Automatically Generating Large Freely Available Image Datasets From the Web

Spencer Fonte
University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy, Colorado Springs, CO USA 80918
spencer.fonte@knights.ucf.edu

Abstract

Although there are a few standard datasets in the computer vision community, there are several issues with creating new more challenging datasets. Most of these issues stem from privacy and copyright concerns. This project extends on work done by Mears [1] to develop a new paradigm for collecting and sharing image datasets. In this paradigm, only links to online images are shared using image feeds. Filters can be created and used to produce a new feed that is a subset of an already existing feed, allowing for the easy creation of a specific dataset by using an existing broader dataset feed or the cleaning up of a feed generated by a web crawler. The system consists of three main parts: a dataset feed generator, a feed subscriber, and a contest engine which will allow computer vision contests to be participated in in real time. Architectures for all three parts are provided in this paper and the first two have been implemented. The framework presented in this paper aids in the creation of new computer vision datasets that contain a large number of images, are more representative of the real world, and are less subject to copyright and privacy issues.

1. Introduction

Computer Vision experiments require a large number of images for training and testing algorithms. Creating large datasets that are publicly available can be challenging due to privacy and copyright issues. Most current public datasets are staged photos taken for the purpose of creating a dataset [2, 3, 4, 5, 6]. The images in these datasets do not reflect most of the images people encounter in the digital world today. Efforts have been made to use images found on the web to construct datasets, an example being Labeled Faces in the Wild [7] which is a dataset of facial images of famous people collected from the web. This type of dataset provides images for facial recognition tasks that are more analogous with average facial images taken in the real world. An example image of Tim Allen from Labeled Faces in the Wild is shown in Figure 1. However, there is only an average of around 2.3 images for each person in the dataset. This can be an issue because machine learning algorithms require a lot of data to train. This project seeks to create a paradigm and tool set that allows for the easy creation of large datasets from the web.

Figure 1. Example Image From Labeled Faces in the Wild

Recent work exploring dataset bias [8] highlights that in dataset competitions, algorithms often over-adapt to the peculiarities of a dataset and lose their generality. This work experiments with training and testing on different datasets for object recognition and shows a significant drop in performance. They also discuss the problem with computer vision datasets not being representative of the real world. Our work solves these problems by generating living and breathing datasets from the web. Not only will this prevent over-adapting to datasets, but these images will be representative of the real world.

This project extends work done by Mears [1] and aims to create a system that generates large datasets of images from the web while avoiding privacy and copyright issues. To avoid copyright and privacy issues, the datasets will not be composed of image data but instead of links that point to freely available images on the internet.

A large mass of images without any organization would not be very useful to researchers, so the system must provide a way to clean up the data. This will be done by allowing anyone to be able to create a filter. A filter will take in any existing stream and will output a subset of it as a new stream. Streams will be able to be filtered based on characteristics of the meta data, the image itself, or both the image and meta data.

The system will consist of three main parts; a feed generator, a feed subscriber, and a contest engine. To create these three main parts, a web crawler, downloader, filter creator, feed generator, database, and interface between Matlab, Python, or any other language that a researcher writes code in and the system are necessary. Some of these parts have already been created by Mears [1]. This includes the web crawler, downloader, and some filters. Mears never got a fully functional system working, and we rewrite some of the things he has created. In subsequent sections we will discuss what we have modified and have added to create a functioning system and also what extra features and ideas will be explored once the project in complete. Figure 2 shows a use case diagram for the system.

2. Previous Work and Differences

Although this is a continuation of Mears' work there will be significant differences. The system we will create will be more general and modular and will be more of a framework that will allow researchers to create any type of dataset they would like.

2.1. Web Crawler and Downloader

Mears [1] had modified the Heritrix3 web crawler [9] to gather links and alt text from images on the web along with the website title. He had also created a database to store the information in and then has written a downloader to download the images in order to analyze them. We will keep the project in one language, Python. We wrote a webcrawler from scratch instead of modifying and existing web crawler. This allows our web crawler to be easily customized and used for creating any time of dataset.

2.2. Filters

Mears had implemented a way to detect and remove duplications using a hashing method from [10] [11]. He also used the OpenCV [12] version of the Viola-Jones face detector [13]. He also had found and interesting way to detect logos an such from [14] [15]. Instead of creating filters for specific tasks, we create a filter template that allows for the easy generation of new filters. We also provide examples of filters.

