ramaiah Institute of Technology, Bangalore, India

L.M. Patnaik
IISc, Bangalore, India

A volume in the Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Emerging research surrounding power consumption and performance issues in utility computing / Ganesh Chandra Deka, G.M. Siddesh, K.G. Srinivasa, and L.M. Patnaik, editors.
pages cm
Includes bibliographical references and index.
ISBN 978-1-4666-8853-7 (hardcover) -- ISBN 978-1-4666-8854-4 (ebook) 1. Electric utilities--Management--Data processing. 2. Electric utilities--Information technology. 3. Distributed generation of electric power--Data processing. 4. Electric power consumption--Data processing. 5. Demand-side management (Electric utilities)--Data processing. 6. Management information systems. I. Deka, Ganesh Chandra, 1969-
HD9685.A2E573 2016
338.4'7004--dc23
2015022447

This book is published in the IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) (ISSN: 2327-3453; eISSN: 2327-3461)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 20
Face Recognition:
A Tutorial on Computational Aspects

Alexander Alling
University of Rochester, USA

Nathaniel R Powers
University of Rochester, USA

Tolga Soyata
University of Rochester, USA

ABSTRACT

Face recognition is a sophisticated problem requiring a significant commitment of computer resources. A modern GPU architecture provides a practical platform for performing face recognition in real time. The majority of the calculations of an eigenpicture implementation of face recognition are matrix multiplications. For this type of computation, a conventional computer GPU is capable of computing in tens of milliseconds data that a CPU requires thousands of milliseconds to process. In this chapter, we outline and examine the different components and computational requirements of a face recognition scheme implementing the Viola-Jones Face Detection Framework and an eigenpicture face recognition model. Face recognition can be separated into three distinct parts: face detection, eigenvector projection, and database search. For each, we provide a detailed explanation of the exact process along with an analysis of the computational requirements and scalability of the operation.

INTRODUCTION

The ability of human beings to detect and recognize objects in our surroundings is currently unmatched by even the most powerful computer. We are able to nearly instantly and effortlessly recognize objects, even if we have only seen them once, or only heard a description, beforehand. We can tell the difference between very similar objects, and determine what we are looking at when it is rotated, or even partially obscured. Perhaps most impressively, we can pick out the subtle differences required to recognize other humans with only a quick glance.

DOI: 10.4018/978-1-4666-8853-7.ch020

A computer is far less capable of the same feats of object recognition. Where man can instinctively learn to recognize new objects and people, a computer must complete a much more arduous process. A computer must be explicitly taught exactly what it is looking for. Generally, this requires many images of an object to be compiled and analyzed. After that, new instances of the object can be found by the computer. For complex objects (such as a human face) this is far easier said than done. There are tens or hundreds of small details that make up the visual identity of an object. Figuring out which ones to focus on and analyze is not always straightforward. Actually picking them out of a sample image can be very computationally demanding for complex objects.

There are two separate, but related parts of this problem: face detection and face recognition. Detection is finding any and all human faces in an image. Exactly whose face it is doesn't matter. Recognition, conversely, is connecting a specific name to a specific face. These two operations together are what allow a computer to pick a face out of an image and identify them.

FACE RECOGNITION PROBLEM

A facial recognition application is a computer program that identifies faces in a scene presented to it and matches them to its own database of people. This type of application was once considered one of the most difficult to achieve goals in the field of computing, though significant strides have been made since the early days of computer vision. There are many different schemes available for computer object recognition. There are two broad categories of objection recognition methods, which have some overlap. They are photometric (appearance) based methods and geometric (feature) based methods. Photometric applications use things like skin color and a person's contrast with the image background to aid in face recognition. A simple method based on features is edge-detection, finding discontinuities in image brightness or color and comparing them to a database. Many other recognition algorithms are based on this same idea: finding regularities (or irregularities) in an image, and comparing them to a database. Many different types of features can be picked out and compared. Edges, corners, and 'blobs' are the three main types. The most diverse type is blob detection, where a blob is a region of a picture with a consistent property (color, intensity, etc.) that differs from the surrounding regions (Lindeberg, 1991).

In the mid 1960's, the first successful face recognition system was developed by Woodrow Wilson Bledsoe. It was almost entirely a manual operation where a worker would have to record the coordinates of facial features such as the center of each eye, the tip of the nose, and so on. On average, each face's dataset could be recorded in about 90 seconds (FBI, 2011). A computer was then used to determine which face from the database most closely matched the new face. In the 1970's, Goldstein, Harmon, and Lesk created a similar system based on features such as hair color and lip thickness (Goldstein, Harmon, & Lesk, 1987). Truly autonomous face recognition quickly gained steam after that. In 1988, the eigenface method was developed by Lawrence Sirovich and Michael Kirby. Their algorithm using Principle Component Analysis is still one of the primary methods for real time face recognition (Sirovich & Kirby, 1987). In recent years its accuracy in varied lighting and angle settings has been surpassed; but it is still one of the best freely available algorithms capable of running in real time.

Recognizing faces in real time is a difficult task because of the complexity of categorizing the features of a human face. Detecting those faces in large images is difficult because of the large amount of background data that has to be sifted through. The first face detection algorithm capable of operating in real time was developed in 2001 by Paul Viola and Michael Jones. The Viola-Jones object detection

framework is still today considered both the fastest and most accurate face detection algorithm. It is the algorithm in use on nearly every smart phone and camera capable of detecting faces.

The structure of the complete face recognition application contains three steps, and utilizes two of the aforementioned methods. The first being face detection, followed by the related operations of projection and database search. The Viola-Jones detection method is capable of quickly locating the general features in an image (such as eyes and a nose), but is not able to match these features to a specific person. As such, it is capable of finding the faces present in an image, but it is not capable of recognizing a specific person (Viola & Jones, 2001). Conversely, the eigenface recognition method is specifically made to compare and contrast individual faces. However, it requires these faces in a consistent format, and is not able to work with a face before it has been cropped from a complex scene. As such, these two methods complement each other and are used together.

The initial face detection is therefore done with the Viola-Jones object detection framework. The detected faces are cropped from the image, and handed off to the eigenface framework. The eigenface work done can be split into two operations. The first is projecting the detected faces into an eigenvector. This is done with a computationally intensive set of matrix multiplication. The final step is to compare the eigenvector with the precompiled database of eigenfaces, and determine which original face the new image most closely resembles. When the complete face recognition application is split up like this, the three components can be referred to as *Face Detection* (FD), *Projection* (PJ), and *Search* (S).

FACE DETECTION (FD)

The Viola-Jones object detection framework was proposed in 2001 by Paul Viola and Michael Jones, and is the first object recognition algorithm considered capable of operating in real time. It is, essentially, a type of blob detection algorithm. This implementation of blob detection utilizes Haar-like features, which are pixel intensity sums and differences between rectangular sections of an image (named for their similarity to Haar wavelets). More simply, objects have consistent areas where color intensity changes by a known amount. By finding these same changes in a sample image, an object can be recognized (Papageorgiou, Oren, & Poggio, 1998). The Viola-Jones framework operates on this same principle, in a slightly more complex form. The features are combinations of multiple rectangular sections. Known patterns of features can be found in the image and be identified as known pieces of the image database (such as eyes, cheeks, etc.).

The framework has three key features that allow it detect faces with a high success rate in real time, as follows:

- **Integral Image:** A method for representing the sum intensity of subsets and features of an image in an efficient format.
- **Face Feature Classifiers:** A strong classifier is constructed from multiple weak classifiers that recognize Haar-like features.
- **Cascade Classifiers:** Negative areas are ruled out quickly while possible faces are more rigorously tested.

