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

Braden D. Licastro
Department of Computer Science
Allegheny College
licastb@allegheny.edu
http://skynetgds.no-ip.biz/srthesis

November 22, 2013

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 identical images.

1 Introduction

Computers are absolutely everywhere, and society is becoming more digitized every day creating a need for storage of large amounts of data. When running a website, webmasters must manage storage space utilization, and the physical disks in the servers being used may not be centralized under one roof, but may in fact be located around the world. 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 [13]. 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, $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, especially when the data in question is backed up regularly. This research will target the duplicate data problem encountered by the large photo sharing websites similar to Photobucket and imgur. Most of these type of sites already implement systems that limit the number, size, type of image, and compress the uploaded images to save additional space, but there is still room for significant improvement. To address this problem, the research aims to create an intelligent image uploading system capable of identifying near-duplicate image uploads on-the-fly. An implementation of a duplicate-reducing image sharing website will be introduced that is capable of managing large volumes of images using a database of identifiers and keys that identify each

individual file. By using this system, it is possible for the files to be stored in one central location or distributed across several systems while avoiding excessive processing and wait times caused by traditional duplicate image matching systems. By using text based entries in a database, a hashing and collision resolution function can be implemented which is capable of producing a unique identification key for each file in the collection. Because this key is unique for each different file, it is a reliable gauge of uniqueness. When multiple files are found to be identical only one physical copy will be kept but multiple pointers to the database entry will allow easy access to the image from multiple locations. On large, distributed network storage servers, the benefits become ever more apparent. As users upload images, the system will create an identifier for each file and search for a match in a database, thus allowing the user to add to and access the data, but eliminate the overhead of storing numerous copies of identical data. An increase in computation time will exist but should be minimal as text comparisons will be the primary task in determining uniqueness. This additional time cost should be outweighed by minimized physical storage requirements and costs of data backup. The implementation of this system could be used in tandem with other methods of disk space cost reduction technologies. By not only using the proposed method of duplicate reduction, it could in theory be possible that with image expiration times, file size maximums, and similar technologies, storage needs will plateau after the initial surge of additions and the expiration time has passed for the earliest uploaded file. This would allow webmasters to better predict overall needs and better predict upload trends and react accordingly with preventative maintenance and any required space additions.

2 Related Work

Though I will not be comparing existing images located in a database, the paper by Lee, Ke, and Isard [9] can be useful in method of approach. During their research they utilized a partition min-hash function for discovering near-duplicate images. Partition by hash functions are used to break down a large data set into a number of equal sized segments that can be identified by a generated hash. Instead of looking at the image as a whole as many other algorithms do, it looks at the image as several smaller pieces through the use of the partition hashing function. They claimed that the algorithm is actually faster and more effective than standard min-hash functions which look at the entire image. Since the proposed research I will be performing is highly efficiency sensitive due to its web based structure, I will consider implementing a variant of this method of comparison.

Another article that will be vital to creating an effective and efficient image matching algorithm will be the 2010 paper by Srinivasan. This research was targeted at web-based image matching, which is exactly what I am going to perform. This research claims that traditional near-duplicate detection systems are not applicable for the de-duplication of large-scale web image collections, [14] the research performed targeted an image matching system which was scalable, highly efficient, and effective.

In order to perform the effective image matching the authors required, they decided to implement a thumbnail matching based system. This algorithm would generate a 130-bit thumbnail representation of the image and was capable of finding near-duplicate images while operating in $O(1)$ time [14].

Figure 1: Possible image manipulations

When using this thumbnail method, the authors were able to adjust automatically for differences in resolution, arbitrary amounts of cropping, caption, logo, and other manipulations, in addition to color and rotation variations, examples which can be seen in figure 1. Implementing this method could benefit the research being proposed, as each of these are highly probable manipulations, and will likely each be included in the final tests performed on the implemented system.

