Detecting locations from images utilizing supervised and unsupervised learning techniques.

Camilo Correa Restrepo*

I. Introduction

The aim of this project was, simply put, to construct a system capable of performing image processing on images uploaded to a remote server, in order to determine what physical location that image corresponds to. This document, then, aims to present the methodology followed as the software was developed, and aims to cast light on the design decisions that went into the development of the system.

II. Design & Architecture

A very minimalist design philosophy is present throughout the project, in particular, all elements are as simple as can be, in order to simply serve the core functionality to the user, without any embellishment. To that end, the *Flask* framework was chosen as the core piece of the design architecture. This was done given that it is simple to develop and deploy. Several software components were chosen to provide some much needed functionalities:

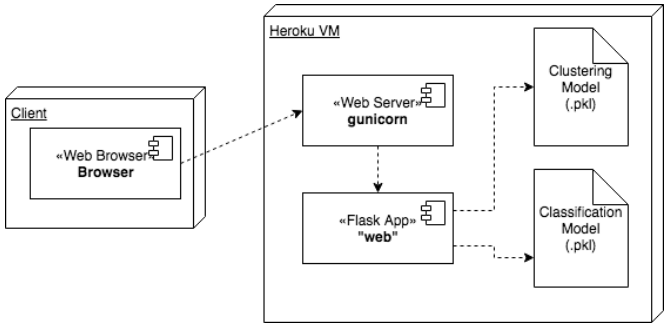
- OpenCV: Image processing, feature extraction.
- Numpy: Numeric computations.
- scikit-learn: Supervised and unsupervised learning algorithms.
- pandas: Data Analysis
- dask: Distributed processing for large scale datasets.
- joblib: Model persistence.

III. Structure of the Application

The application then, will be split into two distinct components:

- Model Creation Scripts: Here, a collection of scripts will take a sample dataset and prepare the models that will be used to classify the images in the main web application.
- Web Application: This web application has been designed to allow users to upload a photo of their current location and determine what building said photo corresponds to.

Now, the architecture for the web service is as follows:



Essentially, what the architecture pictured above shows is the fact that there will be two distinct learning models that will be employed. The way this is all set up to function is based on the Bag-of-words model, and, thus, to construct the 'visual words' we will need the first of these models. This will be a clustering algorithm, that will transform the several thousand feature vectors that a 'local feature extraction algorithm' such as Scale-invariant feature transform (SIFT) generates, into a far more manageable histogram. The frequencies in this histogram will thus represent the occurrences of these 'visual words', where each image feature will correspond to a certain cluster, and thus, the number of matches to each cluster will be the frequency reflected in the histogram. The second model that will be required will then classify said histogram, having been trained on a sizable amount of images of the different locations of interest. In the following sections we will examine how these models are constructed. Later, we will look at how the models are then fit into and deployed with the web application.

IV. Preprocessing the data

Both before work began and during the development phase, the images that were to be used to train the models were manually checked, in order to remove any samples in incorrect folders, and images that captured the place of interest inadequately. Through this process a sizable amount of the images were eliminated, given that they were unsuitable or unrepresentative.

V. Feature Selection & Extraction

The SIFT algorithm has been chosen as the primary algorithm for extracting the *visual features* that exist within a given image. This robust algorithm extracts several thousand local image features which, for the purposes of later manipulation, are exported into a *csv* file that basically represents every image through its features.

*Estudiante, Ingeniería de Sistemas Universidad EAFIT, código: 201410046010, correo: ccorre20@eafit.edu.co

VI. First Model: Clustering

In order to create an adequate set of feature vectors to train a classification model, it is necessary to transform the many and highly disparate features created by sift into a homogeneous histogram of k bins, with k corresponding to the number of *visual words*[1] being considered. To do so, it is necessary to fit a clustering model that finds a way to transform an immense amount of features into the k most representative instances. This is done utilizing a variant of the *KMeans* clustering algorithm, that is better suited to working with such large collections of data. With this clustering model complete, all of the features that were used to fit it can now be run through its *predict* function to allow it to construct a histogram that represents the unique combination of *visual words* that correspond to each image's features. After this is done, each of these *histograms* can be labeled and then added to a new dataset, where each sample consists of a 1 dimensional array corresponding to the values of the histogram[2].

VII. Second Model: Classification

The previously constructed dataset allows us to now train a classifier, as our samples now constitute a clearly labeled and homogeneous dataset. In this opportunity, a Support Vector Classifier has been chosen, given that it has presented better accuracy when tested over a simple test set. And thus we can now export both of the models that we will upload to the server, so that they may be used to calculate the classification of any image. These are exported as *pickle* (part of the joblib library) files, that permit the persistence of objects like trained models.

Note: The details of the implementation can be found within the source code for the project, in particular, every script is fully commented and provides a view into the entirety of the inner workings of the project.

VIII. Web application

The web application's architecture is simple, as was pictured above. Thus, all it does is do one of two jobs, either receive user images, or classify them. The web application was deployed on the Heroku Service, where it is deployed on a simple virtual machine and can be accessed at the following address: *imagetest185.herokuapp.com*. In order to successfully deploy the application on a PaaS service, it was necessary to utilize a different web server (namely: *gunicorn*) and, second, it was necessary to provide and install a set of dependencies in order for the image analysis to run on the containers. This was done by modifying the way the container is deployed and installing a set of packages before the image is finalized.

[1]Now, the meaning of this is essentially the following: In order for the Bag of Words model to work, you must create a collection of *words*, where, in essence, you count the number of occurrences of each word within a document. What this means is that you must run a clustering algorithm over the entire collection of *visual features* from all the images, in order to extract the aforementioned k *words*. These, then, in essence represent the k most representative features within the set; thus, they can be termed *visual words*.

[2]It must be noted though, that in this case the histogram is expressed as the value of its probability density function instead of the absolute frequency for each bin. This is calculated automatically by *numpy*.

IX. Evaluating Performance

Two key metrics are of particular interest in the design of this system. First, it needs to run quickly, to allow the web service to respond within the constraints set by the provider[3]. And second, the model must be reasonably accurate, and give reliable results.

In terms of speed, a satisfactory performance was achieved, with average web response times under 10 seconds (with a best time of 5s observed). In terms of accuracy, however, several cluster sizes were tried in the hope of improving accuracy, which are related in the table below:

K Size	Accuracy over Set
300	70%
500	71%
1000	71%
3000	35%

Thus, for this particular instance, the $k = 500$ models were chosen. This means that although not necessarily perfect, the model performed reasonably well, and will allow the user to determine their location a good percentage of the time. Evidently, the best course of action to improve this accuracy would involve creating a better and larger dataset, and possibly tweaking the feature extraction algorithm.

X. Conclusions

Evidently, the final performance of the project leaves a bit to be desired, as the sought after improvement could not be achieved, however, the accuracy is satisfactory and, thus, the project can be deemed successful. In addition, the project runs flawlessly on a Heroku container, which is no trivial feat given the complex web of dependencies that exist. Further development of this project would see a push towards a greater amount of samples, and perhaps, a better feature extraction algorithm, such as SURF. In any case, the final product meets all required properties in some form or another, and, as such, can be considered complete.

References

[1] https://docs.opencv.org/
[2] http://scikit-learn.org/stable/documentation.html
[3] http://pandas.pydata.org/pandas-docs/stable/
[4] http://www.numpy.org/
[5] https://dask.pydata.org/
[6] https://pythonhosted.org/joblib/

[3]Heroku restricts processing time to 30 seconds. Any request that takes longer to process is stopped, thus speed is of the essence.