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

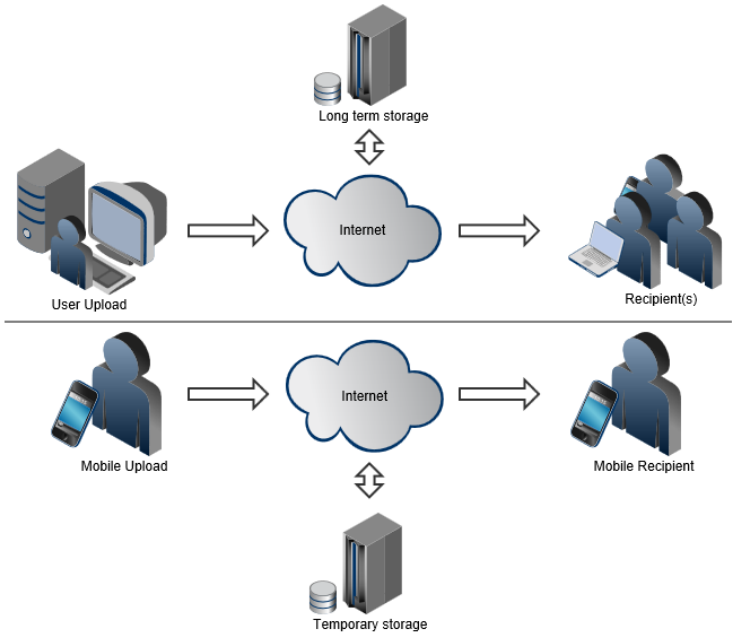

Figure 1.1: Various Image Hosting Service Structures

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

Related work to be completed shortly.

2.1 Primary Sources

Primary Sources coming soon

2.2 Recent Results

Recent results still being compiled and coming soon.

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 3.1 To 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. he image sharing functionality of the website requires several different pieces of information to be stored in a database. The first column of the database houses the identifier for each entry. This is known as the primary key, which is a column in a database where all entries must be unique and the key is used to identify the information in the row. This identifier is used by the website to track the order of

the submissions to the server since the date uploaded is not important to the research and will not be tracked. The next column, as seen in Figure 3.1 is the `ILookup` column. This is what the website uses to look up each image location on the server when supplied with a URL containing this identification code. This column must always have a value so the `NOT NULL` flag was set. The third column is `IName`, the column that houses the actual file name of the image being stored on the server. This column must also have a value so the `NOT NULL` flag was set. This is concatenated to the end of the `directory` entry which cannot be null, and provides the website with the exact location of the image file on the server with respect to the Linux root directory.

Each of the remaining database columns are specific to the image de-duplication scripts. First, the uMethod column is nothing other than a single integer that marks what upload method was used to place the image on the server. If this number is st to "0", the image was uploaded using the non duplicate reducing functions, on the other hand, if this number is set to "1" it is known that the image was uploaded and checked for duplicates before committing to long term storage. Both the hash and fingerprint fields contain information that allows the scripts to rapidly search the database for duplicate files when a new file is uploaded. Both of these columns are allowed to have a null value. This is allowable due to the fact that non duplicate reduced images will not have any image matching data associated with them. These non duplicate reduced images will be discussed in Section 4 when discussing the base case that the results will be compared to. Finally, the last column tracks the total time in milliseconds that it took to upload the image to the server. This is not allowed to be null as both the base case and the duplicate reduced case will require this information. In Section 4, this data will be used to compare the resulting data gathered by uploading images in a traditional manner and using this duplicate reduced function being researched.