Another name for an integral image is a summed area table. It is a data representation that allows values of a rectangular subset of an image or grid to be summed efficiently (Crow, 1984). In the context

of face detection, the value of a given point is the sum of all the pixel intensity values above and to the left of the point. The entire summed area table can be computed easily, as each point can be found by adding its value to the summed areas of the points directly above it and to the left of it, and subtracting their overlap. Once the table is complete, any rectangle's sum can be found with just four values. The shaded rectangle in the image below can be summed by adding the area value of point C to point A, and subtracting both B and D, as shown in Figure 1. This operation takes a constant amount of time regardless of the size of the area summed.

As the Haar-like features used by the Viola-Jones algorithm are all combinations of two, three, or four equally sized rectangles sharing edges together, their intensity sums and differences can be calculated very quickly (Yang, Kriegman, & Ahuja, 2002) . The main mechanism by which the framework functions is the use of classifiers and Haar like features. To detect a face in a sample image, simple rectangular features characterizing a face are located. There are many tens of thousands of possible Haar features, but a much smaller number are useful in detecting a face. To determine which features are used, a variant of the machine learning algorithm AdaBoost (Adaptive Boosting) is used. It combines multiple weak classifiers and iterates through them, forming a single strong classifier by creating a weighted average (Freund, Schapire, & Abe, 1999). Once classifiers are trained, a small set of Haar features can be located in a sample image. Shown in Figure 2 are two of the best face identification features, recognizing that eyes are darker than the cheeks below them or the bridge of the nose between them. These two features can successfully identify almost 100% of faces in images. Unfortunately, they are broad enough to apply to many images that *don't* have any faces to detect; upwards of 40% (Viola & Jones, 2001).

The key feature of the Viola-Jones object detection framework is the Classifier Cascade (Cho, Mirzae, Oberg, & Kastner, 2009). Further performance improvements are realized by using an attentional cascade algorithm which orders many classifiers by increasing complexity. After the first wave of facial feature detections, the possible faces are handed to the next set of Haar feature classifiers. Windows from the picture that are determined not to contain a face are discarded, and those that may contain a face are passed to the next stage of the pipeline. This allows the framework to operate in real time via early rejection of non-objects and the focusing of more complex classifiers on only relevant data. Each classifier is more sophisticated than the last, but has to operate on less and less of the data. The number of stages required in the pipeline is determined by the required false positive and false negative ratings. Thirty or so stages are sufficient to reduce false positives to less than 1% of images while maintaining

Figure 1. The sum pixel intensity of the shaded rectangle can be computed with only 4 stored values

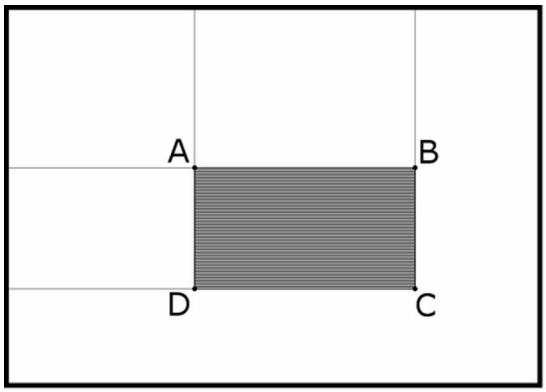
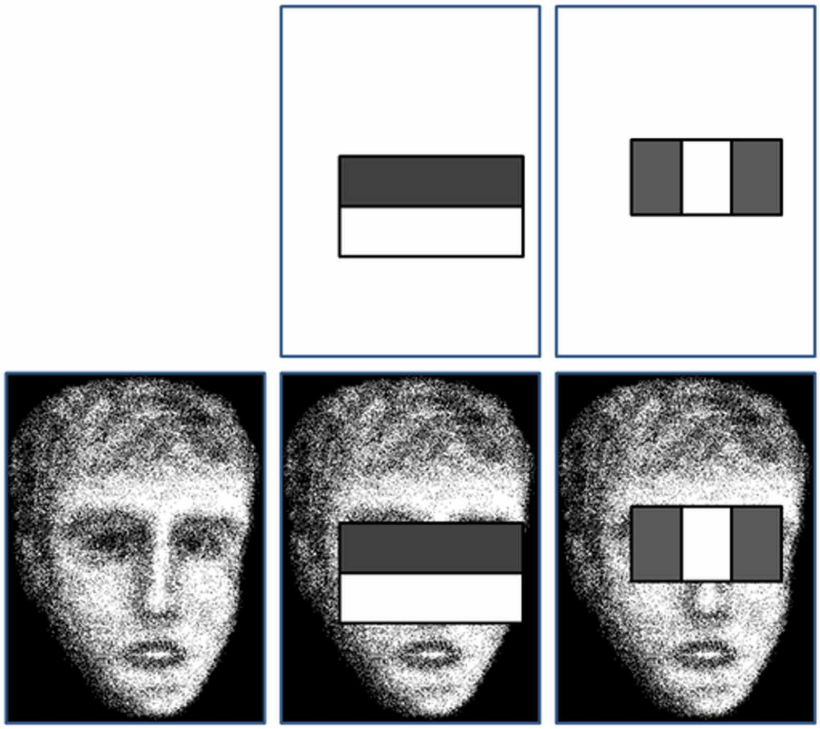


Figure 2. Two Haar features that recognize the eyes and the nose, respectively



around a 99% detection rating (Soyata T., Muraleedharan, Funai, Kwon, & Heinzelman, 2012). Upon completion, the detection function forms an indexed list of rectangles containing each face in a given source frame and returns the number of faces detected in the frame (if there are any).

PROJECTION (PJ)

Complementing the face detection algorithm is the eigenface recognition algorithm. It is an appearance based face recognition method, and was first developed in 1987 by Lawrence Sirovich and Michael Kirby and expanded upon by Matthew Turk and Alex Pentland (Turk & Pentland, 1991). The eigenface method works by reducing a set of database images into a set of basis images. These images (known as eigen-images or eigen-faces) can be combined linearly in order to recreate an image of a human face. New images can be projected into the same eigen-space. It is this which allows faces to be recognized by this method.

