The Intelligent Image

Max Jaderberg

Keble College, University of Oxford

Trinity Term 2012

Abstract

In this report, I shall discuss some awesome stuff.

Chapter 1

Introduction

The web contains billions of images, which are often the focus of attention on the web pages that include them. However, there is almost no information about the content of these images. Images on websites are purely binary data files, occasionally with some associated meta data included in the image's HTML[1] code. Very little information is available about the image, let alone the objects or scenes contained within the image. The aim of this project is to create a system which automatically recognises the objects within these images, thus releasing the information within them. This is a large scale object retrieval problem.

There are a large number of applications which would benefit from having detailed information on the contents of images. With more knowledge on the objects contained within images, one can create more effective search engines, better cataloguing and classification systems, user interfaces which engage viewers more, and relevant advertising based on the content. Novel applications could also be built, for example, which retrospectively embed geographical information in the image binary by recognising where the image was photographed. The plethora of useful applications provides a great deal of motivation for this project.

The result of the project is that a query image is inputted, the objects contained within the image are recognised, and the image is returned with the recognised objects "tagged" i.e. the region of the object in the image is outlined and it can be clicked to give relevant information. In this way the standard image has been transformed into the "intelligent image" - one which knows about the contained scene and can offer up information to the consumer about it's contents.

[1] Hyper Text Markup Language `http://en.wikipedia.org/wiki/HTML`

Figure 1.1: An example "Intelligent Image" showing the various hover over states

Recognising a wide range of objects in an image is not a trivial problem. For such a system to be useful the following needed to be addressed:

- Acquire and filter enough data to create a model to perform matching on a large range of objects.
- Create a retrieval system which provides accurate matching. False positive matching needs to be avoided as much as possible.
- Ensure matching can be done quickly on a database of millions of objects.

To recognise millions of different objects, reference images are needed to create a model to match against. From the outset, Wikipedia[2] is used as the primary model data source. Wikipedia is a crowd-sourced online encyclopaedia with many images contained within the articles, and is considered an accurate source of information. In the system, each Wikipedia page defines an object, with the images in the article being used to provide the data to match against. The content of the Wikipedia article is used to give the user further information about the object. A web crawler was written to extract the relevant images of desired Wikipedia pages and create the image database. Filtering is also performed to ensure only useful images are included in the database. The data sources are fully explained in Chapter 3.

Object retrieval uses a method employing a bag-of-words model. This builds upon the work described in x and y, more information of which is provided in Chapter 2. The visual words in the image database from Wikipedia are precomputed. At run time, the words in the query image are computed and searched against the database of precomputed words to find the top image matches. The top matches are spatially verified, with the first verified image being the result. Subsequent improvements to the base line system

[2] `http://en.wikipedia.org`

(described fully in Chapter 4) were made, including geometric improvements to the spatial verification and descriptors used to increase speed and accuracy of spatial verification, and matching improvements through query expansion using crowd-sourced data (dubbed "Turbo-boosting") from Microsoft's Bing[3] search engine. The geometric improvements and turbo-boosting is reported in Chapter 6 and Chapter 7 respectively.

Finally a website was developed to provide a front end interface for the system. A user can simply navigate to the website and upload an image. They then start the automatic tagging process, during which a realtime log of the process is displayed. After the query is complete, the intelligent image is displayed, which the user can interact with, see the names of the objects contained within the image, and click on the object to go to it's Wikipedia page. The software architecture of the website and the backend systems is explored in Chapter 5. The result is a realtime automatic tagging system which could recognise, for example, all the buildings in a tourist's photo album of London in seconds.

The aim of the project is to work towards recognising every object on Wikipedia, however due to time restraints a subset of objects was used for development and testing of the project. The subset chosen were the pages that appear on the Wikipedia page "List of Structures in London"[4].

[3] `http://www.bing.com`

[4] `http://en.wikipedia.org/wiki/List_of_structures_in_London`

Chapter 2

Background

Chapter 3

Data

There are three main datasets used by the application: the images from Wikipedia used to build the database of objects (Section 3.1), the images from Microsoft Bing used for the Turbo-boosting (Section 3.2) and the images from Google Images used for validation and testing (Section 3.3). All images that are used are resized so that their larger dimension does not exceed 1000 pixels to reduce storage space. This chapter describes the various datasets and how they are acquired.

3.1 Model Images

The model comprises of a dataset of images which depict the objects that are to be able to be recognised. To enable an object to be recognised, the object must be represented by one or more images contained within the model. For example, the object "Tower Bridge" will be best represented by multiple images of Tower Bridge taken at different angles and focussing on different features. Wikipedia was used as the source of the images to represent the objects described by the Wikipedia pages. The resulting representations work well - an example object representation is shown in Figure 3.1.

