Technical Report CS14-11

# Approximate Algorithmic Image Matching to Reduce Online Storage Overhead of User Submitted Images

Braden D. Licastro

Submitted to the Faculty of
The Department of Computer Science

Project Director: Dr. Robert Roos
Second Reader: Dr. Gregory Kapfhammer

Allegheny College
2013

*I hereby recognize and pledge to fulfill my responsibilities as defined in the Honor Code, and to maintain the integrity of both myself and the college community as a whole.*

---

Braden D. Licastro

**BRADEN D. LICASTRO. Approximate Algorithmic Image Matching to Reduce Online Storage Overhead of User Submitted Images. (Under the direction of Dr. Robert Roos.)**

## ABSTRACT

Reducing the number of duplicate images uploaded to public servers is an ever more relevant problem as the number of images shared increases dramatically every day. Methods of data reduction such as file expiration dates only lessen this load by a small amount while common methods of image matching are in many cases resource exhaustive, time consuming, or highly inaccurate. This research aims to derive an algorithm capable of identifying near-duplicate images through file hashing, pixel difference, and histogram comparisons. In order to test the feasibility of implementing such an algorithm, a basic photo sharing website has been developed and tested on a fixed collection of images.

# Dedication

To Professor Cupper. He was more than just an advisor and professor; he was a member of the Alden family and a father away from home.

# Acknowledgements

I would like to thank my thesis advisor, Professor Roos for the time, expertise, and guidance he provided me as I worked through this project. I would also like to thank my family and girlfriend for their support and encouragement that kept me going until the end.

# Contents

# List of Figures

# Chapter 1

# Introduction

Sharing media with the public is becoming a more integral part of social interaction every day. Static images are just one of these many forms of media, and the number of daily uploads to image hosting websites is absolutely staggering. According to a recent survey by All Things Digital, as of May 2013, more than 500 million images are uploaded to image sharing websites each day, and this number is expected to double by the end of 2014 [4]. With figures this large, it immediately becomes apparent that multiple issues come with this trend of increased image sharing. Namely, how much space does this number of images required, and is there a technology available to reduce this requirement. The simple answer is yes, there are tried and tested technologies that will reduce storage costs, but before looking into these technologies, it is important to understand what an image hosting website is and how they function.

## 1.1 Image Hosting Services

Image hosting services, or image sharing websites are sites that allow users to upload images to the internet and share them publicly with the link they are provided. These image sharing websites mostly operate in the same fashion, but recently a new breed has emerged. As seen in Figure 1.1, both submission processes are similar, but both have inherent advantages and disadvantages.

Looking at the first submission process variant at the top of Figure 1.1, a user would like to share an image with the public. The user can upload this image through the internet from any internet connected device, and the image will be stored on a publicly accessible server. From here, any number of people can access this shared image through an internet connected device indefinitely. Nearly all image hosting services operate on this model, some of the more popular services include but are not limited to Flickr, Imgur, Photobucket, Shutterfly, and Instagram.

The second image submission variant shown at the bottom half of Figure 1.1 is currently only used by one host called Snapchat. With this system, multiple differences are immediately apparent. First, this service model will only accept images from mobile users. It can also be seen in the diagram that when a user uploads an image through the internet, it is no longer accessible long term from a server, but it

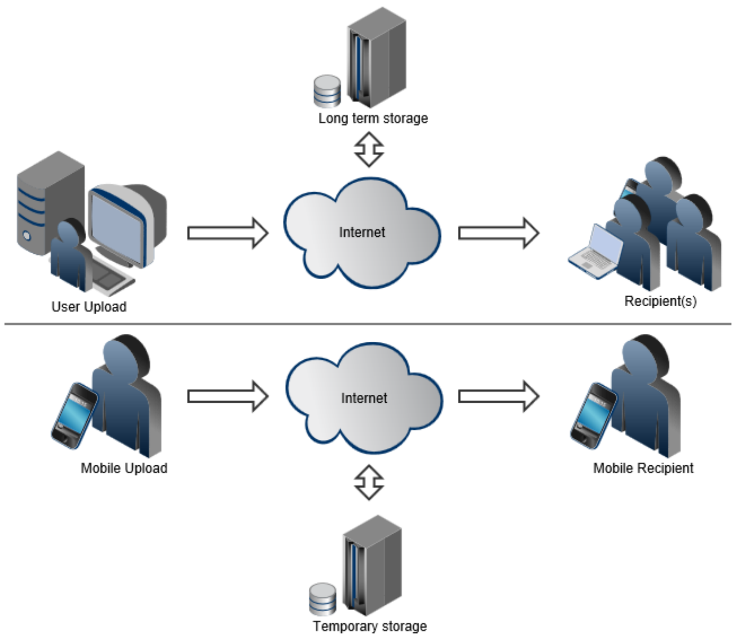


Figure 1.1: Various Image Hosting Service Structures

is in fact only temporarily available for a specified recipient to receive. This service actually allows the uploader to set an expiration time anywhere from 1 second to 10 seconds [9]. After the recipient opens the image and this time period has expired, the image is deleted permanently from the server and no longer accessible to either party.