Due to the web based nature of the proposed research, another article, written by Foo, Zobel, Sinha, and Tahaghoghi in 2007 is highly related to the proposed research. This paper discusses the topic of web search based image matching [6], mainly the finding and determination of types of copyright infringement as most near-duplicate images are derived from one source. The goal of their research was not to derive an algorithm capable of matching images, but to locate duplicate and near-duplicate images on the web using search engines, and determine the most common methods of image alteration leading to redundancy and possibly copyright infringement. Their research will be invaluable to the proposed research as it will then be possible to target the most common methods of image alteration when creating a matching algorithm. To locate their image set to work with, they used the most popular search queries of 2005 and collected the number of images, and determined the number of unique images based on the results returned. As a note, images with non-unique URLs were not included to prevent false duplicates from the same sites. In conclusion, the authors found that the most common alterations, ordered from one to 10, one being the most common, were as follows [6]:

1. **Combination:** Images with more than one alteration.
2. **Scale:** Images that differ in size.
3. **Crop/Zoom:** Images that are cropped from the original.
4. **Picture in Picture:** Images that contain another image.
5. **Contrast:** Images that have adjusted contrast.
6. **Border/Frame:** Images that have added borders or frames.

7. **Grayscale:** Images that have been converted to grayscale.
8. **Recoloring:** Images with colors that have been modified.
9. **Mirror:** Images that have been mirrored to prevent copyright infringement detection.
10. **Rotate:** Images that have been rotated.

The research proposed will not focus on every one of these common parameters that have been found, but will focus on a select group of these common alterations outlined in section 4.

Finally, the most relevant information found will provide a starting codebase for the algorithm and research. An online coding tutorial provider, CatPa, outlined a method of using PHP libraries to generate real-time thumbnails and hashes of images and compare them to ones currently located on a server before uploading [2]. If the image is a duplicate it would be denied the privilege of being uploaded and the function would notify the user. This system is extremely limited as it provides the user with no way of adding an image to the server if it is not a duplicate and is only a false positive, and if it is unique, does not provide them with a method of accessing the already-present file.

The image comparison code developed by CatPa will be used as a base to generate a fully functional image sharing site with duplicate-reduction systems and allow for the testing of the effectiveness of such a system over traditional methods of sharing with duplicates allowed. The system created by the author [2] generates a 16×16 thumbnail of each image on the server, and of the image being uploaded. From here, the algorithm will generate and compare black and white and color histograms, the thumbnails, and determine if it is a duplicate based on the allowable difference threshold setting which will be determined after running initial tests outlined in section 4 and analyzing the results [2]. The proposed research will utilize this exact method, but will provide code improvements to utilize less memory by implementing the PHP Improved libraries and also checking images for immediate similarities in resolution, exact image matching with a full MD5 image hash, and by comparison of metadata. These methods require minimal calculation, just a simple comparison of strings compared to the generation of histograms and resizing of images. After the initial string comparison approach, the code will be used to calculate the average deviation between the two images and will allow easy expansion of the system to provide a faster, more effective image matching system.

3 Method of Approach

In order to implement the proposed research, the website will be run on a custom server with an allocated 5 gigabytes of dedicated storage, 8 gigabytes of RAM, and a Core i3 2.43 GHz processor. All services and programs will be shut down during the experiment except for Apache, MySQL, PHP, and Webmin to ensure consistent results. Throughout the duration of the experiment, no software updates will be installed other than security related operating system updates. The website must be designed to be lightweight, have no extraneous scripts or applets running on the upload page, and will be compatible with WebKit browsers such as Firefox. This limitation will 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 start, a database will be implemented so that it is able 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 2.

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 2: Proposed schema for file uploads

The website will run multiple scripts for each upload. The first script will be a 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 4. This initial upload process can be seen in figure 3.