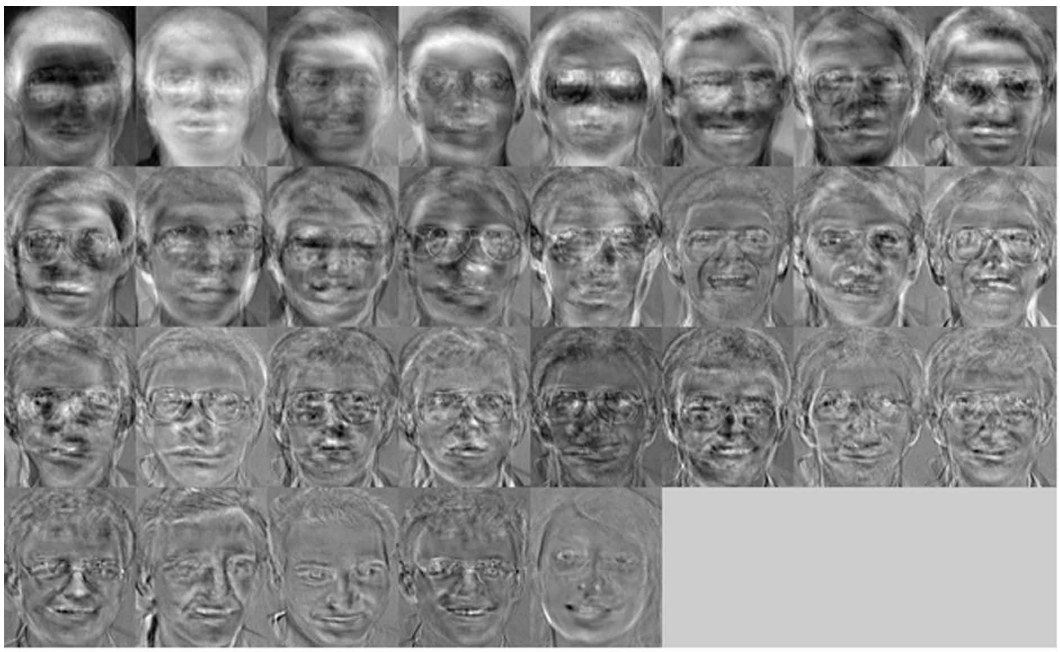
There is a large amount of information in a picture of a human face, even after it has been cropped by a face detection algorithm. Much of the information encoded in the picture is useless or redundant for recognizing a face. To reduce an image down to what is required to recognize the face, the eigenface recognition model is used. The database of training images of people that the system is meant to know are turned into eigenfaces, as are any test images the user wants recognized. The operation that creates these eigenfaces is Principle Component Analysis (PCA) (Kim, Jung, & Kim, 2002), and is referred to as Projection.

PCA is a statistical operation that converts a conventional dataset into a dataset of variables that are linearly uncorrelated (Pearson, 1901). When used to create a low dimensional representation of a set of faces, the basis vectors formed are called eigenfaces. After PCA is done on the training images, the resulting eigenfaces can be linearly combined to recreate any of the original images, and (theoretically) any other human face. A face can therefore be represented by an array of values (which are eigenvalues) that represent the weight each eigenface holds when combining them into the face. For example, a given set of faces might be re-represented using the near-orthogonal Eigenfaces as follows:

Face1: $\left(0.25 * Eigenface1\right) + \left(0.12 * Eigenface2\right) - \left(0.15 * Eigenface3\right) + \ldots$
Face2: $\left(0.18 * Eigenface1\right) + \left(0.22 * Eigenface2\right) - \left(0.45 * Eigenface3\right) + \ldots$
Face3: $\left(0.36 * Eigenface1\right) + \left(0.13 * Eigenface2\right) - \left(0.35 * Eigenface3\right) + \ldots$

Figure 3 depicts a set of 29 eigenfaces that were computed to re-represent an initial set of 500 frontal-face images (Soyata T., Muraleedharan, Funai, Kwon, & Heinzelman, 2012). At the maximum, there are a number of eigenfaces equal to the number of training images. For even a small 100 x 100 image, there are far too many eigenfaces for most computers to hold in their memory, let alone utilize them in real time. Fortunately, a small portion of the eigenfaces can represent enough of the set of sample images to provide accurate face recognition. Most of the eigenpictures can be discarded without a large loss of precision (Sirovich & Kirby, 1987). Each eigenfaces has an eigenvalue associated with it that represents how much the eigenface differs from the *average* face from the training set. A small eigenvalue indicates

Figure 3. A set of 29 Eigenfaces to represent a database of 500 images
Reprinted with the permission of the authors in (Soyata T., Muraleedharan, Funai, Kwon, & Heinzelman, 2012).



that eigenface represents less significant details or smaller changes. The percentage of information covered by the first k eigenfaces $P(k)$ can be calculated with the following equation:

$$P\left(k\right) = \frac{\sum_{i=1}^{k}\left(\sum_{j=1}^{n}\Omega_{i,k}\right)^2}{\sum_{i=1}^{n}\left(\sum_{j=1}^{n}\Omega_{i,k}\right)^2}$$

where $\Omega_{i:j}$ is the i^{th} component (eigenvalue) of the eigenvector for the j^{th} image in the database and n is the total number of images in the database. The more vectors retained, the more precise the algorithm can be. One consequence of the equation is that the smaller the databases size, the larger the required ratio of eigenfaces to training images to retain the same portion of information. This is counteracted by the fact that less information is required to accurately recognize faces from a smaller set (as fewer faces will have fewer people with similar facial features). Generated along with the eigenface database is a mean-image face, which is a pixel average of every image in the training set.

The actual projection step occurs during program runtime, after the eigenface database has be precompiled. Detected faces do not necessarily occupy a constant number of pixels in the source frame. They also may be subject to lighting conditions and irregular coloring. Therefore, each face is pre-processed which first involves resizing of the face to that of the training images in the database. This is a requirement in the eigenfaces method which preserves a consistent dimensionality for every calculation. Eigenvectors of different dimensions cannot be added together or compared. Next, the image is converted to grayscale and histogram equalization is applied which enhances the contrast of the image and maximizes the pronouncement of discernible features. The test face is subtracted by the mean and reshaped into a single row. Subtracting the mean allows a comparison done on the aspects that most differentiate each face from one another.

The image is run through the same Principle Component Analysis algorithm used train the database, projecting the new face it into the same space as the database of eigenfaces. The new image can be formed from a weighted average of eigenvectors from the eigenface database. These weights are the values used in the search portion of the algorithm to determine which person from the training images is the closest match.

SEARCH (S)

The search algorithm is quite simple in comparison to the previous operations. At this point in the overall recognition process, the test face is represented by an array of values. Each of these values is an eigenvalue paired with one of the eigenfaces from the database. Each eigenface is a different dimension in the eigenspace representing possible human faces. The eigenfaces can be linearly combined with weights corresponding to the person's eigenvalues in order to recreate their face. The vast amount of data required to describe a human face is reduced to a small array of values.

This allows two different faces to be compared is a simple and objective way. Since each face is a single point in a many dimensional space, the degree of similarity between any two faces can be determined

quickly by finding the distance between these two point. The standard method for finding the distance between points in an arbitrarily many dimensional space is the Euclidean distance metric.

The distance $d_E(k)$ between the test face projection and that of the k^{th} face in the database is shown below,

$$d_E(k) = \sqrt{\sum_{i=1}^{n} \left(\Omega_{i,testface} - \Omega_{i,k}\right)^2}$$

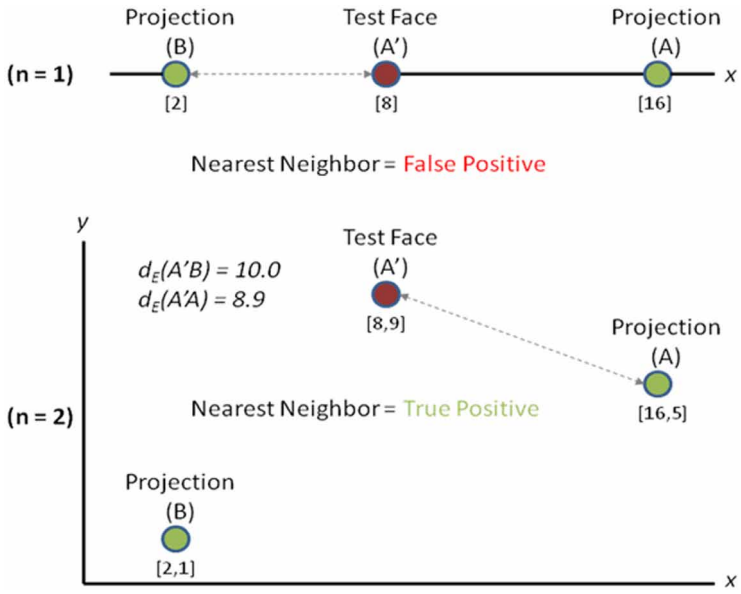
where Ω^i is the i^{th} component of the projection vector for the test face and the k^{th} face in the database and n is the number of components (dimensions) defining the eigenspace. The face residing the shortest distance away is matched to a name by using a reference index corresponding to name data contained in a separate file. It is possible that the test face does not match any of the faces inside the database. To account for this possibility, a threshold is set. Any match whose distance is greater than this limit is rejected, and the test face is considered to be a new person. Additionally, a second threshold can be used. If this distance is surpassed, the test face is considered to be not a face at all. Of course, images that are not faces should not make it to projection or search, but a simple double check doesn't negatively affect the face recognition application in any way.