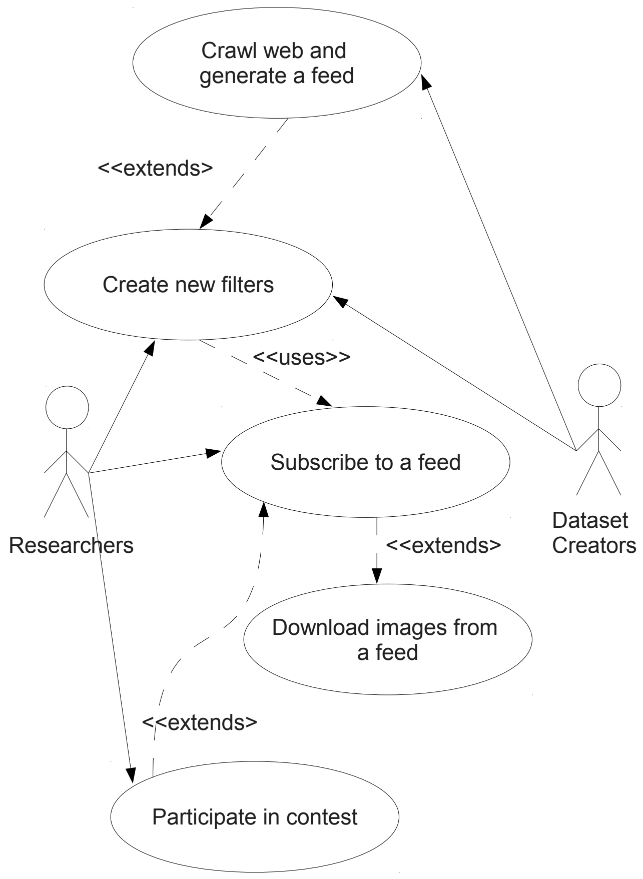

Figure 2. Use case diagram of the system

3. Architecture

As stated previously the system is composed of three main parts. Below they will be discussed in detail.

3.1. Dataset Generator

The architecture for a typical dataset generator is shown in Figure 3. It is composed of two parts, a crawler and a filter. It takes as input a list of seed websites for the web crawler, with this list the web crawler will crawl the web and output the URL and meta data for every image it finds. The format of this output is a custom comma separated value feed.

It is unlikely that a dataset creator will want a dataset of all images found on the web. Thus in the typical case a filter will be used to prune the feed generated by the crawler. A filter could prune results based on any criterion on the image or meta data. For example a dataset creator may want to only include images

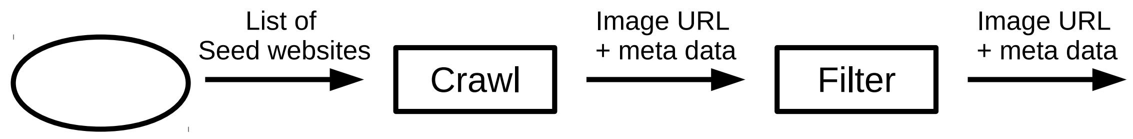

Figure 3. Architecture of Dataset Generator

in which the alt text from the image included the word 'rabbit'.

A more complicated example could be that a dataset creator only wants images of faces. They may choose to accomplish this by creating a filter that downloads images from the web crawler feed then running a face detector on those images with a low confidence threshold. The feed outputted from this filter would then contain images of faces and also a lot of false positives. The dataset creators could then create another filter that would take in the feed outputted by the previously described filter and would then create a new filter that utilizes Amazon Mechanical Turk and only outputs a feed of image URLs and meta data of verified images of faces.

As one can see, the architecture of the system is designed in a way that it is very flexible. In the typical case a web crawler will provide the first feed, but this does not have to be the case. As long as there is a URL to access the image a dataset creator could generate a feed from any source. Also it should be observed that filters can be chained together and be complex. We provide a template to generate simple filters but researchers may choose to create very complex filters on their own, like the Amazon Mechanical Turk example discussed above. Remember a filter just takes in a feed and outputs a feed that is a subset of its input.

3.2. Feed Subscriber

The architecture for the Feed Subscriber is shown in Figure 4. It takes in a URL of a feed that is generated by a Dataset Generator. Feeds are just a standardized file format the first step is to parse the feed file. Once it is parsed any new image information in the feed will then be sent to a module that checks a local database. If the image and its meta data are not in the database, then the image will be downloaded and inserted into the database.