Database: thesisDB | Table: shareTracker			
Name:	Type:	*Description:*	Extra
ID	int(11)	*Gives every image a unique ID*	Primary Key Auto-Increment
ILookup	varchar(6)	*Unique URL ID Lookup*	Not Null
IName	varchar(21)	*Images file name on server*	Not Null
directory	varchar(15)	*File location from server root*	Not Null
uMethod	int(1)	*Upload method used*	Not Null
hash	varchar(40)	*Hash of the image for exact dup matching*	
fingerprint	varchar(32)	*The MD5 fingerprint of the histogram array.*	
processTime	int(11)	*Upload time, from start to completion*	Not Null

Figure 3.1: Proposed schema for file uploads

In order to test the effectiveness of the research being discussed, a base case must be created to compare the results of running the algorithms against. To create this data, the website will perform one function before moving on to the duplicate

reduction test. A separate directory was created on the server where every single image sunmission will be place regardless of whether it is duplicate or not. This will mimic the upload process of a non duplicate reducing image sharing website. Each entry into this folder will be placed in the database exactly the same as the duplicate reduced entries are, but will contain no image identification information. This first process will be referenced from henceforth as a traditional upload method. This upload method will be unknown to the user as it will not return a URL allowing access to 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 (which will not be given to the user), 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 then 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 4.

After the initial upload completes, a second script will be called. The second script has been designed to operate in two steps. This function will perform mostly the same task as the traditional upload, 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 using the following SQL Statement: `"SELECT * FROM ‘share_tracker‘ WHERE ‘uMethod‘ = ’1’ AND ‘hash‘ = :fileHash’"`. If this statement returns any results relating to a matching hash on a duplicate reduced uploaded image, it has found an exact matching image.

In the event where a match is not found, the script will continue to the second stage of duplicate finding function. This function will operate in several different stages. The first stage toward finding an approximate image match is creating a fingerprint of the image file. In order to do this, the script will create a GD image object by using the PHP GD Image Library that was configured on the server in Section 3.1. After the object is created, the GD Library has built in functions that allow the iteration through every pixel in the image and view each pixel's Red, Green, Blue value, or RGB value. These values will then be used to return a fingerprint in the form of an MD5 hash of the color frequency of the image. This color frequency is also known as a color profile. The next step to identifying a duplicate image would be to query the database for any images that have a perfectly matching color profile. This profile will be the same for an image no matter the rotation, size, dimension, or other variation as long as the image color and contrast are left unmodified. At this point, if there is not a matching color profile, it can be said that there is most likely not a matching image on the server and the function will return no matches found.

If the color profile does in fact match that of one or more images on the server, the script will then make a 16×16 pixel full color copy of the image being uploaded in addition to 16×16 copies of each of the images with a matching color profile.

This process will also utilize functions provided by the GD Library that will allow pixel by pixel comparison of each image. Due to the nature of re-sizing an image, some colors will be averaged together. This will allow the comparison of both images even if they were different dimensions as the resulting 16×16 files will have similar averages after re-sizing if they are in fact a match. Because of a possibility of slight variations in color, this function will not look for exact match, but will allow a set amount of deviation between two thumbnails. To compare the pixels of both images, the RGB values of the corresponding pixels on each image will be analyzed. First, the red value of a pixel will be subtracted from the red value of the corresponding pixel's red value in the other thumbnail. The absolute value of the difference will be recorded. The same process will be completed for the green and blue values of that pixel. Once the difference from that pixel is calculated, the same comparison will be performed on each of the remaining pixels. In order to allow for very slight variations in color with each pixel, the resulting total calculated deviation will be divided by the number of pixels to 'normalize' the result. After that calculation is complete, if the final deviation is less than or equal to the limit set by the website administrator, the function will return that a matching image was found.

If after all of the above processes are completed and no duplicate image is located on the server, the image 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 long term 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, the thumbnail created for histogram comparison, 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 3.2 when no duplicate image is located.

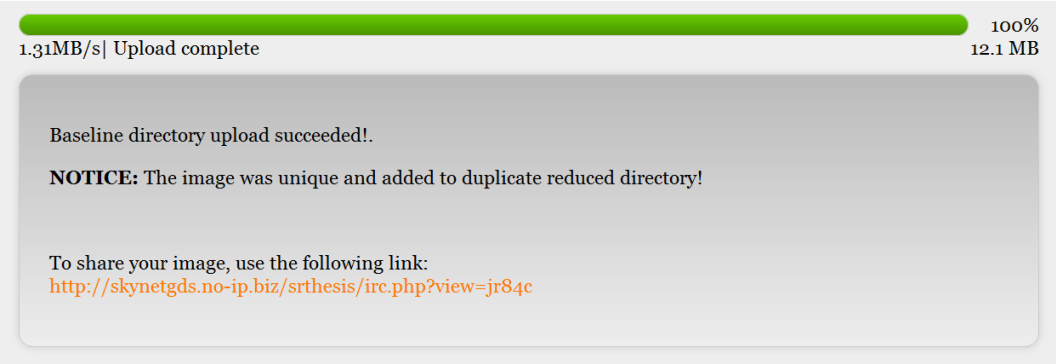

Figure 3.2: Successful image upload response when no duplicate is located.