If neither type of threshold is needed (such as for a use case where only faces in the database are tested), then the squared Euclidean distance can be used instead of the normal Euclidean metric. The squared Euclidean distance simply does not apply the square root operation. The relative distances can still be compared and the shortest one can still be found, so the exact distances are not needed. This optimization is mostly unnecessary, as the computational complexity of a single square root operation is insignificant in comparison to the rest of the face detection and recognition operations.

One implication of the eigenface search routine is that a database with m images, and an eigenspace of n dimensions requires $(m \times n)$ multiplications (square operations) in order to search the entire database for the nearest neighbor. Consequently, searching times show a non-linear relationship to database size. For eigenspaces formed with a dimensionality equal to the image number, the relationship is essentially quadratic, implying search times will increase dramatically as the database set grows large.

A rejoinder of this concern is that a higher dimension eigenspace will provide a greater degree of precision when defining eigenfaces. If more components are used to describe a unique face, there is a reduced chance of a test face residing in a neighborhood of eigenfaces in which a false positive is possible. This is demonstrated as simply as possible in the figure below, where projections are first described in a one-dimensional space. The projection of Subject A' (the test face) is found to be nearest to the projection of Subject B, which would result in a false prediction. By introducing another component and describing projections in a two-dimensional space, the contribution of the component in the y-direction results in a more accurate prediction. It follows, therefore, that a higher dimensionality will reduce the potential for erroneous results. This comes at costs impacting computation time and storage requirements for the database proper, thus considerations must be made during the PCA in order to balance performance with accuracy.

Figure 4. One and two dimensional representations of Euclidean distance



IMPLEMENTATION

When put together, the FD, PJ, and S algorithms can create a functional facial recognition program capable of detecting and identifying people in real time on regular consumer grade hardware. The entire process is computation and memory intensive, and the performance is heavily dependent on a number of factors. Most of the intensive work is done by the GPU. Both Face Detection and Projection utilize CUDA (Nvidia CUDA) to accelerate the maximum theoretical frame rate of the face recognition program. In this section we detail the computational and memory requirements and the inner-workings of the complete program.

Face Detection Algorithm Implementation

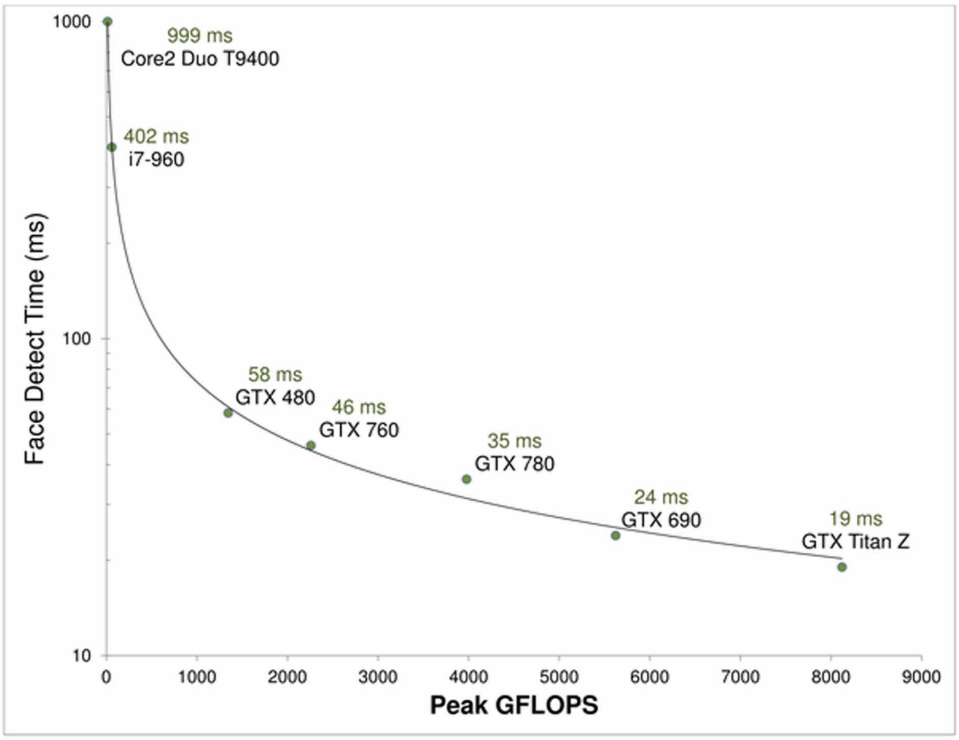
The Viola-Jones object detection framework is a well-studied algorithm. It is considered capable of real time detection even running on hardware from 2001. While even weak CPUs are capable of detecting faces in under a second, modern GPUs push computation times into the tens of milliseconds range. Detection time decreases as a power of the floating point operations per second of the processor. At a image resolution of 800x480, we observed that the average relation is as follows:

$$T_{FD} = 5172.7 * GFLOPS^{-0.622726}$$

where T_{FD} is the detection time in milliseconds. Figure 5 depicts the trend with example processors from low end CPUs to high-end consumer GPUs.

The computational complexity of the FD operation is dependant almost entirely on a single variable: the resolution of the image. The cascade classifier method looks at each sub-window of the image, and

Figure 5. Face detection time on different CPU and GPU architectures



determines if it is a candidate for containing a face. Sub-windows that are candidates are passed to the next stage of the pipeline. The nature of this method means that most of the analysis goes into the first stages of the pipeline and that most of the computational power of the early stages of the pipeline is used for analyzing sections of the image that don't contain faces.

Table 1 tabulates the number of operations needed for the first stage of the classifier cascade. Included are separate operations of Multiplication (MUL), Addition (ADD), and Multiply-Add (MAD) operations both for integer and floating-point data types. These values scale linearly with the total pixel count of the image. As such, the detection time scales linearly with the number of pixels. Combining this with the Detection Time vs. GFLOPS equation on the previous page gives the following equation:

$$T_{FD} = n_{pixels} * 0.01347 * GFLOPS^{-0.622726}$$

Table 1. Number of million Integer and Floating Point Operations (MIOP and MFLOP) to perform face detection on an 800x480 image

Operations	Total	ADD	MUL	MAD
MIOP Count	508	286	40	183
MFLOP Count	75.8	12.1	12.5	51.2

where T_{FD} is the detection time, n_{pixels} is the pixel count of the images, and GFLOPS is the peak number of floating point operations per second of the processor.

Somewhat counter-intuitively, the number of faces in an image has a very small effect on the total detection time as shown in Figure 6. This is owed to the high false positive rate of each stage of the cascaded classifier. Sub-windows that do contain a face must go through every stage of the pipeline, which does introduce waste computation time. Still, the computational requirements of false positives and areas rejected at the first stage are much larger, meaning the time difference between detecting one face and sixty faces is less than 6 milliseconds for a 800x480 image, when executed on a GTX760 GPU.