Both systems have their own inherent benefits, but they also have detriments. Though they reach from the infrastructure needed to support the service all the way out to the end user, this research only focuses on the infrastructure function. In order to realize the application of the proposed research, it is important to understand why such a focused application is required. For the first image hosting variant in Figure 1.1, the long term storage of image files raises concern. By storing files for an extended period of time, it is a guarantee that duplicate data will eventually make its way to the storage servers. Determining duplicates and preventing their addition can save not only a considerable amount of space as the amount of redundant data increases, but it can also save a large amount of money when discussing the cost per gigabyte of data storage. This research will not target the application of the temporary system seen at the bottom of the figure as the amount of stored data is minimal due to the rapid turnover of data being housed.

## 1.2 Data Management Technologies

The image hosting services outlined in section 1.1 utilize numerous technologies to help lessen the load of the files they must store. Though official word and articles discussing the technologies these companies implement are nonexistent, after examining the services several different technologies became immediately apparent. The most prominent method of space reduction is the reduction in size of uploaded images. By reducing the quality of the image by a small percent, these websites are able to significantly reduce the amount of space needed to store the uploaded images. The next most common method of space reduction used is the expiration of uploads. After locating the earliest available images across the websites it was determined that the file expiration can possibly work in one of two ways. The implementation either functions as a countdown from the date of upload or in the other case the countdown begins after the last date the file was accessed. Each time the file is accessed, the timer will reset. Finally, some of these sites also limit the size of the uploaded files and the number of uploads per user. In order to remove these restrictions, many sites also offer paid services which assist with the cost of upkeep.

## 1.3 Motivation

Summarizing the information presented thus far, image hosting websites clearly require complex infrastructures, not only to allow the sharing of files but to manage the large numbers of submissions. Although the numerous technologies used to lessen the space requirements of the submissions work well, it should be possible to further improve on the system by reducing the number of duplicate submissions. According to a recent survey by All Things Digital, more than 500 million images are uploaded to long-term image sharing websites, and this number is expected to double by the end of 2014 [4]. In addition, a study published by NTP Software found that nearly 20% of stored data is duplicate [14]. These numbers are staggering, especially when there is a possibility of reducing an additional 100 million images.

To bring the possible savings into light a rough calculation based off of the 2013 average of $.05 per gigabyte and an assumed image size of 1MB will show that by removing the duplicates a significant amount of money could be saved. More specifically, approximately $18 million can be saved annually at the current sharing levels. By allowing unregulated user submitted data, it quickly becomes apparent that data redundancy reduction can save a significant amount of storage space and money.

## 1.4 Goals of the Project

The purpose of this project is to develop and test a system capable of identifying and reducing the number of duplicate image submissions on an image sharing website. In order to fulfill this goal, an image sharing website will be developed that is capable of accepting images as uploads and providing a link to the user for public access of the image. To develop a functional website it must be able to perform several functions. This website should accept an image file through an online submission form. At this point, it will process the image and match it against the collection of images currently stored on the server. This process will be completed using a series of algorithms and checks outlined in in section 3. The website will also be developed using PHP, the PHP GD image library, and HTML5 to ensure minimal chance of conflicts between scripts and languages. Using this website, the proposed duplicate image detection tool will be implemented using both original and existing code from other resources and tested thoroughly to assess the effectiveness of using this method of duplicate reduction.

## 1.5 Thesis Outline

The remainder of this thesis will discuss the work in further detail in addition to existing formation relating to the topic. First being Chapter 2, which discusses related works and existing research that has been completed on the topic of image matching and comparison. Chapter 3 will cover the project details pertaining to the website and supporting infrastructure it will operate on. This will include a brief discussion of the hardware and configuration of the web server in addition to the website design and implementation of the duplicate image detection tool that will be integrated and tested. Chapter 4 will then discuss the collected results and the accompanying evaluation of the collected data. This evaluation will include the performance costs of implementing this system over a passive one, which only acts to prevent a specific file from ever being submitted to the server. The evaluation will also include the results of the image de-duplication and a determination of whether such a system is a feasible method of reducing storage requirements. An additional section outlining threats to validity will also be included in this chapter. Finally, Chapter 5 will summarize the research and conclusions completed throughout this Senior Comprehensive Project and inslude possible areas of future work.

