System for Automatic Development of a Visual Attribute Vocabulary

Ankush Gulati
Stony Brook University, NY
angulati@cs.stonybrook.edu

Prasad Prabhu
Stony Brook University, NY
prprabhu@cs.stonybrook.edu

(*In fulfillment of CSE 523 Master's Advanced Project*)

ABSTRACT

The goal of this project is to design an automated system that develops a dictionary of image attributes based on human responses in addition to low level image information. The current implementation automatically harvests images of objects or scenes specified by textual descriptions from the web and further acquires detailed attributes describing these objects and scenes using Amazon's Mechanical Turk to verify image contents. The project report will progress by discussing the goal of project, motivation, exploring certain challenges, why do we need automation, current implementation, entire life cycle of obtaining image tags and future work.

KEYWORDS

images, tags, mechanical turk, crawler, attributes, human response.

INTRODUCTION

The overview of the image tagging project -

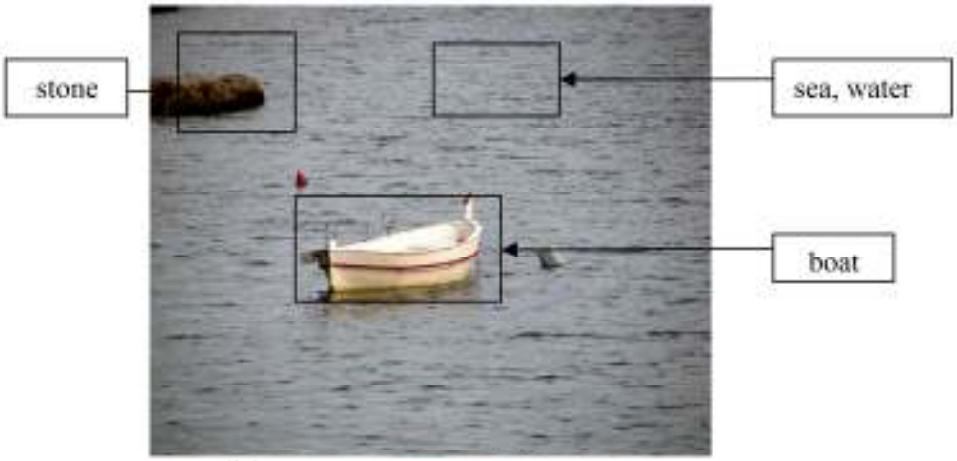
Image Tagging

Most machines are able to process textual queries easily but searching images based on content is a non-trivial task. Thus, a mechanism to translate image to text is needed for efficient querying on images. The description of image could be anything, a long prose or might contain keyword that describe spatial, temporal or emotional aspects. Popular applications that take advantage of such image tagging are Flickr, Google ImageLabeler, Adobe Photoshop Album, etc.

Automated Image Tagging

Automatic image tagging seeks to assign meaningful words to images (e.g book, animal, moon, etc) and describe contents found in those images without manual intervention. The idea works on finding template patterns on training set and associate it with set of words or tags. In order to obtain such templates, cataloging of relevant images along with human responses are necessary. If patterns are observed in human responses for images, it could alleviate some challenges in image tagging. The aim of project is to make entire process automated with minimum efforts. This may encourage further research in image tagging and reduces set up time.

Here is an instance of human tagging of images,



Source: Flickr

Certain image tagging applications target certain qualities and attributes. This project aims to build a generic image tag application and encourage further research by providing easy and quick groundwork.

Issues in automated image tagging

Tagging based on image's low-level information such as image histogram do not scale up well to real world data. An extra level of information, such as a human annotation is necessary to understand the information with better accuracy.

For instance:

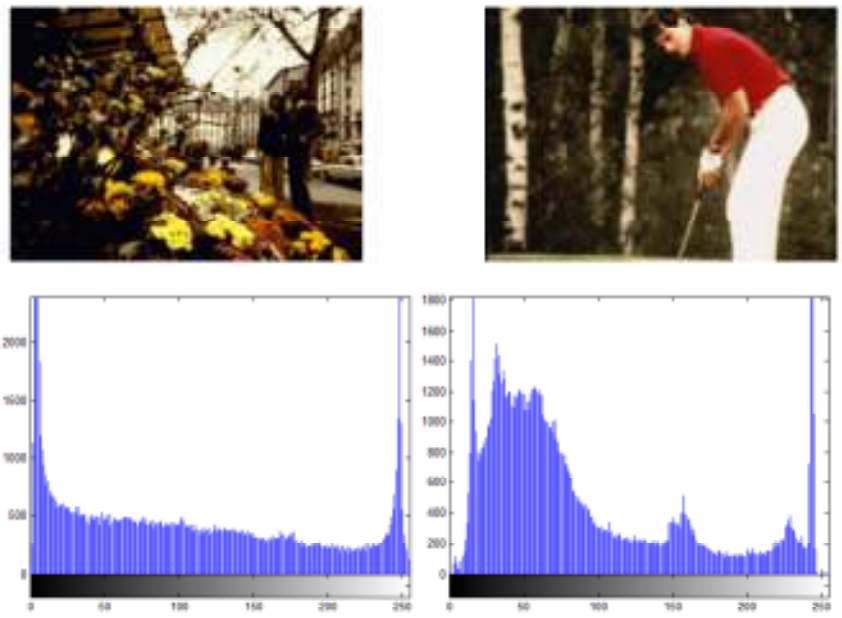


Fig: Images and their gray-level histograms. (top image) Source (1)

As it is obvious from above images and their histograms, the image content above are totally out of context, however their histogram information does not give us a convincing difference. Human annotators

alleviate the problem in resolving such ambiguities. Tags such as golf, flowers would definitely help to identify such two disparate images .

Automated techniques depends primarily on correctness in human responses. However, such responses are bound to human errors and inconsistencies. Image tags are susceptible to one's own interpretation and common sense. Certain images consist of lot of information and by no means can be described completely by few set of words. Certain tags are irrelevant to the image and sometimes tag that are too detailed or too specific are of less use. The major issue is the order of tags. Given a set of tags, it is non-trivial to conclude as to which tag describes which part of image.

Crowdsourcing

We are creating a catalogue of relevant images and their respective tags. Specialized crawler is built to harvest images from web that we desire. Later, images are tagged using third-party programming techniques such as Amazon Mechanical Turk, which allows us to submit image tagging jobs called as HIT(Human Intelligence Task) to online unknown workers. A HIT is a simple non-intelligible task that expects inputs from unknown human worker. We as a requester, create image tagging templates that displays set of images and place holders(3 in this case) for providing image tags using mturk tools. The entire process from automatic HIT creation as well as harvesting images from web is described in the further sections.

IMPLEMENTATION

Initial set-up

The initial part of project consists of programming python scripts that is easy for set-up and that requires minimum efforts in maintaining results. The entire arrangement needs a back-end server (PHP v5.0+) that manages the external source question and a database(MySQL) to maintain results and statistics. We chose database over file system to leverage efficient and simpler querying.

Automated H.I.T Creation

In order to use the script, the requestor needs to maintain the list of images urls in a directory relative to its template. The template file can be stored anywhere in public_html directory. Following command needs to be executed to create a HIT.

```
python create_hit.py [-u base_url] [-t template_name] [-n hit_name]
                    [-l image_source_list] {OPTIONAL[-o output_dir_name] }
```

E.g.:

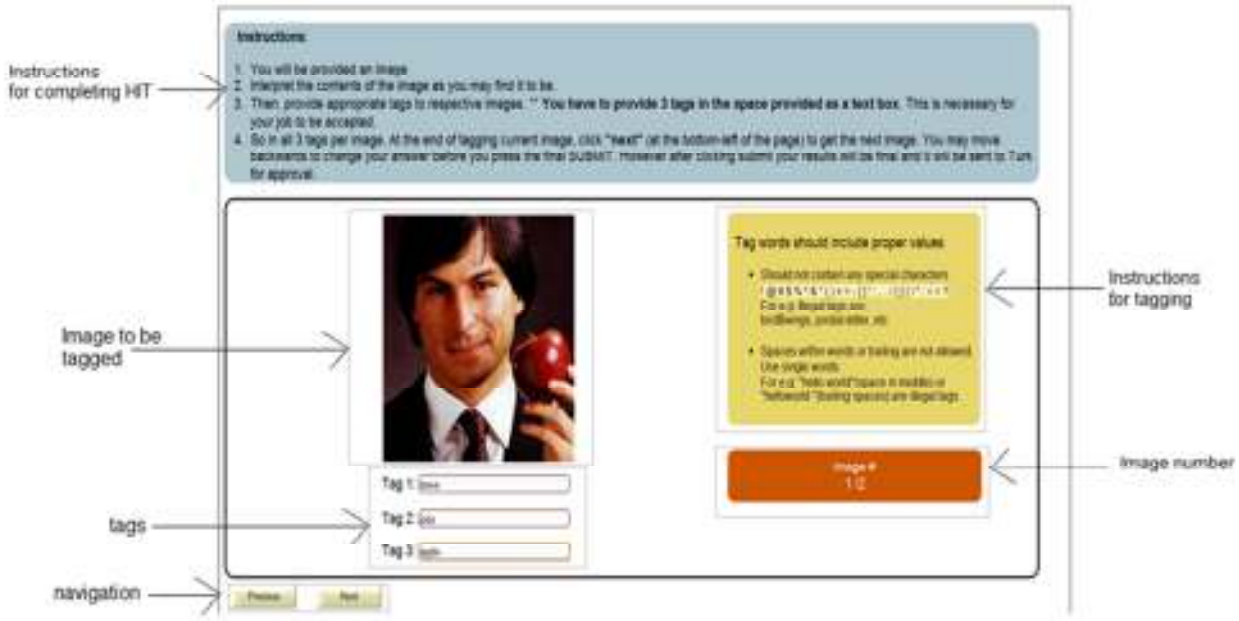
```
python create_hit.py -u   recognition.cs.stonybrook.edu/~ykou/
                     -t   template_dir/tag.php
                     -n   TagMeHitFolder
                     -l   job_directory/image_source_list.txt
                     -o   TagMeHitOutput
```

