National Cancer Institute Internship Mid-Semester Accomplishments: A Smorgasbord of Image Preprocessing Algorithms

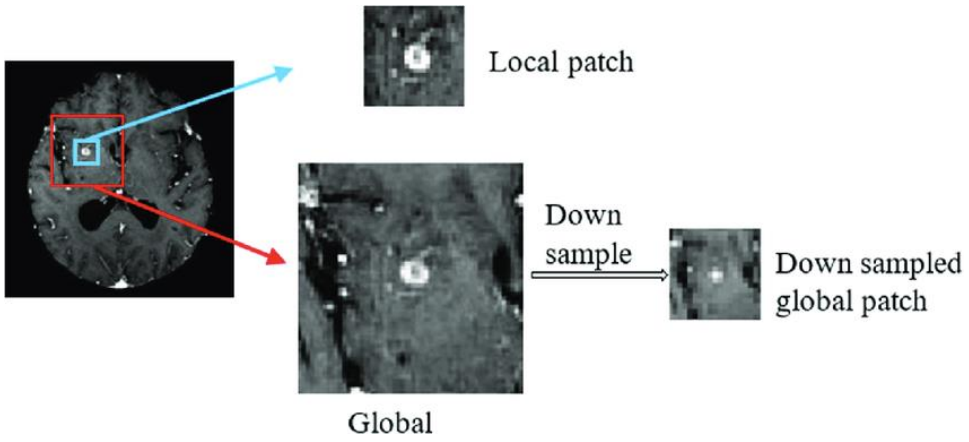
Chris Pondoc

Introduction

Since the end of my training period at my internship, I have been asked to implement a plethora of image preprocessing algorithms from scratch in Python. In this report, I discuss a couple of the tasks I have been able to accomplish. All of the code can be found here.

Task 1: Patch Extraction

The process of patch extraction involves taking an image, extracting small slices of the image called patches, and then training the network on those individual patches. In our case, we extracted 256 x 256 patches with a 128-pixel shift every time we took patches. Given that the default input sizes for the majority of the pre-trained architectures we were utilizing were roughly 224 x 224, it made sense to go with this patch extraction size. Using the Python Imaging Library, we were able to extract a total of approximately 92,000 patches.



Task 2: Generating Random Patches

Aside from simply generating patches, I also had to make sure to randomize where I generated patches -- after all, it would be better to train a neural network on patches from a variety of different images from a variety of different subtypes of carcinoma and sarcoma. This meant that I had to do random patching. To implement this algorithm, I simply divided up the entire image into rows and columns and treated the image as a coordinate plane. Then, I would generate a random "point" on the plane by outputting the number of a random column and row. This, in turn, would serve as the coordinates of where I would extract my patch. From each image, I would end up extracting only about 4-5 patches to ensure enough variability.

Task 3: Throwing Away Bad Patches

Part of the issue with extracting patches is the usability of the patch. For example, some patches feature way too much black in the image, while others feature way too much whitespace. Thus, our team decided to throw away some of the patches if the color contents of the patch contained either too much black and/or too much white. Let μ = the unusability of an image.

$$\mu = (\%_{Black} + \%_{White}) \times 0.1$$

If .5, then the patch will not be used. If <.5, the patch will be used.

After parsing through all of the patches and analyzing the color contents of each, we determined that out of the 92,000 total patches we extracted, only 69,000 of the patches were usable. While our networks are trained on significantly fewer images after this operation, this ensures that our network will not be trained on strictly black or white images, which lend no helping hand in distinguishing between sarcoma and carcinoma.

Task 4: Color Normalization

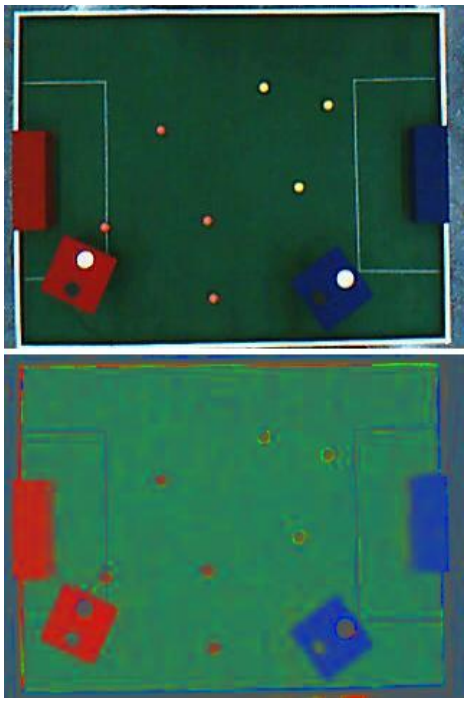
To help increase accuracy and to make our networks converge faster, our team also employed the strategy of color normalization, standardize the colors of the image and allows the network to converge faster. In our case, our team used the "RGB Max Technique", which manipulates an RGP tuple (R, G, B) using the following set of expressions:

$$R = 255 \times \frac{R}{(R + G + B)}$$

$$G = 255 \times \frac{G}{(R + G + B)}$$

$$B = 255 \times \frac{B}{(R + G + B)}$$

An example of an image with its colors normalized can be seen below. In general, the distortions that may have been caused by external lights or shadowed have disappeared, with the computer still having the capability of processing this image.



Resources Utilized

In terms of programming, I wrote these programs in the language Python. I also utilized several frameworks, which include (but are not limited to):

Keras and Tensorflow

Python frameworks employed for machine learning testing and neural network creation.

Python Imaging Library (PIL)

Framework used to manipulate, create, and work with images in Python.

Matplotlib

Framework that is typically used to show graphical representations of data, but was used in my cases to programmatically show the images within my development environment on a grid.

Numpy

Framework that helped me deal with the images as numerical data values (i.e. colors for each pixel for my color normalization algorithm).

In terms of resources, I used the helpful book Deep Learning with Python by François Chollet. This is the book I have been utilizing since the beginning of my internship, and I still reference it now and then when trying to build specific machine learning algorithms. For many of the tasks that I completed, their implementations could not be found in the book; however, the conceptual foundations of the algorithms could be found and were immensely helpful when creating them from scratch.

Significance of Work

During the end of the summertime, I experienced a transition in co-interns -- with one of the interns working over the summer and another intern coming in to help me out during the school year, I was the only person who understood the entirety of the project. Thus, the implementations of the aforementioned algorithms allow for the work on our publication, which covers deep neural network classification of low magnification region of interest images, to move forward and to serve as an example for other interns to come.

Furthermore, understanding how the best work with images is a huge part of the field of machine learning. Aside from programming neural networks, a portion of my time has been spent working on creating “ground truth” image labels using GIMP. Without properly annotated and processed data, neural networks do not have a good standard to train with, thus rendering the entire notion of training neural networks useless.

Helpful Prior Experience

While I did not know much about the Python programming language or libraries such as the Python Imaging Library, I did have a lot of experience with basic programming concepts, such as object-oriented programming and loops. Furthermore, throughout my experiences with programming in the past, I learned perhaps the most important skill when trying to code: perseverance. Thus, even though I had to learn pretty much an entirely new language throughout my internship, having a solid grasp of the fundamentals and an

awareness of the persistence needed to achieve success in such a position enabled me to solve these tasks.

Skills Learned

One of the things that seemed rather straightforward but never really dawned upon me too much was looking through documentation and/or open-source code on Github. In the past, being as though I didn't have a lot of ideas as to what I would program, I would almost always try to start by watching tutorials on YouTube and developing ideas based on the examples I would program. Now, however, given that the task is already outlined, I would oftentimes need help with a small little detail about the program, whether it be the utilization of a single function or importing a specific library. In these instances, watching a video is neither necessary nor helpful; however, reading documentation -- for example, on the Python Imaging Library -- or reading through someone's sample code is. Now, whenever I am stuck on a bug that isn't too grandiose of an algorithmic task, I will always make sure to go to the source of the library or example programs that utilize the library, which gives me a better understanding of how to program and use certain functions in context.

Another important skill that I learned was to outline my programs before simply writing them all out in code. Don't get me wrong -- on occasion, I will still make just copy and paste a bunch of code and play around for a couple of minutes to see how they might piece together. Nonetheless, throughout my internship, especially through the image preprocessing programs, I have learned to write down my ideas on paper first, then organize them into a most efficient algorithm, and then begin to write code. This process of brainstorming, outlining, and then finally coding seems obvious but can be overlooked, especially when you're in a "zone". Over time, though, I have learned to always adhere to this simple routine to keep my programs efficient and clean.