Each page of Wikipedia that contains images represents an object which can be matched. The database of images which is used to build the model is simply created by visiting each page on Wikipedia for the objects desired and downloading the relevant images contained on the web page, labelling that image as being associated with the object. A script automates this process of building the model database.

Figure 3.1: The representation of the object "Tower Bridge" by images

Figure 3.2: The Wikipedia page for "Tower Bridge". Note the images contained are those used in the model to represent this object.

To automate the downloading of object images from Wikipedia, Python[1] is used. WIkipedia offers a public application programming interface (API) over HTTP as it is built on the MediaWiki framework[2], however it is cumbersome and not easy to consume. Instead, a web crawler was written to explore Wikipedia pages and extract the relevant images.

A crawler object in Python finds all the images and notes the urls of them for subsequent download. Firstly, the HTML of the Wikipedia page must be downloaded, as it appears to a web browser (an example of a web browser being Google Chrome). However, Wikipedia does not allow crawlers and automated bots to access its web pages. To overcome this, the HTTP header[3] of the crawler is edited to emulate that of a browser. This is implemented using the urllib2 library[4]. The code shown in Listing 3.1 shows an example of how to read the HTML of the main Wikipedia homepage. The HTML document for each Wikipedia

[1] `http://www.python.org`

[2] The MediaWiki framework was originally developed for Wikipedia and provides an API over HTTP as standard. `http://www.mediawiki.org/wiki/API` provides documentation for the API.

[3] `http://www.w3.org/Protocols/rfc2616/rfc2616.html` describes the Hyper Text Transport Protocol and the various header fields.

[4] `http://docs.python.org/library/urllib2.html`

page is parsed using the BeautifulSoup library[5]. All the anchor elements are found and stored for further crawling. The images contained within the HTML are also found by looking within the part of the HTML document that is unique to the specific Wikipedia page (see Listing 3.2).

Listing 3.1: The code used to gain access to Wikipedia's content using a crawler by emulating a browser.

```
### Python
import urllib2
# Emulate the user agent as that of a browser
user_agent = "Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.1.7)
    Gecko/2007091417 Firefox/2.0.0.7"
headers = {"User-Agent": user_agent}
# Request the webpage
req = urllib2.Request("http://en.wikipedia.org", headers=headers)
resp=urllib2.urlopen(req)
# Read the HTML of the response
html = resp.read()
```

Listing 3.2: Parsing the Wikipedia article HTML document to find the relevent images.

```
### Python
def _get_content_body(self, soup):
        main_content = soup.find('div', {"class": "mw-content-ltr"})
        if main_content is None:
            return None
        # remove navboxes
        navboxes = main_content.findAll('table', {'class': 'navbox'})
        [navbox.extract() for navbox in navboxes]
        return main_content

def _get_image_links(self, soup):
```

[5] http://www.crummy.com/software/BeautifulSoup/

```
        return soup.findAll('a', {'class': 'image'})


# Parse the HTML
soup = BeautifulSoup(html)
# Get the main article body
soup = _get_content_body(soup)
# Get a list of image links
image_links = _get_image_links(soup)
```

The output of the crawler is a CSV file of the image urls and the object class the images belong to. The object class is simply named from the url of the Wikipedia page (for example all images appearing on `http://en.wikipedia.org/wiki/Tower_Bridge` will have class `Tower_Bridge`).

The images mentioned in the CSV file produced by the crawler are then downloaded to local storage. Each image that appears on Wikipedia has a "file page" which displays the image along with properties and metadata on the image[6]. This "file page" is visited for each image, and the page is parsed to extract the storage url of the image as well as its file format and original size. As the application resizes all images that exceed 1000 pixels to 1000 pixels, it is a waste of time and storage space to download the original image and later resize it. Instead, Wikipedia's inbuilt thumbnail engine is exploited, which resizes the image on Wikipedia's servers and allows you to download a thumbnail of a user selected width[7]. The thumbnail images are downloaded and are saved in a folder named after it's class. Therefore if the original image on Wikipedia exceeds 1000 pixels, the 1000 pixel thumbnail version is downloaded instead. The result is a directory containing a folder for each class, within which are the images for that class.

This process creates a structured dataset of model images from the Wikipedia pages visited. For the List of Structures in London dataset used, there were 701 classes which had 3963 images associated with them.

[6] For an example see `http://en.wikipedia.org/wiki/File:Tower_bridge_London_Twilight_-_November_2006.jpg`

[7] `http://www.algorithm.co.il/blogs/programming/wikipedia-images/` describes how this is exploited

3.2 Turbo-boosting Images

3.3 Validation Images

Chapter 4

Base line system

Chapter 5

Software architecture

Chapter 6

Geometric Improvements

Chapter 7

Turbo-boosting

Chapter 8

Summary

Bibliography

[1] P. Turcot and D. G. Lowe. Better matching with fewer features: The selection of useful features in large database recognition problems In *WS-LAVD, ICCV*, 2009.