We plan on supporting common database management systems (DBMS). If the user does not want to use a DBMS, then SQLite will be used. Since the system will be very modular if the user wants to use some other non-supported storage system they can just modify the modules that check the database and insert into the database.

Once a user subscribes to a feed they will start downloading the dataset. There could potentially be an issue if multiple research groups want to compare their algorithms and they all subscribed at different times. It is our hope that since disk space is cheap, researchers will subscribe early to feeds that they have a potential interest in. If it is the case that research groups wishing to compare algorithms have subscribed at different times a more recent subset of the feed can be used.

3.3. Contest Engine

Figure 5 shows the architecture of the user side of the Contest Engine. The Contest Engine allows computer vision contests to be preformed in real time. Previous to the contest, the contest host can provide a training feed which would include ground truth. Then during the contest, the contest host would provide a test feed. The research groups participating in the contest would use the Contest Engine which will first subscribe to the test feed then check their local database for the image, download it if necessary, perform their algorithm on the image, and finally output their results as a feed.

The motivation behind the contest engine is to prevent participants in a contest from over adapting to the dataset. Sometimes participants in contests will have algorithms that specifically adapted to the dataset being used. This does not help push the area of the contest to be better as the winning algorithm may be very good on the dataset being used in the contest but perform poorly in general. The contest engine will allow contest to be carried out and since the dataset is live from the web and constantly changing participants must solve the general problem at hand.

We describe the architecture for the contest engine above but we did not implement it. This would be great future work.

4. Implementation

This project aims to be cross platform and to be easy to modify. All of the components are written in Python. The project uses some open source libraries

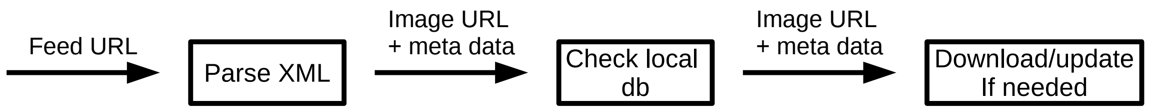

Figure 4. Architecture of Feed Subscriber

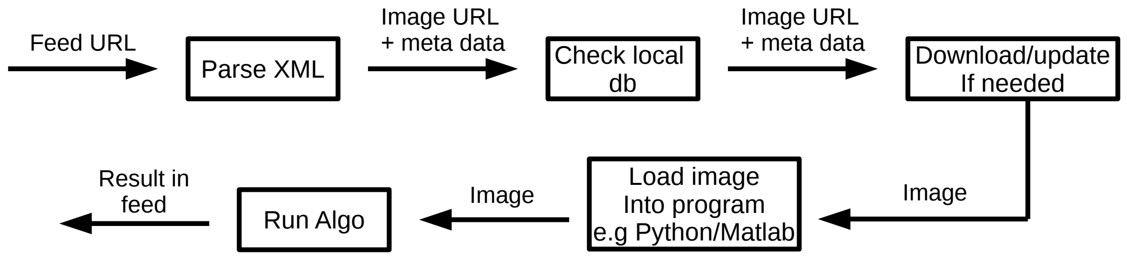

Figure 5. Architecture of Contest Engine

and these will be bundled with the project.

4.1. Dataset Generator

4.1.1 Web Crawler

The dataset generator is organized into a Python script for the web crawler and another script for the filter. The web crawler script utilizes a feed class. The web crawler begins by putting one or more “seed” URLs into a pool. Then a URL is selected from the pool and its content is downloaded and parsed for links. The found links are then added to the URL pool. Beautiful Soup is used to find all the images that the web page refers to. Relative links and image references are both made absolute. Every image link found along with the URL of the web page it was found on, its alt text, and the current time are formed into a tuple which is then added to a list in the feed class. After visiting a web page the crawler calls a publish method on the feed class. This outputs the contents of the feed to a file as comma separated values.

The web crawler does not strive to be extremely fast. It is not multi-threaded, it even pauses for a few seconds after it visits each site. We do not strive for speed in the web crawler because it generates a feed which subscribers must read, parse, download images, and process images from. If the web crawler performs at a significantly quicker rate than the subscribers the amount of ”new” unchecked items in the feed will increase. A result of this is that the time between the web crawler placing an image URL in the feed and the time that URL is checked by the subscriber can be a significantly long time, this can result in links no longer be active or accurate which is undesirable.