Eigenvector Projection Algorithm Implementation

Projection has shown to be the most computation centric component in the Face Recognition application. Both the creation of the eigenface database and performing principle component analysis during runtime require a significant amount of computational power. The time required to compile the eigenfaces for the database grows exponentially with the number of pictures in the database. As the compile time is directly dependent on the number of operations that must be performed, a more powerful processor reduces the time in a linear fashion.

Table 2 illustrates a significant limitation to the eigenface method. The time required to form the initial database set quickly grows to be unmanageable. The computational complexity of additional faces outpaces the reduction in compute time from using a processor with more peak GFLOPS. Even

Figure 6. Face detection time as a function of the number of faces in an image; the result is almost independent of the number of faces due to the high false positive rate of the cascaded classifiers.

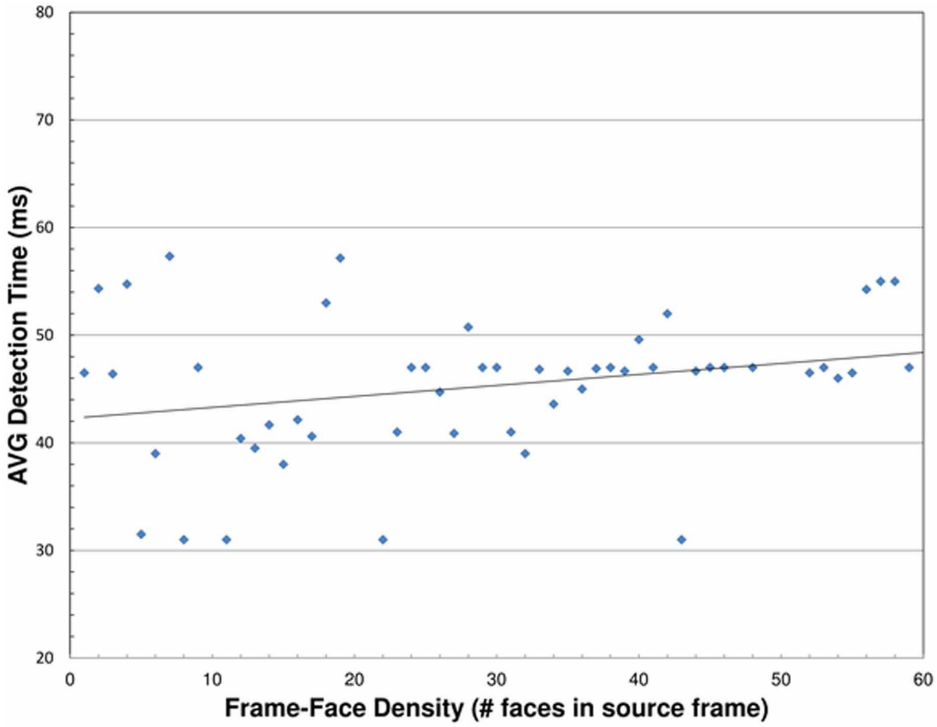


Table 2. Computational complexity of building the Eigenface database, along with the time required to build on a GX760 GPU

Number of Images in Database	Floating Point Operations Required (Billions)	Computation Time	Eigenface Database Size (GB)
1,000	90,700	8.5 Minutes	1.1
2,000	616,000	57 Minutes	2.2
3,000	5,530,000	8.7 Hours	3.4
4,000	31,300,000	49 Hours	4.5
5,000	116,000,000	182 Hours	5.6
10,000	5,240,000,000	342 Days	11.4
50,000	6,230,000,000,000	1,115 Years	57.9

using the best supercomputers of today, building a database from 50,000 images takes *weeks*. This limits its potential applications to some degree. Additionally, application run-time requires the entirety of the database to be loaded into the local RAM of the platform running the application. This will somewhat restrict the feasible size of the database. Most modern consumer grade computers contain 4 or 8 GB of RAM today (VALVe, 2014).

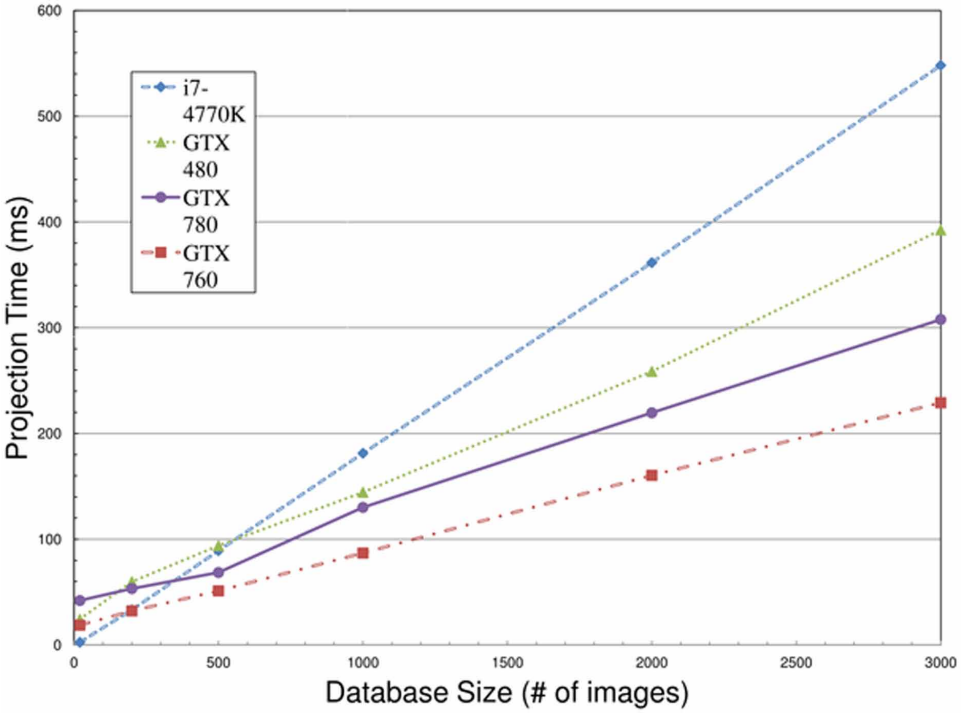
While prepping the database does require a large commitment of resources, it only needs to be done once. After that, performing PCA during runtime is the most intensive portion of the face recognition algorithm. The primary component of PCA is matrix multiplication. The standard for implementing matrix multiplication is to use Basic Linear Algebra Subprograms (BLAS), a set of low-level operations established in 1979, and still used today. A modern implementation on a GPU is to use the CUDA level-3 BLAS function GEMM (General Matrix Multiply) (Dongarra, Du Croz, Hammarling, & Duff, 1990). There are two Nvidia libraries that help develop face recognition code significantly: CUBLAS (Nvidia CUBLAS) for BLAS acceleration including GEMM, and CUFFT (Nvidia CUFFT) for acceleration FFT-b ased operations. Each call to GEMM involves uploading the test face ($8 * n_{pixels}$ bytes in size and the eigenvector array ($8 * n_{pixels} * n_{components}$ bytes) to device (i.e., GPU) memory. A multi-threaded kernel is then launched which computes the multiplication. Kernel execution of the matrix multiplication consists of billions of both integer and double-precision floating-point operations. Table 3 shows the number of operations done for a single invocation of GEMM on a 5000 image database (with one eigenvector per image, and 180x180 images).

A number of factors affect the projection speed. The obvious one is the power of processor used to perform PCA. The factor that is perhaps the most important is the number of images used to create the database. These two factors can be seen in Figure 7.

