Alex Jensen, Amanda Shen, Eunjun Choo, & Sam Fertig

Professor Roberto Hoyle

CSCI 241 - Systems Programming

15 May 2019

"Visual Searching" Using Google's Vision API and Image Segmentation

Introduction

Our project can best be described as, for lack of a better term, a "visual search." We created an interface that allows the user to enter a search query on an image and either receive a "not found" message or display the desired element within the original image.

While brainstorming ideas for this project, image manipulation appealed to all of us. However, we didn't want to just segment or reconstruct images, and so we strove to come up with an idea that utilizes segmentation for a greater, more ambitious goal. Part of our motivation for this project was to gain experience building something that has utility, and image segmentation by itself did not seem too useful.

At our level of success, the program's functionality is limited to classroom demonstrations. With more work, though, we could optimize it to find elements in an image otherwise invisible to the naked eye, which has obvious implications in an industry setting. For example, in a photo of a crowded forest floor, the program might affirm the presence of a grasshopper the user hadn't been able to see or perhaps hadn't noticed. In terms of utility, the program is telling of its own potential—it's not accurate enough to find things that the user cannot find, but it could be in time. Our goal was not to have it working at maximum efficiency,

but rather to build a solid program capable of demonstrating its own potential and serving as a base for further expansion.

Another motivating factor was making our project interdisciplinary. There are many moving parts to the program, all of which confront their own issues and solve different problems. We needed to extract elements from an image, so we used image segmentation. We needed to check those elements against an AI model with a wide range of capabilities and a large database, so we used the Google Vision API. We needed a way to communicate and manipulate this data cleanly, effectively, and efficiently, so we created a user interface which greatly simplifies the user's role in the program and allows us, the developers, to test, debug, and patch with ease. Computer science, despite its youth, is an increasingly interdisciplinary field; we took advantage of this by using different technologies with distinct advantages and uses to accomplish a single goal, which is not only useful—consider our motivation of practicality above—but also interesting.

This interdisciplinary approach was also intuitive. To solve our problem and to accomplish our goals, we split the project into smaller, more manageable components. Each of these components had a clear solution, and so we approached our program knowing that each separate module would be working before we would integrate everything into a final deliverable. Looking at the project as a whole was somewhat overwhelming and this compartmentalized approach gave us a sense of direction which allowed us to stay motivated and on track.

Methodology

Our methodology is almost identical to what we had envisioned in our proposal. We used C# to implement our program. Though there were obstacles along the way—at one point we almost switched to another language, or to a new combination of languages, because C# was giving us trouble due to its Windows-based nature—we persevered and remained consistent across the board by writing all code in C#. The fact that it is object-oriented was also just as useful as we had predicted; image segmentation in particular was facilitated by the creation of namespaces, objects, and classes and class methods. We relied heavily on online resources to familiarize ourselves with the language[1] and to import modules relevant to image manipulation.

We relied even more heavily on online resources when it came to image segmentation. In the end, we implemented two segmentation methods: k-means clustering (KMC) and edge detection. KMC, a popular image segmentation method, was the most applicable to our program. It proved useful in splitting images into smaller components and allowed us to save each image portion as its own file, which is necessary to use the API.

The KMC algorithm creates an initial cluster with the image and a set of centroid pixels randomly selected from the image. Then, it computes the actual distance using the Euclidean distance formula between these centroids and each pixel in the image. We add pixels that exist within our defined constraints—distance and color constraints—into a cluster and repeat this step until there are no more pixels to consider. After this, we convert each cluster into a new image and calculate the nearest mean by using the center of mass formula. Finally, we return all of the segmented parts and reconstruct the original image using said parts.

[1] A full list of citations can be found on the works cited page.