4.1.2 Feed Format

The feed is contained in a custom comma separated value file. The first line of the feed file contains the Unix time for the last update to the feed. This allows the file to be quickly checked for updates as only one line needs to be read. The second line contains the name of the feed. The third line contains the URL where the feed is located. The fourth line contains a short description of the feed. Then there is a blank line and the sixth line contains the names of the fields for each entry separated by commas, the first entry is always the date of the entry. Each line after that contains a line for each entry in the feeds. There is an entry for every image reference. Bellow is a sample of a feed.

```
1311712160
Bikes vs. Mobiles
http://www.anexample.com
Bikes and mobile Phone pictures from Craigslist

date pub,img url,site linked from,alt text
```

Then the feed would contain all the entries on separate lines. These are too long with all the URLs to show an example for in a sensible way.

4.1.3 Filters

Filters must subscribe to an existing feed in order to filter its results. This may be a feed being used for another dataset or the feed being generated by the web crawler. To subscribe to a feed the filter framework periodically opens the feed file of the feed it is subscribing to. It checks the first line of the feed file which contains the time the file was last updated. If this time is greater than the time of the last check the filter framework subscriber will parse and process the feed file line by line until it reaches the entries it has already processed.

The filters themselves are implemented in a functional manner. Each filter is a function. These functions can be chained together by having one filter call another one. Every time a new entry is parsed from the feed, the first function in the filter chain is called and is passed the data for the entry. To create filters easily Open CV or the Python Imaging Library can be imported and their functionality can be used within the filter functions.

4.2. Subscriber

The subscriber periodically checks a feed for new image URLs and if there are new image URLs it will download them and store their meta data in a database. The subscriber is implemented in Python like the rest of the project. In the same way the filter subscribes to feeds the subscriber only needs to check the first line of the feed file in order to determine if any changes have been made. If there have been updates the subscriber only reads the file until it has read all the new entries it has not yet read before. The database being used currently in a Sqlite3 database. The Python library is used to create and interact with the database file. The images are downloaded into a standard directory and the paths to the images are stored in the database with the other meta data related to the image and the website it was found on. This allows quick querying on the image meta data and then a simply using the path to retrieve the image data if it is desired.

5. Observations

To test the current system, which consists of a web crawler that generates feeds and a filter which subscribes to a feed and filters the results, we inserted code into the filter code to display the output of the filter. We first used a filter that simply outputs everything from the feed it reads. When running this filter on the output from the web crawler it is clear how messy the images on the web are. A significant proportion of the images found are styling elements of websites, this includes logos, images for tool bars, and one-by-one pixel images used to make shapes of one color. Some examples are show in Figure 6.

Figure 6. Examples of logos found when crawling the web

We begin experimenting with other simple filters. We observed that a filter which only keeps images in which their size in each dimension is over a certain threshold is very effective at eliminating logos and other images for styling websites. However we want to reiterate that the system does and will not filter out these images by default. Although we will include this filter to be used, some people may want to create datasets that include these images. Our system is made to be flexible and easily modifiable.

On a small set of 434 websites 20418 images are referred to. This averages to 47 images per site. However most of these images are just logos and other features used to style web pages, and also avatars. The histogram in Figure 7 shows the minimum dimension size for each image found.

6. Testing and Applications

In order to test the system and demonstrate its potential uses we have crawled the web to create feeds, filtered these feeds, and subscribed to these feeds. We started out with choosing a seed and generating a feed based on the web crawler unrestrictedly crawling, and then using a basic filter to filter our small images as discussed in the previous section.

6.1. Craiglist: Bikes vs. Mobile Phones Dataset

To demonstrate how customizable the system is and to show a potential application we limited the web crawler to Craigslist classifieds web pages for bikes and mobile phones. Then we subscribed to this feed and downloaded the images and meta data. The result of this is a simple example dataset that could be used for object detection of mobile phones and bicycles. We let the subscriber download images for a few hours and the number of images acquired is shown in the following table.