Table 3. Integer and floating point operation counts for a single GEMM (Generalized Matrix-matrix operation) function call

Integer	Total	ADD	MUL	MAD	SCADD	dFMA
MIOP Count	2457	701	375	588	671	-
MFLOP Count	-	-	-	-	-	5200

Figure 7. Projection time as a function of the number of training images in the database



Since the number of eigenvectors is proportional to the number of images in the training database (equivalent in our case), it clearly follows that the number of operations (FLOPs) necessary to carry out the multiplication will have a linear relationship to the database size. The number of components (which explicitly expresses the dimensionality of the eigenspace) can be selected to balance performance with the representational accuracy of eigenface vectors. The relationship between the number of eigenvectors used and the projection speed is therefore linear. Also visible in Figure 7 is the dependence of projection speed on processing power. Performing projection with a GPU approach in lieu of CPU results in a reduction of processing time in excess of 50% with some devices. The peak GFLOPS of the processor, in most cases, is related to the projection time by the following equation:

$$T_{PJ} = 1000 * \left(\frac{0.16004 * n_{db}}{GFLOPS_{device}} \right)$$

where T_{PJ} is projection time in milliseconds, and n_{db} is the number of images in the database. This equation assumes that the application is using 200x200 pixel images, and one eigenface is formed per image in the database. Modifying these two variables affects the accuracy of the face recognition application to some degree. Larger images and more eigenvectors both cause the application to be more accurate, with diminishing returns. 200x200 images and an eigenvector per face already capture enough information for an accurate face recognition application. The performance effects of these operations are quite straightforward. The projection time is linearly related to both the pixel count and the number of eigen-

faces used. The main operation of PCA is simply matrix multiplication of an $1 \times p$ array by a $p \times n$ matrix, where *p* is the pixel count of the database images and *n* is the number of eigenfaces. Increasing either parameter increases the total number of calculations performed proportionally. A projection time equation including these variables looks like this:

$$T_{PJ} = \left(\frac{0.004001 * p_{db} * n_{db} * r_{eigenfaces}}{GFLOPS_{device}} \right)$$

where T_{PJ} is projection time in milliseconds, p_{db} is the pixel count of an image in the database, n_{db} is the number of images in the database, and $r_{eigenfaces}$ is the ratio of eigenfaces to database images.

There are two cases where this equation is not quite accurate. With a database size of less than around 500 images, memory transfer speeds become dominant. While all processors perform PJ in under 100 ms in this case, it is difficult to predict the exact time required. CPUs are generally faster in this range, as they don't have the same memory transfer overhead that GPUs do. Another inconsistency in this formula is related to the omission of the execution patterns of a thread block in a GPU. For example, while the GTX780 is listed to be a higher-performance GPU as compared to the GTX760, its visible performance is worse according to Figure 7. This is due to the kernel call that is used by the Open CV library (Bradski & Kaehler, 2008) to execute the GPU code. When improperly used, these inconsistencies arise in GPU hardware, since GPUs leave a significant portion of the runtime execution responsibility to the programmer, whereas CPU architecture has advanced over the past four decades to push almost all of the runtime execution responsibility to the CPU hardware.

Indeed, the consistency of the process data (test face and eigenvectors) has a direct impact on the number of computations that must be performed, as well as the amount of data that must be transferred across memory pipelines. The poor relative performance of the GTX 780 in Figure 7 as compared to GTX 760 is due to the suboptimum performance of certain library APIs. Specifically, the GPU occupancy is very low during kernel calls (approximately 33%), starving the GTX 780 threads of useful work, thereby lowering its performance. These APIs will be optimized in the future, but for now may be a limitation to some face recognition systems.

There are a few more concerns related to the performance of PJ. The above data comes from a process that is entirely sequential; only one test face is uploaded, projected and downloaded at a time on a single synchronous CUDA stream. This approach severely limits the performance potential of the application. A real world face recognition application could benefit from improvements that may be found by invoking kernels into different streams, allowing for concurrent processing of multiple test faces. The use of Kepler-based GPUs in most professional computing clusters most certainly presents this possibility, as they are increasingly designed with concurrency in mind (Nvidia Corporation, 2014).

Database Search Algorithm Implementation

The final step in the recognition process is to search the database formed during the PCA and return information corresponding to the entry with the highest degree of similarity to the face under test. The database consists of projections for each image formed during the PCA. The database is first stored in a file on local media (hard drive). The values contained therein are loaded into referenced objects in host memory (RAM) during the application run-time. Each projection in the database expresses a point in

the multidimensional eigenspace corresponding to each training image and is considered unique. The image with the greatest likeness to the test face is determined using the Euclidean distance between the point expressed by the test face projection and the points expressed by the database projections.

In most instances, the time required to perform a search is much less significant than the time required to do face detection and projection.. The only operation done in the search stage of a face recognition application is a least Euclidean distance comparison between two arrays of data. The process time of the routine increases approximately linearly with the number of images in the database. This is simply because an additional comparison must be made for each additional image. Search time also increases fairly linearly with the number of components (dimensions) of the eigenspace. As the number of dimensions of the eigenspace is equal to the number of eigenfaces used to represent the database (which is itself dependant on the number of images in the database), the process time in effect scales quadratically with the size of the of the database.

$$T_{search} = 1000 * \left(\frac{3.575 * 10^{-7} * n_{db}^{2} + 1.018 * 10^{-4} * n_{db} + 9.773 * 10^{-3}}{GFLOPS_{device}} \right)$$

where T_{search} is search time in milliseconds, and n_{db} is the number of images in the database. The device in question is a CPU, rather than a GPU, and generally has a much lower floating point performance. The resolution of the images forming the database is not relevant to the search operation. The search algorithm compares the weights which describe how much of each eigenface a person's face is made from. The actual eigenfaces are not analyzed during this step.

While search time does scale faster with database size than projection time does, it is initially much smaller. Below a thousand images, the search time is almost entirely negligible. It doesn't become a particularly demanding process until around ten thousand images or so. That is reaching the limits of the maximum practical size of an eigenvector face recognition anyway. Search doesn't overtake projection until the database is over 30,000 images in size, out of the practical ranges of the recognition algorithm.

The search portion of the eigenface algorithm can be implemented on a CPU without worry of bottlenecking performance. This may not hold true in the future; as GPUs have matured faster than CPUs and may continue to do so, they may reach a point where they can project face data faster than a CPU can perform a search on those projections.

Table 4. Projection and search times on a workstation with a GTX760 GPU and an Intel i7-4770K CPU

Database Size	Projection Time (ms)	Search Time (ms)
1,000	70.88	2.647
2,000	141.8	9.272
3,000	212.6	19.93
4,000	283.5	34.62
5,000	354.4	53.35
10,000	708.8	207.5
20,000	1418	818.3
50,000	3544	5071

CLOUD-BASED PERFORMANCE AUGMENTATION AND OTHER APPLICATIONS

Performing Face (or object) Recognition over the internet using cloud resources is an interesting application (Keller, 2011) (Soyata T., Muraleedharan, Funai, Kwon, & Heinzelman, 2012) (Soyata T., et al., 2012). Due to the incredible amount of computation required for this operation, it is natural to access the cloud resources (Kwon, et al., 2014) (Soyata, Ba, Heinzelman, Kwon, & Shi, 2013) (Wang, Liu, & Soyata, 2014). There is typically a 10,000x to 1,000,000x disparity between the computational capabilities of mobile devices and the cloud (Soyata, Ba, Heinzelman, Kwon, & Shi, 2013), cloud resources can be accessed to take advantage of specialized accelerator hardware (Guo, Ipek, & Soyata, 2010) (Kirk & Hwu, 2010) (Li, Ding, Hu, & Soyata, 2014) (Soyata, Friedman, & Mulligan, 1997) (Soyata & Liobe, 2012). Additionally, the path from the mobile phones to potentially multiple different cloud servers can be modeled as a graph (Soyata & Friedman, 1994) (Soyata, Friedman, & Mulligan, 1993) (Soyata, Friedman, & Mulligan, 1995), where edges represent the links between the mobile phone and the cloud servers, or, between cloud servers. This allows combinatorial optimization algorithms to optimally reroute (i.e., retime) the data to achieve optimum overall algorithmic performance (Soyata & Friedman, 1994) (Soyata T., 2000).