# Chapter 2

# Related Work

A typical second chapter deals with a survey of the literature related to the thesis topic. The subsections may be organized in whatever manner seems best suited to the material—chronological, or by topic, or according to some other criteria (e.g., primary versus secondary resources). The examples given in the sections in this chapter are nonsensical in content; they are provided merely to give examples of citing bibliographic references. Resources should be cited by author name(s), not by title. There should be a space between the square brackets of a citation and any preceding words. Thus, "Smith and Jones[17]" is wrong; "Smith and Jones [17]" is correct. If the citation is at the end of a sentence, the period goes after the brackets ("Johnson [23].", not "Johnson. [23]").

## 2.1 Primary Sources

The earliest work done in widget software is described in the seminal 1986 paper by Smith and Jones . Using a networked array of high-performance computers they demonstrate that widgets with $k$ degrees of freedom can be simulated in time proportional to $k \log^2 k$. At the heart of their demonstration is an algorithm for re-encabulating the widgets using a lookup table that can be updated in real time, assuming that the widgets are non-orthogonal. The question of simulating orthogonal widgets is left open, but the authors conjecture that orthogonality will add at most another factor of $k$ to the performance upper bound.

## 2.2 Recent Results

A number of papers deal with issues that are peripheral to the orthogonal case, but Dioşan and Oltean were the first to tackle it directly. In their 2009 paper , Dioşan and Oltean apply evolutionary techniques to the orthogonal widget case, obtaining empirical results that suggest an efficient algorithm might be at hand. Their approach is characterized by the use of a genetic algorithm to evolve other, more problem-specific evolutionary algorithms.

# Chapter 3

# Method of Approach

In order to create an effective image matching algorithm, it was necessary to pull research, knowledge, and pre-existing code from a number of resources in addition to completing original works. Although the research does not target the hardware infrastructure of the network that image sharing sites use, a simple public web server was built and configured. This topic will be briefly discussed in Section **??**o test the proposed image matching method, it was also necessary to build a skeleton that would act as a simple image sharing website. This skeleton will be capable of accepting an image file and returning a link to the user which will allow the submission to be viewed. The site will also allow for the development of the proposed matching function and will provide the necessary functions to accept and handle the output of the research.

## 3.1 Server Configuration

In order to implement the website, it is necessary to have an environment capable of running the required processes. To do this, several options were considered. The first option was to run the site on a locally installed Apache web server. After a small amount of testing, this was determined not to be the best option. Due to the lack of a dedicated machine the web server had wildly varying performance due to interference from other processes that could not be closed. This was even a problem when operating on a fixed amount of memory. In addition to this, the XAMPP environment that was tested frequently became momentarily non-responsive with resource hungry processes such as operations performed on large files.

The next alternative was to purchase web space on a shared server. These servers are readily available at minimal cost from dozens of providers such as A Small Orange, BlueHost, and Byethost. After careful consideration several concerns prevented the use of this method. The primary factor was that these servers are shared with numerous users. Due to the nature of this setup, it is unknown how many websites are being hosted by a particular server, and how many concurrent users are accessing these sites at any given time. In addition, it is not possible to set an allotted amount of memory for a particular environment or control what programs are running that may possibly impact the performance of the research.

Another option, that looked very promising at first was to rent server time from a provider such as Amazon Web Services. With this method it is possible to control not only what programs are operational, but it also allows for more freedom of configuration. This would seem like the most probable solution to the problem, but there are still factors that cannot be eliminated such as the speed of the internet connection. By hosting a local web server, it is possible to control this factor, but how much could it affect the results. To test this concern, a server was rented using the free tier services. From this point, a simple timer was configured and one 15 megabyte file was transferred using SFTP. This process was repeated 10 times and the results were analyzed. After reviewing the results, there was no concerning transfer speed variance making this a good match. In the end, it was discovered that there are restrictions to the service that do not allow an individual to alter settings relating to resource allocation, which was a key concern from the start.

The final option was to implement a local dedicated web server and operate the website and scripts from that. The machine proposed would allow not only very tight control over variables such as resource allocation, installed programs, and custom network hardware configuration, but it also allowed usage on a local network or over an internet connection. By running initial tests on a local area network, it is possible to eliminate internet speed fluctuations and control the number of devices utilizing the network bandwidth. This also opened another path where a real world simulation could be run by submitting images to the system over an internet connection and comparing the behavior with only one variable at a time differing.

To build the server a specification had to be determined that would allow optimal performance. To allow the greatest flexibility, the Ubuntu Server operating system was chosen. From this, the server's hardware specifications were chosen based off of the minimum system requirements given by the operating system, Apache suite, and MySQL Database. This information was used to pick a quantity of memory, hard disk space, and processor speed. The server was built with 4 Gigabytes of random access memory (RAM), 3 Gigabytes allocated to the programs and operating system, a 2.43 GHz (Gigahertz) Intel Core i3 processor, and dual 7200 RPM 500 Gigabyte hard drives. The integrated network card was faster than the available network equipment, so it was not of direct concern. The hard disks were chosen to provide ample space for any reasonable number of tests but not provide so much space as to be considered excessive for their purpose. A 7200 RPM variant was also chosen to allow the maximum data throughput and not become a bottleneck when working with great numbers of large image files. Finally, the disks were configured as a redundant mirror to emulate a simple backup system that duplicates the data as a form of backup. This will allow the collection of data and comparison of storage requirement improvements in a non redundant system, and one with a worst case scenario backup implementation that simply creates copies of the files.