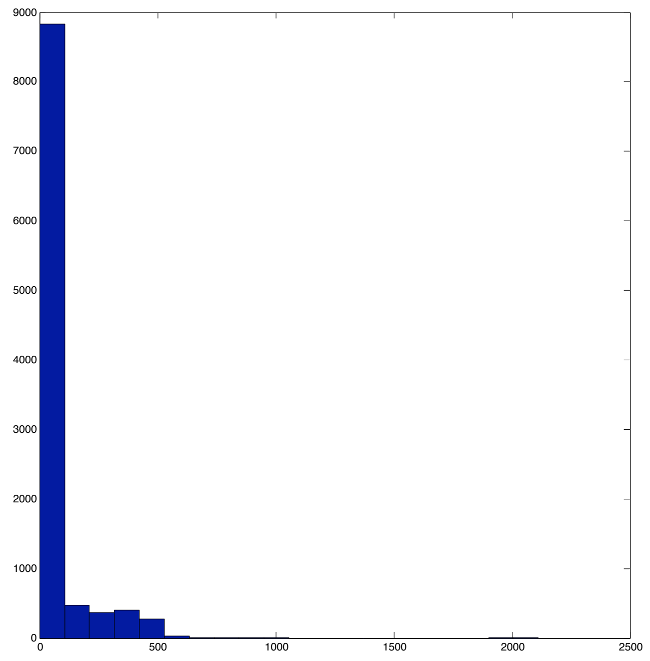

Figure 7. Histogram showing the minimum dimension size for each image

	Bikes	Mobile Phones
Unique images with files	15717	11137
All unique images	16040	11356
Total entries in database	31578	22493

In the table above the first row shows the number of actual unique image files downloaded that we also gathered meta data for. The second row shows the number of unique images that were entered into the feed but that the subscriber failed to download, we end up with the meta data but no image file for these images. The last row in the table is the number of entries of meta data that were entered into the database from the subscriber.

The reason for such a large discrepancy is that the feed generator has an entry for each unique pair of images URL and the URL of the web page it was referred to from. It is not uncommon for the same image to be used on multiple web pages. This is especially true of logos that appear on every web page within a web site.

Figure 8 shows some images that were collected as part of this example. It is clear that they are more representative of images of mobile phones and bicycles that people generally encounter.:

6.2. Future Work

In the future this project can be extended in several ways. Although a basic architecture was described in this paper the contest engine has not yet been implemented. This would be a good first extension to this work.

In this paper we show that the described and implemented framework can be used to create datasets from Craigslist. In the future more example datasets should be generated from the framework, this would help test the framework and also make researchers more likely to use it for dataset generation and acquisition in the future. This is important since multiple research groups must be convinced to subscribe to a dataset feed for the feed to be useful. Ebay would be a good contender for testing out the framework, it has very well defined categories, is always being updates, and has a large number of images. We believe it would be a good resource for creating a object recognition dataset.

After the contest engine it should be used to hold a basic example contest. This will provide more information on the properties of datasets created using the system and will allow the differences between live datasets and old standard datasets to be explored.

Another extension of this project would be to add a graphical user interface. As of now the interface is command line based. This extension could make using the system more intuitive. A web based interface would also be worth exploring as it would be cross platform and allow for people to monitor and control the system off site with ease.

As discussed previously the system is not multi-threaded. Even though we do not view this as a large disadvantage, it could be explored. If the web crawler and feed generator were made mutli-threaded it would be necessary to make the filter framework and the subscriber to also be multi-threaded to avoid the problem of the subscriber being unable to keep up with the rate at which a feed was being added to.

We believe our feed format can handle hundreds of thousands of entries in a feed. To make the reading of feeds more efficient one could split the feed up into multiple files and create a directory file. Then when the feed is read first the directory file would be read and then the appropriate feed file would be read. This would prevent having files that are enormous.

References

[1] B. Mears, "Generating a Large, Freely-Available Dataset for Face-Related Algorithms." [Online]. Available: http://www.cs.uccs.edu/~kalita/work/reu/REUFinalPapers2010/Mears.pdf