Open CV based object detection (Bradski & Kaehler, 2008) is not only used in face recognition, but also many field systems, where animals in the field are detected using supercapacitor-powered field systems (Fahad, et al., 2012) (Hassanalieragh, Soyata, Nadeau, & Sharma, 2014) (Nadeau, Sharma, & Soyata, 2014). Additionally, in tele-health monitoring, Open CV can be used to monitor a patient remotely (Kocabas & Soyata, 2014) (Kocabas, et al., 2013) (Page, Kocabas, Soyata, Aktas, & Couderc). In general, Open CV is one of the most commonly used programs, and it is under continuous development, with new versions released on an ongoing basis. GPU-acceleration just became available for Open CV, albeit with some implementation inefficiencies as we mentioned previously. The version that is being used should be checked for compatibility for the GPU compute capability. In general, a conservative approach should be taken, as we did. We opted to use a little less up to date version of Open CV, since the most recent version had issues with CUDA 6.0.

CONCLUSION AND FUTURE WORK

While once considered a pipe dream and placed in the same difficulty category as artificial intelligence, computer vision and face recognition have made great strides since the early days of manually marking up paper cards. Both face detection and face recognition have been implemented in real time with creative algorithms. The industry standards (Viola-Jones Object Detection Framework and Eigenface Recognition Method) are open source and freely available through libraries like OpenCV.

With the powerful graphics processing units produced in recent years, face recognition can now, for the first time, be implemented in real time on non specialized consumer grade hardware. High performance GPUs are able to detect and recognize many faces. Top of the line and specialized processors are able to deal with frames as fast as regular cameras can spit them out. It is not a stretch to think that within a few years, large crowds of people could be scanned in real time with a high resolution camera.

As it stands, face recognition is not perfect. The fastest algorithms have significant restrictions. Non-uniform lighting significantly reduces the accuracy. The two algorithms discussed in this chapter require the face to be facing almost directly at the camera, ideally without any rotation about any axis.

Things like classes and certain hairstyles can obscure important details. Some of these flaws are already solved by systems that do not operate in real time. New modifications and optimizations are constantly in development in the field of computer vision.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation grant CNS-1239423 and a gift from Nvidia corporation.

REFERENCES

Bradski, G., & Kaehler, A. (2008). *OpenCV Computer Vision with OpenCV Library*. O'Reilly.

Cho, J., Mirzaei, S., Oberg, J., & Kastner, R. (2009). FPGA-based face detection system using Haar Classifiers. *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (pp. 103-112). doi:10.1145/1508128.1508144

Crow, F. (1984). Summed-area tables for texture mapping. *Computer Graphics*, *18*(3), 207–212. doi:10.1145/964965.808600

Dongarra, J., Du Croz, J., Hammarling, S., & Duff, I. (1990). A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, *16*(1), 1–17. doi:10.1145/77626.79170

Fahad, A., Soyata, T., Wang, T., Sharma, G., Heinzelman, W., & Shen, K. (2012). SOLARCAP: Super Capacitor Buffering of Solar Energy for Self-Sustainable Field Systems. *Proceedings of the 25th IEEE International System-on-Chip Conference (IEEE SOCC)* (pp. 236-241). Niagara Falls, NY. doi:10.1109/SOCC.2012.6398354

FBI. (2011). Biometric Center of Excellence. Retrieved from https://www.fbi.gov/about-us/cjis/fingerprints_biometrics/biometric-center-of-excellence

Freund, Y., Schapire, R., & Abe, N. (1999). A Short Introduction to Boosting. *Journal-Japanese Society for Artificial Intelligence*, *14*, 771–780.

Goldstein, A.J., Harmon, L.D., & Lesk, A.B. (1987). Identification of Human Faces. Proceedings of the IEEE, 59(5), 748-760.

Guo, X., Ipek, E., & Soyata, T. (2010). Resistive Computation: Avoiding the Power Wall with Low-Leakage, {STT-MRAM} Based Computing. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Saint-Malo, France (pp. 371-382). doi:10.1145/1815961.1816012

Hassanalieragh, M., Soyata, T., Nadeau, A., & Sharma, G. (2014). Solar-Supercapacitor Harvesting System Design for Energy-Aware Applications. *Proceedings of the 27th IEEE International System-on-Chip Conference (IEEE SOCC).* Las Vegas, NV. doi:10.1109/SOCC.2014.6948941

Keller, J. (2011). Cloud-Powered Facial Recognition is Terrifying. Retrieved from http://www.theatlantic.com/technology/archive/2011/09/cloud-powered-facial-recognition-is-terrifying/245867/

Kim, K., Jung, K., & Kim, H. (2002). Face recognition using kernel principal component analysis. *IEEE Signal Processing Letters*, 9(2), 40–42.

Kirk, D., & Hwu, W. (2010). *Programming Massively Parallel Processors*. Morgan Kaufmann.

Kocabas, O., & Soyata, T. (2014). Medical Data Analytics in the cloud using Homomorphic Encryption. In P. R. Deka (Ed.), *Handbook of Research on Cloud Infrastructures for Big Data Analytics* (pp. 471–488). Hershey, PA, USA: IGI Global. doi:10.4018/978-1-4666-5864-6.ch019

Kocabas, O., Soyata, T., Couderc, J.-P., Aktas, M., Xia, J., & Huang, M. (2013). Assessment of Cloud-based Health Monitoring using Homomorphic Encryption. *Proceedings of the 31st IEEE International Conference on Computer Design (ICCD)*, Ashville, VA, USA (pp. 443-446). doi:10.1109/ICCD.2013.6657078

Kwon, M., Dou, Z., Heinzelman, W., Soyata, T., Ba, H., & Shi, J. (2014). Use of Network Latency Profiling and Redundancy for Cloud Server Selection. *Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014)*, Alaska (pp. 826-832). doi:10.1109/CLOUD.2014.114

Li, P., Ding, C., Hu, X., & Soyata, T. (2014). LDetector: A Low Overhead Race Detector for GPU Programs. *Proceedings of the 5th Workshop on Determinism and Correctness in Parallel Programming (WODET2014)*.

Lindeberg, T. (1991). *Discrete scale-space theory and the scale-space primal sketch* [Doctoral dissertation] Royal Institute of Technology.