The *create_hit.py* script will generate a HIT *TagMeHitFolder* that contains all the Amazon command line tools to run the HIT(run.sh), retrieve/approve *(getResults.sh/ approveAndDeleteResults.sh*) results. *Tag.php* is the template file that serves as the external URL for the mechanical turk. The script will copy

image_source_list.txt to template_dir and assign necessary file permissions. Template dependencies such as stylesheets, JavaScript source files, etc has to be manually managed by the requestor. Amazon provides sandbox to test the template. The argument variable -sandbox of the Amazon command line tool lets us test our template over mturk sandbox.
e.g. `run.sh -sandbox` will run the template file over turk and provide us the link that runs this url.
On retrieving results from Amazon, the requestor has the choice of approving or rejecting the answer and accordingly the workers are paid. In image tagging case, answers will be rejected if it contains non-dictionary words, same tags for images, etc.

HIT Template

A sample image tagging template would like as follows -



On successful submission of tags, workers will see such message.

Results

A subset of columns of results obtained from Amazon are -

Hitid	Assignmentid	Workerid	Answer.results
XXX	YYY	ZZZ	job/image01.jpg:tag1;tag2;tag3|job/image02.jpg:tag1;tag2;tag3|

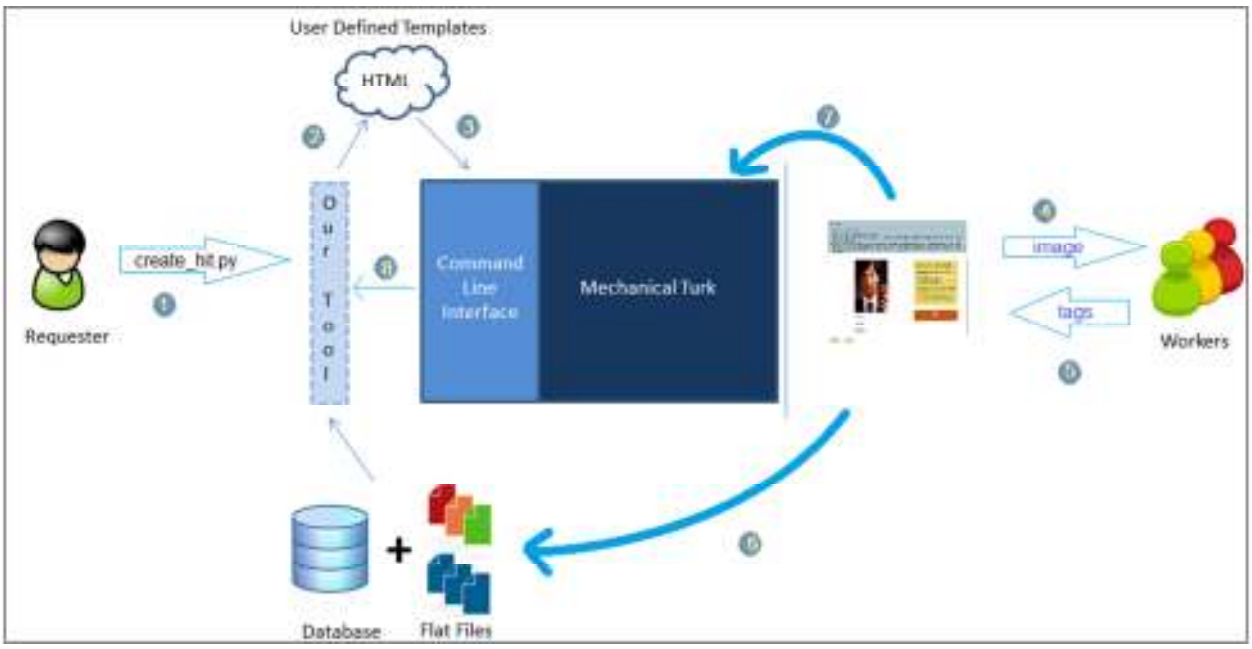
The results are for each of the images are '|' delimited and tags are ';' delimited.
Assignmentid are given for each worker assignments uniquely by Amazon that attempts to tag images.
HITid is same for all assignments.