The software selection is more straightforward. After selecting to use a Linux operating system, Ubuntu Server was chosen as the distribution. I had the most experience with using, configuring, and troubleshooting issues with this environment and decided it was best for this reason. The accompanying software was fairly simple to select as a quick Google search for "Ubuntu web server" will return thousands of results outlining the setup and configuration of a basic Linux-Apache-MySQL-PHP, or LAMP server. The setup process was completed step by step using the ApacheMySQLPHP LAMP Server Setup Guide provided through the Ubuntu Documentation [5]. Next, the code which will be discussed shortly requires that the PHP GD Image Library be installed. To prepare the PHP installation to use this library, the server was configured using the direction of the tutorial hosted by nixCraft [13].

Upon the completion of the prior configurations, the server was updated and running the latest version of all installed software. To prevent updates from altering the outcomes of the future, a hold was placed on all packages to prevent updates from being installed. In addition to this, the firewall was configured to allow HTTP communication through port 80, which allows interaction through an internet browser with the website hosted on the server. MySQL did not require any setup past the installation of the program and was left alone. At this point, the server configuration was complete and the default Apache "Success" page was displayed when accessing the server showing that everything was working properly.

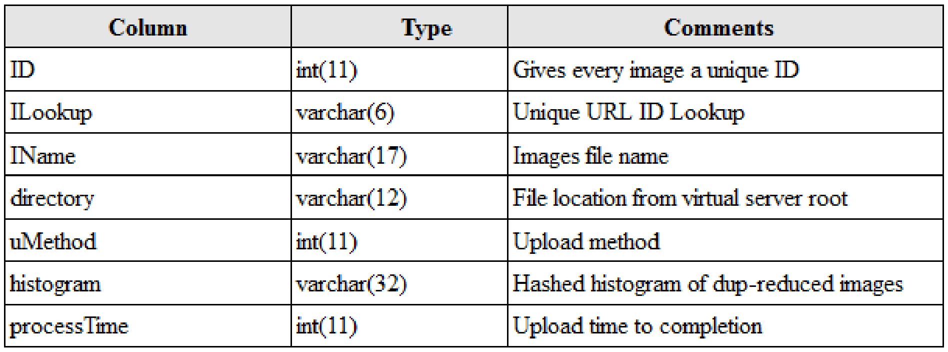
## 3.2 Website Design

The core of this research hinges on the successful implementation of an image comparison algorithm. In order to do this, a website needed to be developed that acted in a similar manner to an simple image sharing site. In order to do this, a specific demographic of users had been targeted. Due to the code limitations if the PHP GD Duplicate Image Finder written by CatPa [2], only jpeg submissions are accepted for the purpose of this research. In addition to this, a 15 MB file size limit is enforced. This is to prevent excessive wait times when transferring the image file to the server during tests run on a large number of files. The website was designed to be lightweight, more specifically it has no extraneous scripts or applets running on the upload page. The purpose of this is to only give processing times relating to the actual upload process and not unessential scripts. Finally, the last restriction placed on the website development is cross browser compatibility. Instead of placing focus on making a website that functions across all common web browsers, it was decided that a focus on the Geko browsers such as Firefox due to the vast array of development tools available for the browser platform. This limitation will also allow for a focused effort in file management on a specific platform and will allow room for further research after the tool has been optimized.

To begin, a database was implemented in such a way that it holds a vast array of information.

to hold a hashed key for each uploaded image and the location of the file on the disk in addition to upload statistics to complete the task. An example of this schema can be seen in figure 3.1.

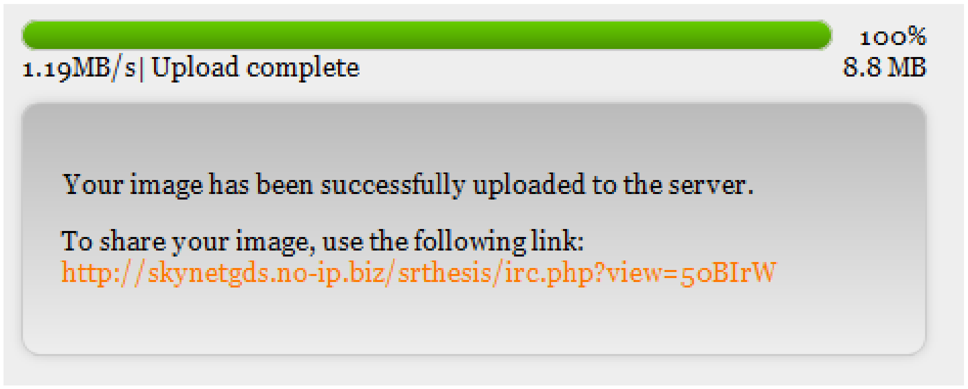
The website will run multiple scripts for each upload. The first script will be a



| Column | Type | Comments |
|---|---|---|
| ID | int(11) | Gives every image a unique ID |
| ILookup | varchar(6) | Unique URL ID Lookup |
| IName | varchar(17) | Images file name |
| directory | varchar(12) | File location from virtual server root |
| uMethod | int(11) | Upload method |
| histogram | varchar(32) | Hashed histogram of dup-reduced images |
| processTime | int(11) | Upload time to completion |