[2] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested

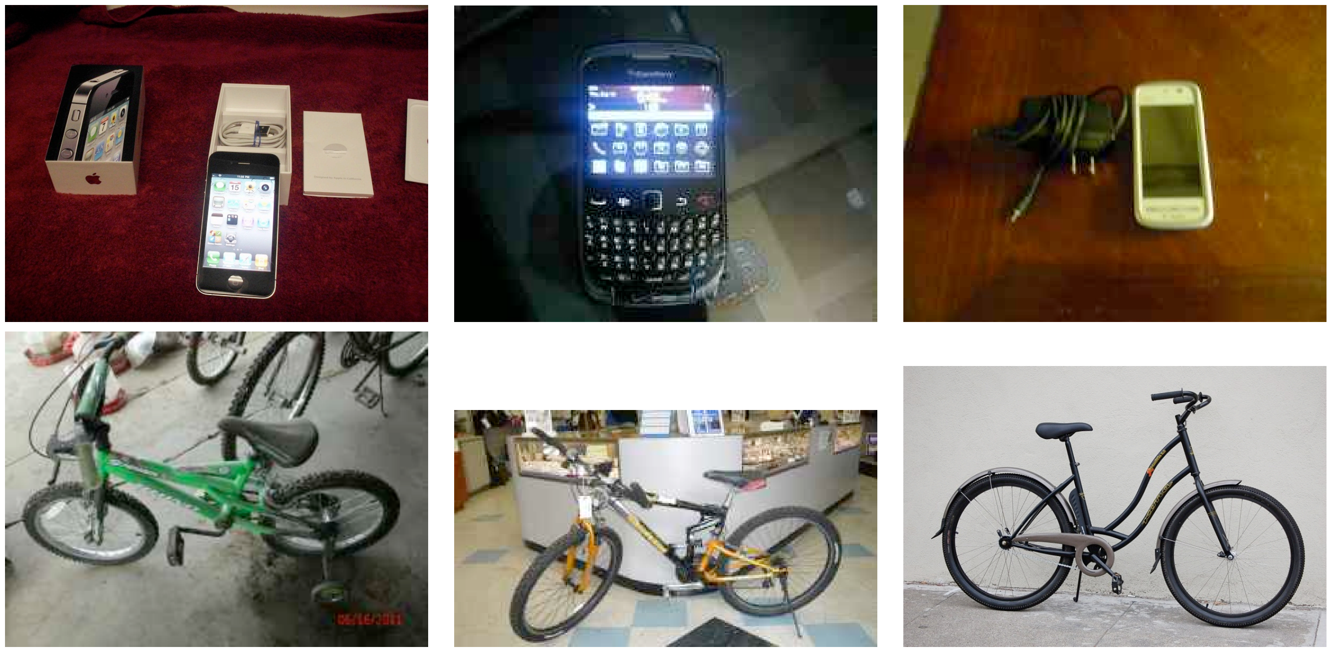

Figure 8. Example images from our bikes vs. mobile phones dataset

on 101 object categories," *Computer Vision and Image Understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[3] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[4] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "The CMU multi-pose, illumination, and expression (Multi-PIE) face database," Technical report, Robotics Institute, Carnegie Mellon University, 2007. TR-07-08, Tech. Rep.

[5] T. Sim, S. Baker, and M. Bsat, "The CMU pose, illumination, and expression database," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1615–1618, 2003.

[6] P. Phillips, H. Moon, P. Rauss, and S. Rizvi, "The FERET evaluation methodology for face-recognition algorithms," in *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997. Proceedings.*, 1997, pp. 137–143.

[7] G. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," *University of Massachusetts, Amherst, Technical Report*, vol. 57, no. 2, pp. 07–49, 2007.

[8] A. Torralba and A. Efros, "Unbiased Look at Dataset Bias," in *Proc. IEEE CVPR 2011.*

[9] K. Sigurdwwon, M. Stack, and I. Ranitovic. Heritrix user manual. Internet Archive. [Online]. Available: http://crawler.archive.org/articles/user_manual/index.html

[10] S. Xiang, H. Kim, and J. Huang, "Histogram-based image hashing scheme robust against geometric deformations," in *Proceedings of the 9th workshop on Multimedia & security.* ACM, 2007, p. 128.

[11] M. Mıhçak and R. Venkatesan, "New iterative geometric methods for robust perceptual image hashing," *Security and Privacy in Digital Rights Management*, pp. 13–21.

[12] G. Bradski, "The OpenCV Library–An open-source library for processing image data," *Dr. Dobbs Journal*, pp. 120–125, 2000.

[13] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple," in *Proc. IEEE CVPR 2001.* Citeseer.

[14] A. Hartmann, "Classifying images on the web automatically," *Journal of Electronic Imaging*, vol. 11, no. 4, pp. 1–0, 2002.

[15] V. Athitsos, M. Swain, and C. Frankel, "Distinguishing photographs and graphics on the world wide web," in *IEEE Workshop on Content-Based Access of Image and Video Libraries, 1997. Proceedings*, 1997, pp. 10–17.