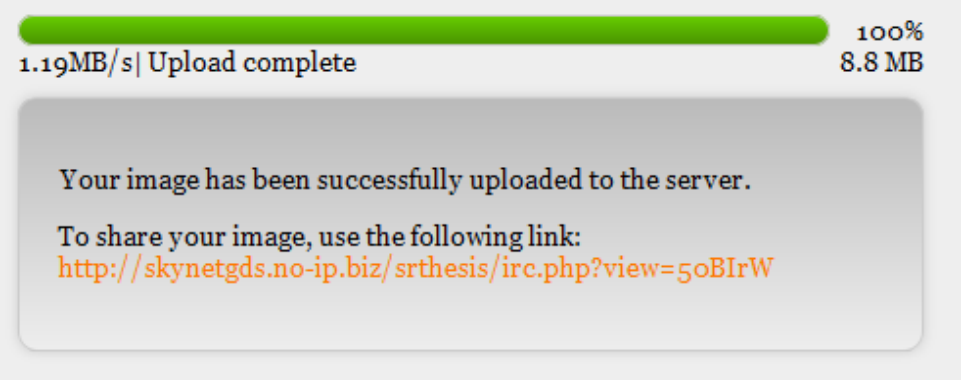

Figure 3: Successful image upload response.

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 16×16 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 4 when no duplicate image is located.

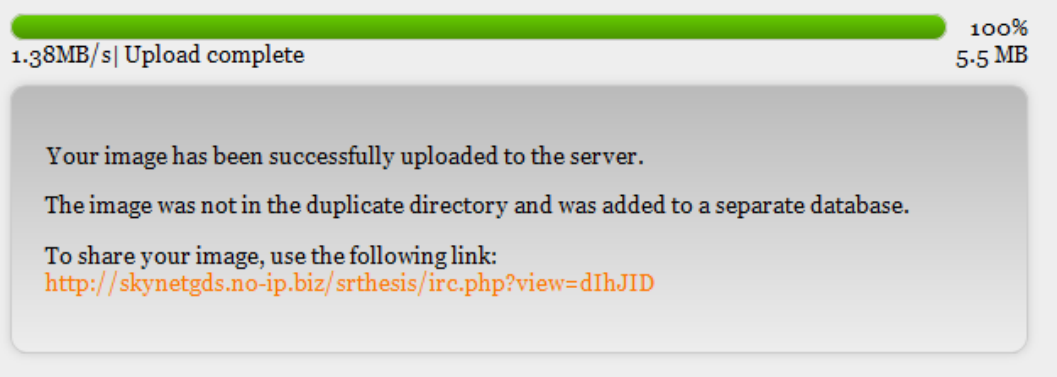

Figure 4: Successful image upload response when no duplicate is located.

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 5 when a duplicate image 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

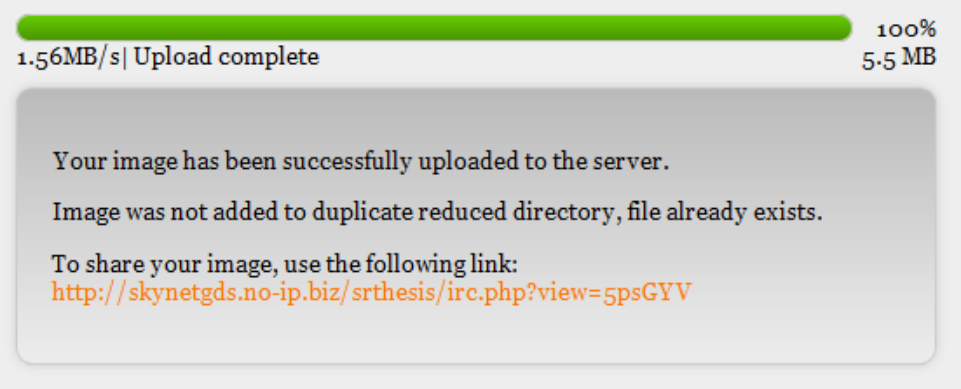

Figure 5: Successful image upload response when a duplicate is located.

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 6.

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 7. 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.