Figure 3.1: Proposed schema for file uploads

traditional upload method where the user supplies a file, it will be uploaded to the server, and the server will return a unique URL from which to access the image at a later time. During this process, each file will be assigned a new, unique name to prevent collisions upon upload. A SQL `"INSERT"` will then be run on the database for the image and will record the file's location on the server, the file name, the fact it was uploaded using the traditional method, the unique URL to access it from, and an `ID` for each insert into the database. After the completion of the process an `"UPDATE"` will be run on the database entry created by the `"INSERT"` above. The time taken to complete the task will be included in that entry and can be used to analyze the efficiency of both systems upon the completion of the tests outlined in section ??. This initial upload process can be seen in figure 3.2.



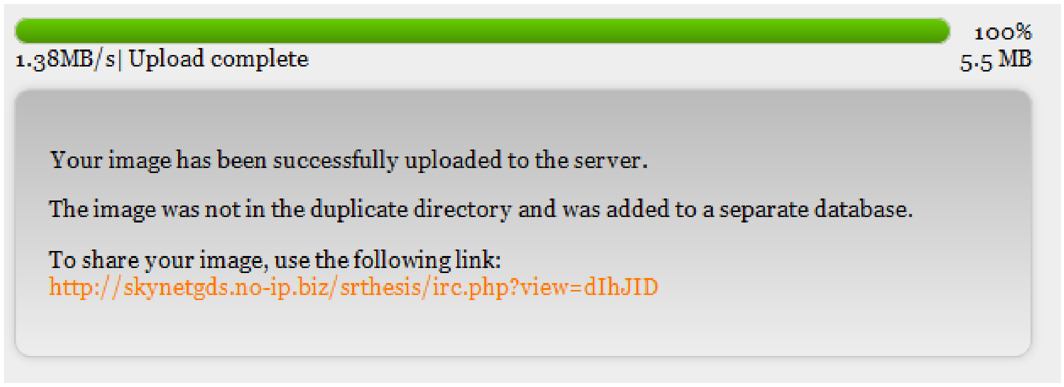
100%
1.19MB/s | Upload complete
8.8 MB

Your image has been successfully uploaded to the server.

To share your image, use the following link:
http://skynetgds.no-ip.biz/srthesis/irc.php?view=50BIrW

Figure 3.2: Successful image upload response.

After the first script completes, a second script will be called. The second script has been designed to operate in two steps, the first of which I have already implemented and tested. This function will perform mostly the same task as the traditional upload script, but will check the image being uploaded for duplicates and handle each case appropriately. First, the image will be matched in the most basic form by hashing the image file using an MD5 file hashing function, after which the resulting hash will be compared to all images in the database. Currently this segment of the code has been tested thoroughly and determined to be successfully identifying duplicates and disallowing them in the duplicate-reduced directory. If a match is not found, the script will continue to the second stage of duplicate finding function. For the time being, this second section of the function always returns no duplicate found to enable testing of only the first stage of identification. When this function is implemented in the future, the image will be taken and converted into a $16 \times 16$ black-and-white thumbnail and a histogram will be produced from that image. The server will then look into the database and check whether or not any images uploaded with the duplicate reduced system are similar to the one looking to be uploaded. After the calculation is complete, the resulting upload will be accepted by the function and stored on the servers disk. At this time a SQL `"INSERT"` will then be run on the database for the image and will record the files location on the server, the file name, the fact that it was uploaded using the duplicate-reduced method, the unique URL to access it from, the image's MD5 hashed histogram that is described below, and an `ID` for each insert into the database. After the completion of the process an `"UPDATE"` will be run on the database entry created by the `"INSERT"` above. A view of the functioning duplicate-reduced upload process can be seen in figure 3.3 when no duplicate image is located.