We also experimented with edge detection, which we thought would boost the accuracy of our program. Edge detection scans through the image, looking for sudden changes or discontinuities in brightness. These changes are labeled as edges. Different forms of edge detection are implemented using different matrices, and we worked mainly using a 3x3 and a 5x5 Laplacian of Gaussian matrix. Since we attempted to find the different elements of an image, finding its edges would allow us to identify these elements with greater accuracy. However, it turned out to be almost impossible to incorporate this into the function of our program. By itself, it is the more accurate of the two techniques. The algorithm is designed to return a single image. As the API portion of our code requires the different elements of an image existing as their own image, we couldn't use edge detection to facilitate our "visual search." However, we did build it in as a separate function within our program.

Consequently, relying on the less accurate[2] of the two algorithms, we adapted our API component of the program to carry more of the workload. We had previously been working with booleans, qualifying an image based on whether or not it contained the user's search query (returning either true or false). However, the program now measures the confidence of all the different API responses by returning a sorted array (in descending order) of the labels in the image. That is to say, if imageLabels[] is an array of GoogleVision's guesses at the contents of an image, imageLabels[0] is the most accurate guess (judged by the API itself) and imageLabels[imageLabels.length] is the least.

To use the API in general, we consulted the Google Vision website and tutorials made available to the public. The kinds of resources that we decided to use varied between the API and

[2] Accurate in a different way, at least.

image segmentation. Since Google Vision is Google's trademarked technology, the company has released clear and comprehensive documentation on its different components and how to use them. The Google Vision site was all we needed in order to utilize it effectively. Image segmentation, on the other hand, is not a marketed product[3] and so we sifted through databases (e.g. JSTOR) and computer science forums (e.g. Stack Overflow) to find a balance between scholarly and more candid instruction. We consulted both types of sources; many proved useful, and some did not.

Findings

As stated previously, our project can best be described as, for lack of a better term, a "visual search." Our interface allows a user to choose between two types of image segmentation: k-means clustering or Laplacian edge detection. If the user decides on k-means clustering, they can then enter a search query and see if that query appears in the image. Otherwise, using edge detection, they can choose between using a 3x3 or a 5x5 Laplacian of Gaussian matrix and the interface then displays the resulting segmented image. In either case, the user has the option of saving the resulting image or returning to a previously displayed image. As it stands, the program works as follows:

(1) The user runs the program and an interface appears. They are prompted to enter a search query and to upload an image on which to search. They may also select a segmentation method, but for the API to work, only k-means clustering may be selected. The edge detection option will merely return the original image but segmented—the query is irrelevant. For the purposes of this walkthrough, assume k-means.

[3] In the context of this project...

(2) The uploaded image is passed into the clustering method. The image is segmented and each segment saved as its own image in the folder the program was run in.

(3) Each image (each segment of the original) is passed into the API function, sending a label detection request to the GoogleVision server using a private .json key for authentication. If successful, the function returns a sorted list (in descending order) of labels for the image. That is to say, list[0] is the Vision API's best guess at what the picture contains.

(4) We iterate through each list in order of API confidence (all 0 indices first) to correct for duplicate segments within an uploaded image and the API's occasional ambiguity.

(5) If a list element matches the user query, the segment is displayed.

(6) If no list element matches the user query, the interface displays "not found."

The end product diverts from the expectations of our proposal. In certain areas it exceeds them, in others it falls short. We exceeded our expectations in terms of the user interface and the quality and speed of both segmentation techniques. Unfortunately, our initial search model failed to take into account how elements are constructed in an image. The k-means clustering algorithm splits an image according to color, and a user-uploaded picture of a rainbow achieves excellent search results. A picture of a farm, however, will not properly segment a chicken, since a chicken might be comprised of different colors. In sum, our program works very well when searching for colors, but fails to find non-monochromatic elements in images. We are excited with what we have accomplished, as we've achieved things we did not expect to: the interface and the efficiency of the algorithms. The fact that we are able to connect to the Google Vision API in real time, as well as segment images extremely quickly and in a manner that is accessible to the user, is exciting in its future prospects. As we move forward with this, the priority will be to use

edge detection to segment images in a way that allows API use. We could then achieve the results we had initially sought after, and include the extra functionality we've inadvertently created.