ENTIRE PROCESS

A template file, (such as tag.php) should be scripted in PHP that will include images obtained from specialized crawler and accept 3 tags per image. The template should not allow any special character or space in the tags as it may be elusive to separate correct answers. On submission of answers, there should be correct set up of MySQL database such as database name, username, table name, table columns. The submit file combines all the images and their respective tags to Amazon turk and does a similar database insert at the client's end. The advantage - the client has the entire information about his mturk activities at a centralized location and is available for reuse as well as manipulation.

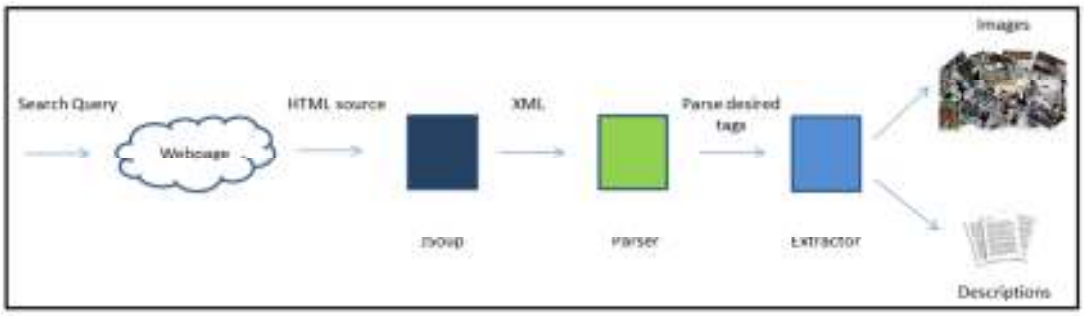
Initially, *create_hit.py* will generate turk tools and does necessary permission modifications on template folder, image source list, submit.php files. If more time (default 60mins) and reward is to be allotted for completion of HIT assignment, requestor has to modify *assignmentduration* and *reward* variables in .properties files. For adding more HITs, requestor has to add more lines to .input file in HIT folder. The line column will point to the corresponding line number in the image source list. Every line in image source list has a list of image urls stored horizontally and space delimited. Also, every line in the HIT point to different HITs. Every line should contain at least one image but has no upper limit on number of images per HIT.

On running the tool, template url is uploaded over Amazon, and returns back the HIT url. The HIT is now available to the worker for tagging. At the end of time frame(initialized in .properties) , results are available to the requestor. The requestor now has the choice to either approve or reject the workers depending upon the relevance of the results. If the requestor does not approve/reject within time period, Amazon by default approves the result.



Specialized Crawler

To build a large repository of images of scenes and objects, we need an automated system that searches the desired websites and extracts the relevant images for us. We have designed a website specific crawler that takes the input query & the URL and fetches all the images related to the query from the website by parsing the DOM structure of the HTML of each page.



We have currently been able to crawl www.like.com using the crawler. We plan to extend the crawler in the coming semester for other websites as well. The current implementation of the web crawler is in JAVA and uses an open source library JSoup for converting the DOM Structure of the source of crawled html page to a clean XML that can then be read using simple JAVA routines. It takes generic parameters like search query, URL (like.com) & the result directory path and writes the relevant images from the search result along with their descriptions (provided on like.com) to the desired directory.

FUTURE WORK

Specialized Crawler
The crawler is currently in the implementation phase. We plan to work more on it next semester to make it more robust and if possible generic (free from DOM structure restrictions of a webpage) so that any regular user can use it for crawling any image site.

Create the training data set
Once the crawler implementation is complete, we plan to crawl multiple image websites to populate our database and get them tagged from workers on Mechanical Turk using the already setup MTurk Automation Tool.

Training the Dataset using computer vision routines
Design computer vision algorithm to associate human annotations with images and form template patterns while learning from those annotations.

Test and analysis
Generate testing datasets from the web and analyze the efficiency of these results.

BIBLIOGRAPHY

1. *Histogram Refinement for Content-Based Image Retrieval.* **Greg Pass, Ramin Zabih.**

2. **Amazon Mechanical Turk Developer Command Line.**

http://docs.amazonwebservices.com/AWSMechTurk/latest/AWSMturkCLT/.

3. **Online Source Code and tutorial.** *http://recognition.cs.stonybrook.edu/~ykou/website/.*

4. **Flickr.** Image Sources. http://www.flickr.com

6. **JSoup**: Java HTML Parser *http://jsoup.org/apidocs/*