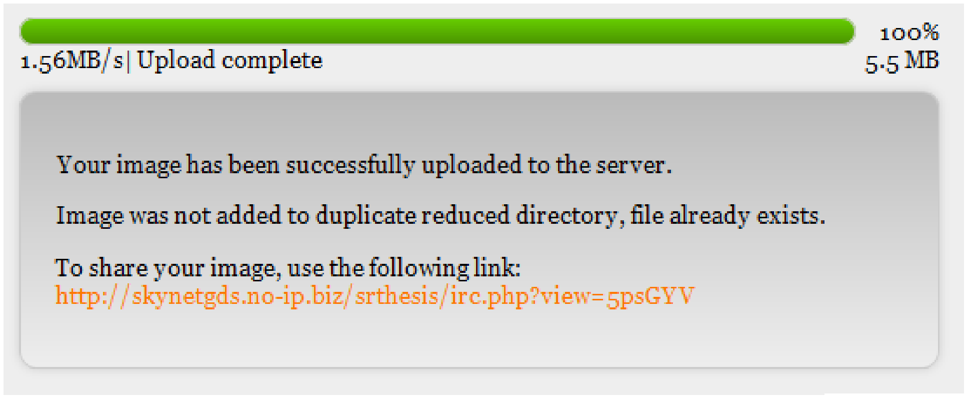
Your image has been successfully uploaded to the server.

The image was not in the duplicate directory and was added to a separate database.

To share your image, use the following link:
http://skynetgds.no-ip.biz/srthesis/irc.php?view=dIhJID

Figure 3.3: Successful image upload response when no duplicate is located.

If the image histogram hash matches the hash on the server, a color profile will be generated in real time for both images and compared pixel by pixel. If there is a match or close match, the script will compare several small, randomly chosen blocks on the images. If these match, then the image is assumed to be a duplicate and a prompt will be displayed to the user showing them the image they provided and the possible match that already exists. If the image is verified a match, the higher of the two resolutions will be kept, and the user will be given a unique link to the image. If the user decides the image is not a duplicate, the system will run an `"INSERT"` on the database as outlined earlier, and it will be added to the database. Following the completion of this process an `"UPDATE"` will be run on the last `"INSERT"` and the time taken to complete the task will be included in that entry. A view of the functioning duplicate-reduced upload process can be seen in figure 3.4 when a duplicate image is located.



100%

1.56MB/s| Upload complete

5.5 MB

Your image has been successfully uploaded to the server.

Image was not added to duplicate reduced directory, file already exists.

To share your image, use the following link:
http://skynetgds.no-ip.biz/srthesis/irc.php?view=5psGYV

Figure 3.4: Successful image upload response when a duplicate is located.

In the case of requiring duplicate files, where the user decides to upload the image regardless of uniqueness, the script will be able to differentiate between the two images with the unique image ID that is generated at the time of insertion into the database. The closest matching occurrence of an image on the server compared to a new upload will be used in the prompt and displayed to the user. This will prevent frustration with multiple prompts every time more than one duplicate is found.

If an image is indeed found to be duplicate, and the user chooses to use the higher resolution image that is on the server, a unique link will be generated, displayed to the user, and an `"INSERT"` statement will be run containing the same information as the matching file's insert did at the time of upload. This will allow multiple links that point to the same image and the user will be unaware of the system operating in the background which will provide a consistent experience. An outline of this full process can be seen in figure 3.5.

When a user accesses the file from the provided link, the system will run a query that looks up the image identifier provided in the URL. If it matches an image on the server, the image location will be used to provide the image to the user for viewing as seen in figure 3.6. The user never notices a difference, but on the server side we have ensured file redundancy has been eliminated and possibly improved user experience by providing the user with higher quality content than what they were expecting. If the requested image is not found, a 404 "Image cannot be found." error will be displayed to let the user know something went wrong.

## 3.3 Histogram Comparison

Algorithm 3.7 (from ) shows a high-level description of an algorithm. There are many options for the display of pseudocode; this uses the `algorithm2e` package , but there are a number of others available at the Comprehensive TEX Archive Network (`ctan.org`). Using any of these other packages might require the additon of one or more "`\usepackage{...}`" commands in the main `AllegThesis.tex` file.