If the image is determined to be a match after the completion of the full image hash function or the completion of the color profile and 16×16 pixel by pixel comparison, the submitted 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 user will be given a unique link to the image that is already on the server, and the image being uploaded will be discarded as a duplicate. At this time the system will run an `"INSERT"` on the database linking the new identifier to the image already on the server, and it will be added to the database. If the user decides the image is not a duplicate, the system will run an `"INSERT"` statement 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 3.3 when a duplicate image is located.

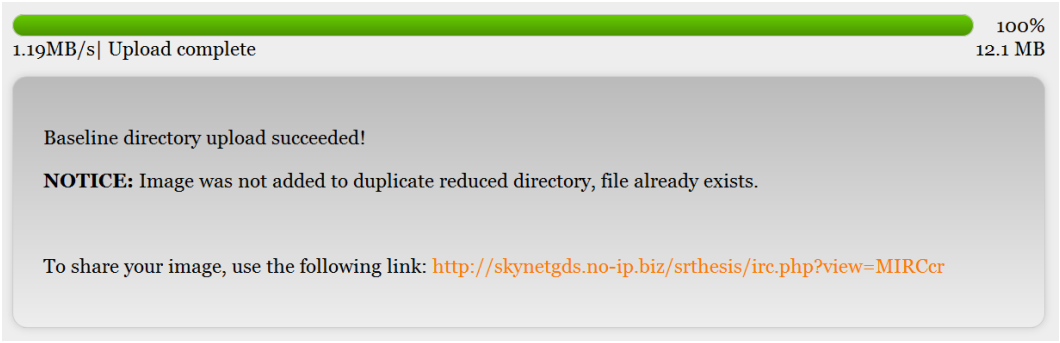

Figure 3.3: Successful image upload response when a duplicate is located.

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. This technique will also allow multiple links that point to the same image while leaving the user unaware of the system operating in the background to provide a consistent experience. An outline of this full process can be seen in figure 3.4.

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 3.5. 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 Color Profile and Histogram Comparison

As mentioned in Section 3.2, an image is composed of RGB values associated with every pixel in an image. In order to use these values in an effective manner, a