Nadeau, A., Sharma, G., & Soyata, T. (2014). State-of-charge Estimation for Supercapacitors: A Kalman Filtering Formulation. *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2013)*, Florence, Italy (pp. 2213-2217). doi:10.1109/ICASSP.2014.6853988

Nvidia CUBLAS. (n. d.). Retrieved from https://developer.nvidia.com/cublas

Nvidia CUDA. (n. d.). *NVIDIA*. Retrieved from http://www.nvidia.com/object/cuda_home_new.html

Nvidia, C. U. F. F. T. (n.d.). Retrieved from https://developer.nvidia.com/cufft

Nvidia Corporation. (2014, August 1). *Kepler Tuning Guide*. Retrieved from http://docs.nvidia.com/cuda/kepler-tuning-guide/#axzz3BocBTboS

Page, A., Kocabas, O., Soyata, T., Aktas, M., & Couderc, J.-P. (2015). Cloud-Based Privacy-Preserving Remote ECG Monitoring and Surveillance. *Annals of Noninvasive Electrocardiology, 20(4), 328-337*. PMID:25510621

Papageorgiou, C., Oren, M., & Poggio, T. (1998). A general framework for object detection. *Proceedings of the sixth international conference on Computer vision* (pp. 555-562). IEEE.

Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, *2*(11), 559–572. doi:10.1080/14786440109462720

Sirovich, L., & Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America*, *4*(3), 519–524. doi:10.1364/JOSAA.4.000519 PMID:3572578

Soyata, T. (2000). *Incorporating Circuit Level Information into the Retiming Process*. University of Rochester.

Soyata, T., Ba, H., Heinzelman, W., Kwon, M., & Shi, J. (2013). Accelerating Mobile Cloud Computing: A Survey. In H. T. Kantarci (Ed.), *Communication Infrastructures for Cloud Computing* (pp. 175–197). Hershey, PA, USA: IGI Global. Doi:10.4018/978-1-4666-4522-6.ch008

Soyata, T., & Friedman, E. G. (1994). Retiming with Non-Zero Clock Skew, Variable Register and Interconnect Delay. *Proceedings of the IEEE Conference on Computer-Aided Design (ICCAD)* (pp. 234-241).

Soyata, T., & Friedman, E. G. (1994). Synchronous Performance and Reliability Improvements in Pipelined ASICs. *Proceedings of the IEEE ASIC Conference (ASIC)* (pp. 383-390). doi:10.1109/ASIC.1994.404536

Soyata, T., Friedman, E. G., & Mulligan, J. H. (1993). Integration of Clock Skew and Register Delays into a Retiming Algorithm. *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, Chicago, IL (pp. 1483-1486).

Soyata, T., Friedman, E. G., & Mulligan, J. H. (1995). Monotonicity constraints on path delays for efficient retiming with localized clock skew and variable register delay. *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, Seattle, WA (pp. 1748-1751). doi:10.1109/ISCAS.1995.523751

Soyata, T., Friedman, E. G., & Mulligan, J. H. (1997, January). Incorporating Interconnect, Register, and Clock Distribution Delays into the Retiming Process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *16*(1), 105–120. doi:10.1109/43.559335

Soyata, T., & Liobe, J. (2012). pbCAM: probabilistically-banked Content Addressable Memory. *Proceedings of the 25th IEEE International System-on-Chip Conference (IEEE SOCC)* (pp. 27-32). Niagara Falls, NY.

Soyata, T., Muraleedharan, R., Ames, S., Langdon, J., Funai, C., Kwon, M., & Heinzelman, W. (2012). COMBAT: Mobile Cloud-based cOmpute/coMmunications infrastructure for BATtlefield applications. *Proceedings of the Society for Photo-Instrumentation Engineers*, Baltimore, MD, USA (Vol. *8403*). doi:10.1117/12.919146

Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., & Heinzelman, W. (2012). Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. *Proceedings of the IEEE Symposium on Computers and Communications ISCC '12* (pp. 59-66).

Turk, M., & Pentland, A. (1991). Face recognition using eigenfaces. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1991. Proceedings CVPR*, *91*, 568–591.

VALVe. (2014, July). *Steam Hardware & Software Survey: July 2014*. Retrieved from http://store.steampowered.com/hwsurvey

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR '01* (pp. I-511-I-518).

Wang, H., Liu, W., & Soyata, T. (2014). Accessing Big Data in the Cloud Using Mobile Devices. In P. R. Dek (Ed.), *Handbook of Research on Cloud Infrastructures for Big Data Analytics* (pp. 444–470). Hershey, PA, USA: IGI Global. doi:10.4018/978-1-4666-5864-6.ch018

Yang, M., Kriegman, D., & Ahuja, N. (2002). Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(1), 34–58. doi:10.1109/34.982883

KEY TERMS AND DEFINITIONS

AdaBoost (Adaptive Boosting): A machine learning algorithm. Used by the Viola-Jones object detection framework to select the most important features and train the Face Detection program.

Classifier Cascade: A system for testing possibly detected faces with increasing stricter classifiers.

Compute-Unified Device Architecture (CUDA): The programming language of the GPUs introduced by Nvidia Corporation. This language includes almost full functionality of the C++ programming language, and has extensions to allow GPU kernel execution.

Eigenface: An eigenvector representing a portion of a human face. Its eigenvalue is the weight it holds when combining eigenfaces to create a single face.

Euclidean Distance: The distance between two nodes in an N-dimensional space. For example, in a single dimensional space, the Euclidean distance is identical to the linear distance between any two nodes. For a two dimensional vector, Euclidean distance is the square root of the sum of the distance-squares of each dimension.

Face Detection: A specific case of object detection, where the *object* is a human face. For this specific case, there are much better established algorithms, such as Viola-Jones, as compared to the generalized object detection algorithm. Face Detection is also the very first step of face recognition.

Face Recognition: The process of determining the identity of a face (i.e., recognizing it) by comparing it against a set of known faces. These set of known faces are typically stored in a database in a pre-processed format.

GEMM (General Matrix Multiply): One of the most common operations in any image processing algorithm. Since images can be represented as matrices, and the image processing algorithms almost always define the performed operations as matrix multiplications and additions, GEMM is at the heart of many image processing algorithms. Most GPU manufacturers, such as Nvidia Corp., provide GEMM subroutines for free with their GPUs to allow the efficient implementation of image processing algorithms.

Graphics Processing Unit (GPU): A massively parallel computational device, generally used as an accelerator to a CPU. While a CPU contains 2 to 6 cores, a GPU might contains 100's of 1000's of cores. However, each core has much lower operating frequencies, and much less control, making them simpler. The primary advantage of a GPU is that, each core contains a floating point unit, which makes GPUs perfect candidates for massively parallel computations that require floating point operations.

Haar-Like Feature: A feature composed of intensity sums that represent human facial components, such as cheeks or eyes.

Image Processing: The process of applying one of the existing algorithms to an input image. Since the pixels (picture elements) in an image can be operated on simultaneously by many image processing algorithms, these algorithms are generally massively parallel in nature, thereby benefitting significantly from GPUs for acceleration.

Integral Image: A system for representing sums of subsets of data in a matrix. It is used to perform face detection efficiently.

Object Detection: In the field of computer vision, object detection is the process of locating an object in an image. For example, in a military application scenario, a harmful object in the battlefield can be detected using one of the established object detection algorithms.

Principal Component Analysis (PCA): A statistical procedure to reduce the dimensionality of a set of data elements by re-representing them in terms of a different set of orthogonal components. For example, in the Projection phase of the face recognition, PCA can be used to represent a set of 500 images in terms of a set of 30 orthogonal face vectors, called eigenfaces.

Projection: The second phase of the face recognition algorithm, and first phase of an eigenface-based face recognition system. An image of a face is projected into the same eigenspace as the database it is being compared to. This allows significant reduction in the amount of data being operated on, thereby drastically speeding up the overall face recognition algorithm.

Search: The second portion of an eigenface-based face recognition system. The projected data is compared to the database and a specific name is associated with the face being recognized. This is done by computing the Euclidean distance of the source face's projection vectors and the ones in the database.