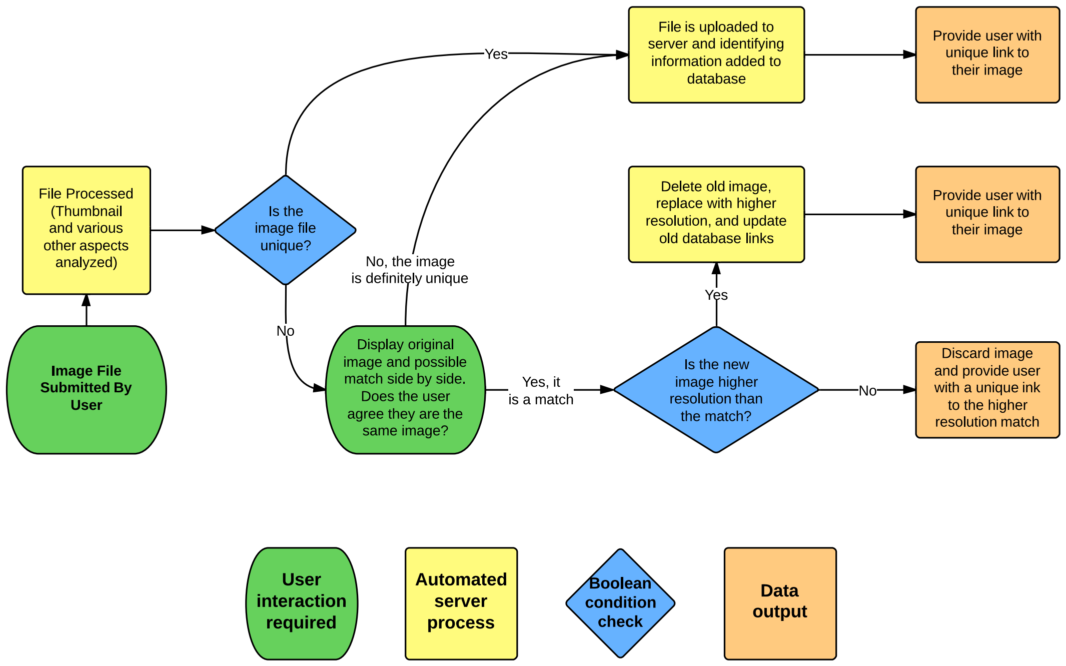

File is uploaded to server and identifying information added to database
Provide user with unique link to their image
Yes
File Processed (Thumbnail and various other aspects analyzed)
Is the image file unique?
Delete old image, replace with higher resolution, and update old database links
Provide user with unique link to their image
No, the image is definitely unique
Yes
No
Image File Submitted By User
Display original image and possible match side by side. Does the user agree they are the same image?
Yes, it is a match
Is the new image higher resolution than the match?
No
Discard image and provide user with a unique ink to the higher resolution match
User interaction required
Automated server process
Boolean condition check
Data output

Figure 3.5: Streamlined upload process

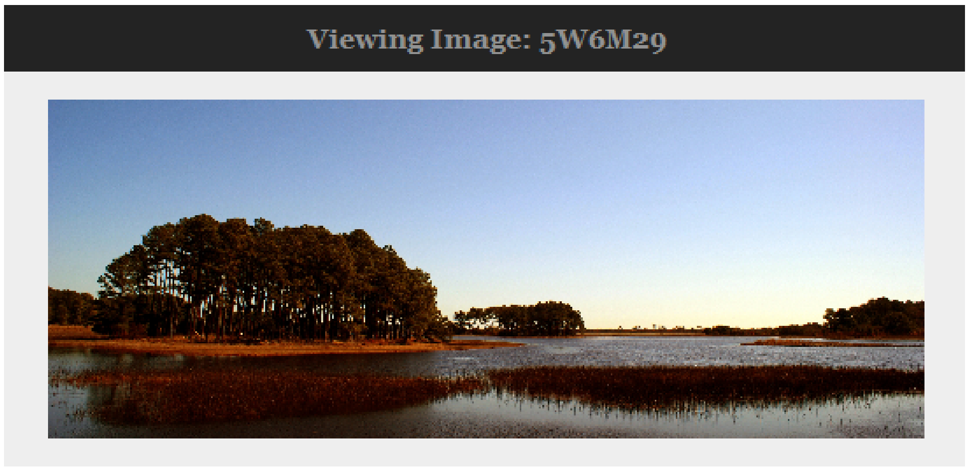


Figure 3.6: What users see when clicking a shared link.

```
Data: this text
Result: how to write algorithm with LaTeX2e
initialization;
while not at end of this document do
    read current;
    if understand then
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end
end
```

Figure 3.7: How to write algorithms (from )

## 3.4 Experiments

Figure 3.8 shows a Java program. There are many, many options for providing program listings; only a few of the basic ones are shown in the figure. Some thought must be given to making code suitable for display in a paper. In particular long lines, tabbed indents, and several other practices should be avoided. Figure 3.8 makes use of the `listings` style file .

## 3.5 Threats to Validity

```
public class SampleProg
{
   private int dummyVar; // an instance variable

   public static void main(String[] args)
   {
      System.out.println("hello world.");
      // Avoid long lines in your program; split them up:
      dummyVar = Integer.parseInt("1234")
           + 17 * ("abcde".substring(1,3).length()
           + 11 - 1000;
      // Use small indents; don't use the tab key:
      for (int i = 0; i < 10; i++)
      {
         for (int j = 0; j < 10; j++)
         {
            if (i > j)
            {
               System.out.println(i);
            }
         }
      }
   }
}
```

Figure 3.8: `SampleProg`: A very simple program

# Chapter 4

# Results and Evaluation

Another possible chapter title: Experimental Results

# Chapter 5

# Discussion and Future Work

This chapter usually contains the following items, although not necessarily in this order or sectioned this way in particular.