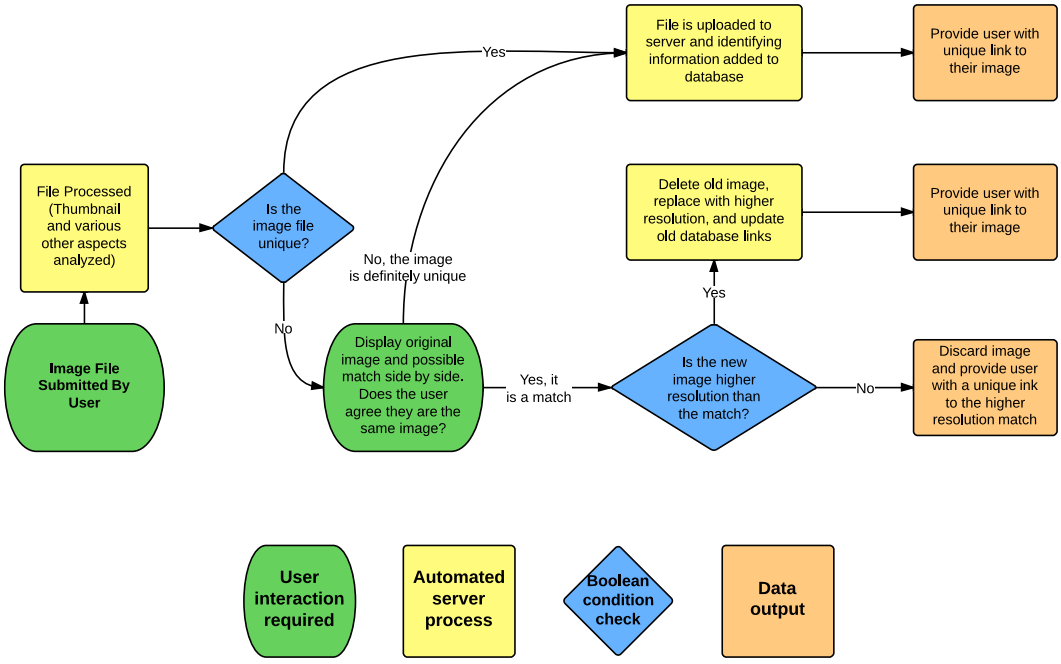

Figure 3.4: Streamlined upload process

Figure 3.5: What users see when clicking a shared link.

thumbprint of the color profile had to be created. This color profile represents a histogram and allows the visualization of each colors frequency. This section will break down the process of comparing histograms and their accompanying data and explain the reasoning behind the comparison method chosen for this research.

As before mentioned, each pixel in an image is composed of an RGB value that allows the storage of the correct mixture of each color in that pixel. This value is separated into three separate ranges of 0-255 values where the higher the number the brighter the color. As seen in Figure 3.6, a histogram has been generated using a popular photo editor by the name of Paint.NET. On the right hand side you can see four different overlaid histograms. The red indicates the frequency of each red value, the lowest of the 0-255 values being at the bottom of the image, and the highest at the top. The green and blue indicate the same for their respective colors. The yellow section of the histogram represents the different shades of black in the image, but we are not concerned with these values as blacks, whites, and grays can be represented by the RGB values. For example $255, 255, 255$ represents a solid white pixel while $000, 000, 000$ represents a pure black pixel, thus these colors do not require their own array.

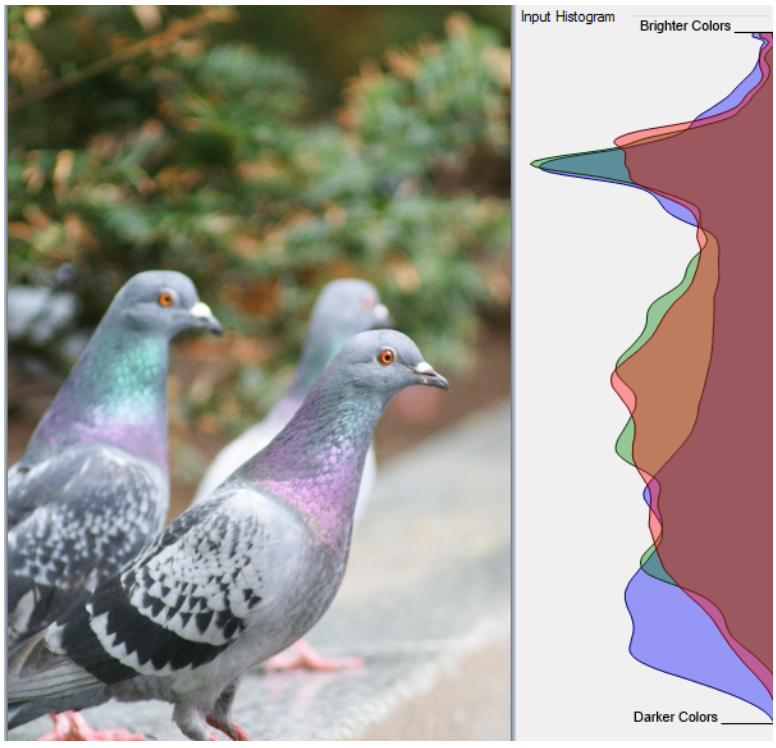

Figure 3.6: Visual representation of an images histogram

This collection of values that make up the histogram is where the comparison process begins. For any comparison of data to occur, each pixel's RGB values must be recorded into three arrays of size 256. The index of the red, green, and blue arrays will correlate to the 256 possible colors within the respective color. As a color

is encountered, a counter in the code will increment the value by one at that array index. This will allow the tracking of the frequency of each of the RGB values. It does not matter what pixel these are related to as the overall number and frequency of each color is of the only concern.

