9/6/2019 Log:

After Dr. Gabor’s talk today about the level of detail we should have in our journal reports, I decided that I’m going to detail my thoughts more technically and thoroughly. It’s better this way so that, as Dr. Gabor said, 10 or even 50 years now, I can look back at this and remember the exact logical steps I deduced, and any appreciate any of my trials and tribulations, along with great revelations.

I realized that the first reaction that I should have had when seeing the intriguing web crawler application (from the PSIAP-CAP annotation app I learned in Beaver Works) is that I should’ve looked at the darn code itself, and not google for other web image scrappers as I have been doing thus far. For the NOAA disaster imagery site (refer to my first journal report to see link), the PyPi Image scrapper wouldn’t work on scraping all the images, so then I thought “Why not look at the web-crawler code on the PSIAP-CAP app?”, and upon looking at the comments of the code, the code looked way more promising, since it seems to recursively go through a website and extract all images, which I failed to see the PyPi image scraper to do. Unfortunately, upon analysis of the web crawler code with its documentation, it looks like the output is embedded within the web app, and not something I could use to download images based on a URL. That’s just my speculation at least, since I’m not really familiar with the web app syntax interfaced with Python. I’ll have to keep looking for either other code or an online tool for image scraping.

9/8/2019 Log:

Connor and I discussed about how we’re going to actually make progress this week for annotating the images; we still have yet to find the right annotation tool and datasets. Although, a specific insight came into my mind. For aerial photos of disasters, maybe instead of just searching up “disaster aerial imagery” as I did earlier, perhaps I should try searching on more specific types of disasters. For example, I could search up “earthquake aerial imagery” or “Hurricane Katrina aerial images” instead. When I tried searching “earthquake aerial imagery”, sure enough, I got many useful Google Images that were relevant to earthquake aerial imagery!

Connor and I also discussed about narrowing down what times of images we’d like to see. I wanted to consider that different images were all taken probably at different altitudes above the ground, and that in google image results, obviously certain images could be irrelevant. Connor assured me by saying that “In theory, since neural networks learn through repetition, as long as there aren’t too many irrelevant images, it shouldn’t matter. I don’t know about different altitudes though.” In the grand scheme of things, I thought perhaps different altitudes weren’t too important, but I wanted to just try searching up same altitude photos anyway, since neural nets learn best when they can detect patterns within data. Having different altitudes might confuse the neural net. Unfortunately, there was no look for me finding same-altitude photos. I decided then as long as all the photos in my training datasets were just aerial (not too close to the ground, and not far up as a satellite), that would be a good enough criteria for my training dataset.

After this discussion about considerations for our dataset, another problem aroused in our minds. Would it be a problem if all the images had different resolutions? Sadly, upon looking it up, Convolutional Neural Networks can only process same-resolution images. Luckily, Connor mentioned that last year in A.I., he had to train a CNN using fast.ai, and they used Google Images. Therefore, Connor / the code he was using to train a CNN had to account for different resolutions somehow, and luckily Connor was able to retrieve the Google Colab file he used to learn about all this and let me have a look at it.

9/9/2019 Log:

Yesterday night, Connor shared with me the Google Colab file that could extract all the images from a google search, and feed all of that into a convolutional neural network to classify them in a set of labels (e.g. different types of bears). For the one block in anchor day I have today, I plan on investigating that code Connor shared and see how I may leverage that code to fit Connor and I’s dataset needs.

Upon the conclusion of class, I was confused by one aspect of the code. I understood that I needed Javascript to get a file of all the image URLs from a google images search, but what I was confused about was where to upload it. The Google Colab file said to upload in my local repository, but clearly the code didn’t show how to connect to my local repository, so I was stuck for a while, thinking of what to do.

9/11/2019 Log:

I decided to try the Google Colab file instructions one at a time before understanding the whole local repository issue. I tried out the JavaScript code to get a list of all the image URLs and the Python code on that site to see if it could truly download images on my local drive. It works successfully, except that I wasn’t too happy that the Google Colab code could only download 200 images at a time. I decided to try a different route then. When I simply looked up how to download google images, an interesting chrome extension popped up: “Download All Images” (<https://chrome.google.com/webstore/detail/download-all-images/ifipmflagepipjokmbdecpmjbibjnakm>). So, I was able to download all 956 images when I searched “earthquake aerial photos” on Google Images. Hurray! Now, all there’s left to address is: “How is the actual annotating going to get done?”

My fellow comrade, Connor, was fortunately able to try out the the “labelimg” python package that I found and he was able to get it to work. He did warn me though that the instructions as to how to get the annotation tool working on the website weren’t exactly correct, as he had to install certain intermediary packages untold in the instructions.

In addition, he found that in order to account for the different image size issue, we might look into using what’s known as a “Dynamic Neural Network” from PyTorch. There’s also the “YOLO” (You Only Look Once) algorithm that is specialized to train neural net to detect multiple features of an image instantaneously. This sounded like the perfect algorithm for our project, since immediate decisions in disaster response is crucial. The YOLO algorithm was also recommended to me by Omkar Bhalerao during the activity where we all went outside and walked backwards to share and ask about each other’s ideas.

You might be wondering why I didn’t refer back to the Google Colab file to see how it accounted for different image sizes, and honestly I don’t have a better answer than saying I forgot. I didn’t want to focus on neural nets too much at the moment, but rather see a feasible and efficient way of getting the annotating done.

.

9/12/2019 Log:

Connor and I discussed about the annotation part, worried. We needed to come up with a method soont. While all our tool researching to use to download and annotate images was useful, we felt as if we weren’t making measurable progress, because our milestone was to get the annotations done by this coming Monday. While looking up specialized software would be great (if we knew how to actually use them in training our neural net), we discussed and came up with a feasible plan. We would create a .csv file where each row would have the image photo ID, and labels for each column. Then, I’d read in that .csv file into a Pandas dataframe. Pandas is a useful Python package that can provide organized collections of data called data frames in order to perform data science operations on it. Because of Pandas’s appeal to organization and cleanliness, I thought that would be useful to package my training dataset in an organized fashion.

I’ll have to see if I can accomplish all this by next Monday (9/16/2019). I want to have the Pandas dataframe store each photo so that the neural net can receive each photos in its full glory, rather than just its ID name. I thought the annotation part was going to be easy, albeit tedious work. However, as you may see, problems and considerations piled on top of another ever since the fundamental question “Well, how am I going to actually store the labeling for each photo?”, making my training-dataset-readiness milestone harder to achieve than expected. Even if I don’t get this done by Monday, I don’t want to spend too much extra time on just garnering up a training dataset, because the real meat of my project should be focused on neural network training and the user interface. Worst comes to worst, instead of labeling multiple features for my first milestone, I’ll just simply annotate for each image whether or not it has a disaster struck in it, and move on to multi-label annotating later. I only need a CNN that can classify whether there’s a problem going on in an image or not for my second milestone. I originally wanted to get all the annotating done in my first milestone of this project (including multi-label imaging), so I wouldn’t have to deal with any annotating later on in my project. However, after what I’ve been through, it may be best to split up single-feature annotation and multi-feature annotation in two different milestones. We’ll see by Monday how all this will go.