## 5.1 Summary of Results

A discussion of the significance of the results and a review of claims and contributions.

## 5.2 Future Work

## 5.3 Conclusion

# Appendix A

# Java Code

All program code should be fully commented. Authorship of all parts of the code should be clearly specified.

```
/**
 * SampleProg -- a class demonstrating something
 * having to do with this senior thesis.
 *
 * @author   A. Student
 * @version  3 (10 December 2013)
 *
 * Some portions of code adapted from Alan Turing's Tetris program;
 * relevant portions are commented.
 *
 * Revision history:
 *     10 December 2013 -- added ultra-widget functionality
 *     18 November 2013 -- added code to import image files
 */
public class SampleProg
{
   private int dummyVar; // an instance variable

   public static void main(String[] args)
   {
      //==================================================
      // This section of code adapted from Alan Turing's
      // Tetris code. Used with permission.
      // http://turinggames.com
      //==================================================
      System.out.println("hello world.");

      dummyVar = Integer.parseInt("1234")
           + 17 * ("abcde".substring(1,3).length()
           + 11 - 1000;

      for (int i = 0; i < 10; i++)
      {
         for (int j = 0; j < 10; j++)
         {
            if (i > j)
            {
               System.out.println(i);
            }
         }
      }
   }

   /**
    * foo -- returns the square root of x, iterated n times
    *
    * @param     x   the value to be interatively processed
    * @param     n   number of times to iterate square root
    * @return    sqrt(sqrt(...())) [n times]
    */
   public double foo(double x, int n)
   {
       for (int i = 0; i < n; i++)
           x = Math.sqrt(x);
       return x;
   }
}
```

# Bibliography

[1] Angela Bradley. Renaming PHP Uploads. `http://php.about.com/od/advancedphp/ss/rename_upload.htm`, 2011. [Online; accessed 10-October-2013].

[2] CatPa.ws. PHP GD Duplicate Image Finder. `http://www.catpa.ws/php-duplicate-image-finder/`, 2010. [Online; accessed 13-September-2013].

[3] Dictionary.com. Definition of Photo Sharing. `www.dictionary.com`, 2013. [Online; accessed 20-November-2013].

[4] All Things Digital. Meeker: 500 Million Photos Shared Per Day and That's on Track to Double in 12 Months. `http://allthingsd.com/20130529/meeker-500-million-photos-shared-per-day-and-thats-on-track-to-double-in-12-m` 2013. [Online; accessed 20-November-2013].

[5] Ubuntu Documentation. ApacheMySQLPHP - LAMP Server Setup Guide. `https://help.ubuntu.com/community/ApacheMySQLPHP`, 2013. [Online; accessed 21-August-2013].

[6] Jun Jie Foo, Justin Zobel, Ranjan Sinha, and S. M. M. Tahaghoghi. Detection of Near-duplicate Images for Web Search. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, CIVR '07, pages 557–564, New York, NY, USA, 2007. ACM.

[7] The PHP Group. GD and Image Functions. `http://php.net/manual/en/ref.image.php`, 2013. [Online; accessed 13-October-2013].

[8] The PHP Group. POST Method Uploads. `http://de.php.net/manual/en/features.file-upload.post-method.php`, 2013. [Online; accessed 09-September-2013].

[9] Snapchat Inc. Snapchat Support. `http://support.snapchat.com/`, 2013. [Online; accessed 9-December-2013].

[10] David C. Lee, Qifa Ke, and Michael Isard. Partition Min-hash for Partial Duplicate Image Discovery. In *Proceedings of the 11th European Conference on Computer Vision: Part I*, ECCV'10, pages 648–662, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] Dejan Marjanovic. PDO vs. MySQLi: Which Should You Use? `http://net.tutsplus.com/tutorials/php/pdo-vs-mysqli-which-should-you-use/`, 2012. [Online; accessed 02-October-2013].

[12] TecNick LTD Nicola Asuni. TESTIMAGES. `www.tecnick.com/public/code/cp_dpage.php?aiocp_dp=testimages`, 2013. [Online; accessed 3-November-2013].

[13] nixCraft. Ubuntu Linux: Install or Add PHP-GD Support to Apache Web Server. `http://www.cyberciti.biz/faq/ubuntu-linux-install-or-add-php-gd-support-to-apache/`, 2012. [Online; accessed 30-September-2013].

[14] NTP Software. Survey Says Nearly Two-Thirds of Files on Primary Storage Are Stale. `www.ntpsoftware.com/pressroom/survey-says-nearly-two-thirds-files-primary-storage-are-stale`, 2013. [Online; accessed 31-October-2013].

[15] S H. Srinivasan and Neela Sawant. Finding Near-duplicate Images on the Web Using Fingerprints. In *Proceedings of the 16th ACM International Conference on Multimedia*, MM '08, pages 881–884, New York, NY, USA, 2008. ACM.