At this point, every RGB value in the image should be stored within the colors respective array. This is the array of values that can be graphed using a histogram, and the visual of the color profile can be printed out for visual inspection. Since we will be using a function to process the values, the actual generation of said histogram would be unnecessary. These values can be handled and compared in a few ways, but the best method for the purpose of this research much be chosen. This array could be processed using the PHP `serialize()` function and stored in the database for later examination. By doing this, each of the arrays would require separate serialization, and all three would require their own columns. With this method, the array could be pulled from the database and run through the `deserialize()` function to return the array to its original state. This would be beneficial for applications that would require the comparison of each individual array index to another array. Due to the large amount of processing time required to iterate over multiple arrays of size 256, this is not optimal. In addition to this, we are not concerned with calculating the difference in color profiles, but are only concerned with an exact match.

This requirement better lends itself to a different method of histogram value comparison. Due to the fact that we only need to find exact matches and that the system accounts for a possible false positive, hashing the the three arrays will suffice. For this process, the three arrays will be concatenated in the order `Red.Green.Blue`. This will generate one array with 1068 indexes representing every possible RGB value. This value will then be taken and processed with the PHP $md5()$ function which will return a 40 character alphanumeric string representing the array. When an uploads fingerprint is created, the database will be searched for an exact match hash. If a matching hash is found, it is known that a photo with the same color profile is already on the server. In the event of a hash collision where two different histograms have the same resulting hash, the script will be performing a pixel by pixel comparison anyway and will be able to decide that the images aren't unique, so this is not of concern.

Due to the fact that the only images of concern are exact color profile matches, it is not required to store the values of the color profile in a recoverable format. This leads away from storing either a plain text array of values in a database, or storing a serialized version of the array in the database. Because of this, it is acceptable for a 40 character alphanumeric hash to be stored in the database and directly compared to the hash of a newly submitted image in order to detect a possible duplicate image.

3.4 Threats to Validity

With any area of research comes inherent shortcomings no matter the care taken to eliminate free variables. By creating a locally hosted server and excluding any extraneous scripts from the website, these variables were controlled to the greatest extent

possible. Due to the nature of a publicly hosted web server, there is always a chance of malicious attack which could skew the results on one way or another. As discussed in Chapter 4, four sets of experiments were performed. One test was performed offline and was performed using only a standard test image library with a very specific image alteration to each image. This environment gives the best possible chance of gathering unbiased results. The second test to be performed utilized the same image library, but was run over an internet connection instead of a LAN connection. This allowed for the performance testing over a connection of fluctuating load and speed. An identical set of two experiments were tested using a set of 10 photographs taken around the Allegheny College Campus during the winter months. This would give a collection of unique images, some of which would inherently have similar color profiles due to the reduced color intensity of the winter months. These tests should give an accurate representation of performance in both laboratory conditions, and in a real world application. This also opens the project to uncontrollable variables as discussed that can greatly vary results over a number of trials.

Finally, the last variable that was not able to be determined was a possibility of image corruption. Due to the nature of images, it is possible that minor file corruption can occur on physical media. Since this is nothing more than the loss or incorrect representation of data, an image will still be able to process. In addition, if corruption occurs on a very small number of pixels, it is possible that the image will visually look identical to the original but still differ enough that it will be seen as an original image instead of a duplicate. This corruption can happen due to a momentarily lost connection, a web browser mis handling an HTTP request, or a server side fault. There are currently numerous open bugs in both the PHP language and the Apache server applications, none of which were researched to ensure the proper function of scripts. It was assumed that a functioning script returning expected results during trial runs is a fully operational script when running final tests and analyzing the results.

Chapter 4

Results and Evaluation

Evaluation Strategy and results to be discussed in this section.

Chapter 5

Discussion and Future Work

Conclusion to be written int he Spring after research and testing concludes.

5.1 Summary of Results

Coming soon.

5.2 Future Work

Coming soon.

5.3 Conclusion

Coming soon.

Appendix A

Website Framework Code

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

Appendix B

Duplicate Image Detection Code

Appendix C

Miscellaneous Code and Configurations

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-months/, 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.