4 Evaluation Strategy

Evaluating the effectiveness of the website will entail several benchmark readings. The total number of files, number of unique files, and the overall size of the benchmark folder versus the duplicate reduced upload folder will all be used in determining the effectiveness of the image comparison algorithms implemented in the website. To perform the tests, a fixed set of standardized test images from TecNick will be used [11]. These images will consist of a collection of 15 artificially generated gray-scale, 15 natural gray-scale, and 15 natural RGB images in addition to 5 duplicate images in each category. A survey posted by NTP Software stated that approximately 20% of data on storage systems can be attributed to duplicate data [13]. By using 15 unique and 5 duplicate images in each category, it is possible to represent this average and determine how effective the reduction method is. The duplicates will consist of one of each of the following

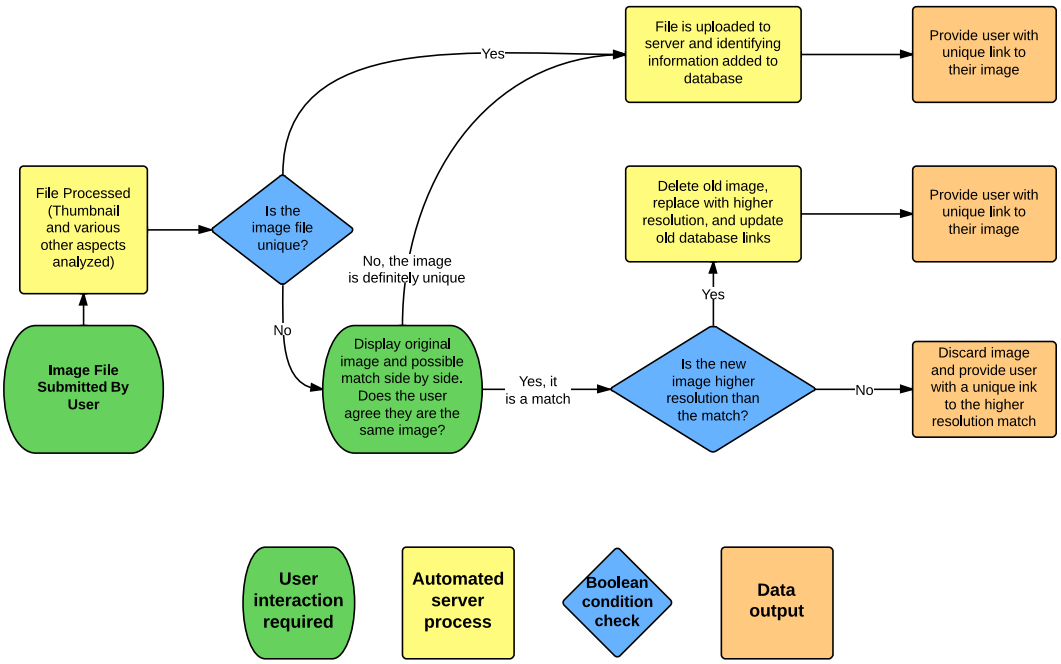

Figure 6: Streamlined upload process

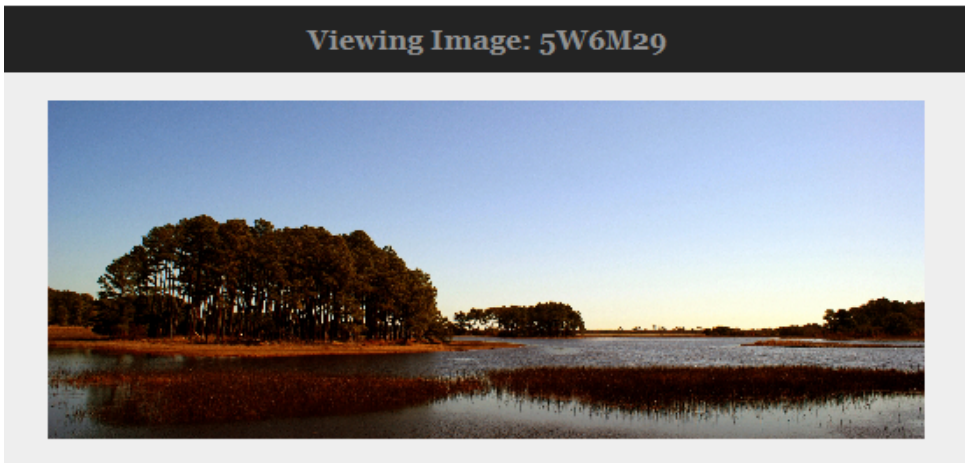

Figure 7: What users see when clicking a shared link.