Though we all discussed with each other periodically, we each developed certain areas of the program. Amanda and Alex implemented k-means clustering segmentation and edge detection respectively, while Eunjun created the interface and wrote the main method capable of combining the aforementioned segmentation methods and the API capability that Sam developed.

Discussion

What we learned can be divided into two types of knowledge: technical and non-technical knowledge.

Technically, we became fluent in C# and learned how to work with its most popular IDE, Visual Studio; we even learned how to use it to build an interface, something we haven't covered at all in our computer science courses. We became more proficient in GitHub since we used it to share code and collaborate effectively with our groupmates. We experimented with two different image segmentation algorithms, k-means clustering and edge detection, which are both prominent techniques in image manipulation. We learned more about how the Google APIs work and, more importantly, how to incorporate them effectively into our code: specifically, how to integrate the shell and the Cloud SDK, how to send requests to the API after authenticating using a .json key, and how to receive information and parse through it to communicate meaningful, relevant user responses.

On a non-technical level, as we predicted in our project proposal, we learned about the good that comes from working in a group as well as the challenges it presents. Of course, the technical and non-technical stuff aren't as segmented[4] as this paper suggests. In reality, much of this project's value comes from exercising the technical skills outlined above while operating in a group setting. The two are complementary, and that's also something we learned over the course of this project.

Our timeline was realistic. Though there were some delays and miscalculations, we achieved our milestones roughly when we said we would. We familiarized ourselves with C# quickly, allowing for some extra time on image segmentation—which we needed. We hadn't expected to implement two different algorithms, but the debugging process was, overall, straightforward. We were therefore able to allocate more time towards the two segmentation techniques—the bulk of the project. Additionally, we were able to create a simple interface for our program.

If we could go back and revise our timeline before getting underway, we'd have allocated more time towards making the program portable, i.e. runnable on machines with different architecture. We ran into some delays because the program only runs on Windows, and three of us MacOS.

Future Work

There's a still a lot left to do. Not in the scope of our project—we accomplished everything we set out to, and more—but this program can serve as a solid base for more advanced, impressive features. For example, our edge detection segmentation method is

[4] Pun intended.

significantly more accurate than its k-means clustering counterpart, but we haven't found a way to split up the segmented image into different saved images. If we did, then the API would return more accurate results. There are also many other image segmentation techniques we can implement. The interface, too, can be expanded; we can make it cleaner, prettier, and more resilient.

Except for the API (because it's not our technology), virtually every aspect of our program has room to grow. As for the API, there are ways to improve its accuracy by improving the quality and function of the methods surrounding it, such as implementing a more accurate image segmentation technique.

We can also rethink the function/purpose of our program. What if we searched for audio snippets within a larger audio file? What if we implemented recursive segmentation: would that mean a quicker, cleaner search? There are a number of ways we can improve and expand upon our program as it stands and a number of ways we can rethink its overall function while keeping it conceptually consistent.

Link to Repository

The GitHub repository for all of our work can be found at https://github.com/danyatingshen/ImageSeg.

Works Cited

"C# Edge Detection." *Instructables*, Instructables, 8 Oct. 2017, www.instructables.com/id/c-Edge-Detection/.

"Cloud Vision API Documentation." *Google*, Google, cloud.google.com/vision/docs/?_ga=2.50784873.-1560128097.1552867065.

Esterhuizen, Dewald. "C# How to: Image Edge Detection." *Software by Default*, 30 June 2013, softwarebydefault.com/2013/05/11/image-edge-detection/.

Ratz, Arthur V. "Implementing K-Means Image Segmentation Algorithm." *CodeProject*, CodeProject, 29 Aug. 2017, www.codeproject.com/Articles/1203227/Implementing-K-Means-Image-Segmentation-Algorithm.

Tutorials Point (India) Pvt. "C# - Introduction to GUI." *YouTube*, YouTube, 5 Feb. 2018, www.youtube.com/watch?v=sHUXbCXmZKU&t=914s.

We affirm that we have adhered to the Honor Code on this assignment.

Sam Fertig

Alex Jensen

Amanda Shen

Eunjun Choo