images: identical, diminished resolution, text overlaid, reduced size, and a color/contrast correction.

$$\left(\frac{Base - Reduced}{Base}\right) * 100 = \%ImprovementOverBase$$

Figure 8: Equation for determining efficiency improvements

In order to test the algorithm, all 15 unique images will be uploaded to the server using the built in upload function. Any false positive matches will be noted and the algorithm will be adjusted to reduce the number of false positives. At this point, the baseline folder and the duplicate reduced folders should contain the same set of data. Next, the duplicate images will be uploaded to the server using the same method. The system should find all remaining images as duplicates and prompt the user with the option to upload anyway or use the existing image. For the test I will not allow the upload. After all images have been uploaded, the total number of files in the baseline directory will be compared to the number of files in the duplicate reduced directory using the algorithm in figure 8, where base is the baseline directory number to be compared and reduced is the duplicate reduced directory number to be compared. The directory sizes and average upload times will also be evaluated using the algorithm in figure 8. The evaluated result, if positive, will denote a positive improvement in efficiency compared to the baseline directory, and a negative result is interpreted an inefficiency over the baseline directory.

After analyzing the results of the test outlined above, the algorithm will be refined to provide more accurate results and the folders and database will be reset. At this point, the tests will be run again, and any changes in effectiveness will be noted. Because the system implements a text-based database, the size difference between the baseline and duplicate reduced will be negligible as no image specific information is stored. In addition, the number of entries will be identical due to the upload process outlined in 3, which generates a unique URL for every image upload request whether the image is uploaded or an existing one on the server is used. Due to the above stated reasons, the database will not be included in space efficiency calculations.

5 Research Schedule

The research schedule for the project is to be tentatively based on the following deadlines. Due to the nature of the project and the wide array of pre-existing software required needing to be modified and implemented, unseen hurdles may arise.

Phase 1:1 month: I decided to use the familiar Ubuntu operating system and set up a fully functional LAMP server to host website and provide storage of submitted images. During this time, I also begin learning about and installing PHP and JavaScript libraries so code implementation could begin shortly after the completion of server setup.

Phase 2:2-3 months: Develop a working image sharing website which is capable of satisfying the requirements outlined in 3. Set up two basic file and database structures to be used throughout the experiment. One will be used as a baseline and the other structure will function as the duplicate-reduced system. Both will host the image files, file information, and time taken to add files to the

database. The system will also track the average upload time for each method, the total number of files, and the total size of each upload directory.

Phase 3:1 week: During this time I will run various tests on the website using the proposed methods in 3 and record the results of the experiment. After the new image files have been added to the collection, some unique, and others duplicate, the submissions directory on the server will be scanned for duplicates and ensure none were accidentally added. Results to be recorded are the final number of files, the number of duplicate images, and the total storage space used.

Phase 4:Remaining time: The results gathered throughout the test process will be evaluated and the website algorithms tweaked and finalized. After modifications are complete tests will be rerun and the final results will be evaluated.

6 Conclusion

With computers becoming ever more prevalent and data being generated and shared at a rapidly increasing rate, it is becoming more and more necessary every day to reduce the redundancy of shared data and create more efficient sharing systems. By removing duplicate image files from sharing sites and preventing the addition of new duplicates we can begin to tackle this issue and begin reducing the storage requirements and costs of running such systems. Aside from just reducing storage overhead it can also be possible to reduce time overhead of searching for images and the amount of bandwidth needed to operate these systems. In addition, a significant amount of money –upwards of $18 million annually can be saved in storage as a result of reduced space requirements. This figure doesn't even account for the reduced electrical costs per node or the reduced number of server nodes required following de-duplication of extreme cases of redundancy.

References

[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. http://php.net/manual/en/ref.image.php, 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] 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.

[10] 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].

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

[12] 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].

[13] 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].

[14] 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.