**Friday 9/20/2019**

Ahh finally, some research on a conc ept more exciting than annotations: convolutional neural network training. Since my 2nd milestone involves a binary classification -- seeing whether there’s a problem inside of a photo -- I googled up how to train a CNN that can achieve just that. I attempted to use this tutorial: <https://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library/> , which had a training dataset I could easily understand, so that was off to a good start. Basically, I was simply following the instructions on the website, which mentioned how to train a neural network model using Keras.

However, I soon realized that the dataset used involved numbers and not images, so then I tried searching for tutorials that focused on image datasets. Unfortunately, no luck by the end of class.

**Monday 9/23/2019**

This link seemed more promising: <https://github.com/sagieppel/Fully-convolutional-neural-network-FCN-for-semantic-segmentation-Tensorflow-implementation>. So, I decided to investigate it further, and see if any of the code could be of any use. According to the instructions, I just need to set up the paths to the folders of the images for the ground truth labels in the “TRAIN.py” file.

Then I realized, “Oh, this is pixel-based annotation, which classifies every single pixel in a certain class, and the neural network outputs a coloring scheme of the image, each color representing a different class. In our intended neural network, we would still like to preserve the original image as output. We only want to know under what labels will the neural network to classify a certain image, and not particularly where, let alone where pixel-wise.”

**Wednesday 9/25/2019**

I continued to read the instructions for how to set up the neural network from the website mentioned on Monday^^. It turns out though that their annotation system seems far off than a simple image annotation task.

I also read on Dynamic Neural Networks: <https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23> to see how they’re able to handle inputs with different sizes, and this is relevant to my project since I need to figure out a way to handle images of different sizes. I simply wanted to understand the theory behind DNNs and how they may possibly be implemented on machine learning libraries such as TensorFlow. Surprisingly, the word “bucketing” came up, as I learned about it in A.I. class.

**Thursday 9/26/2019**

While familiar terms like “bucketing” did appear on the article, in general both Connor and I thought that dynamic neural networks was overkill for what we needed to do at the moment. We decided to go with resizing the images since that’s obviously more feasible to implement an algorithm for, and thus be able to use a normal convolutional neural network after all. I was initially opposed to this idea because resizing all the images might make some of the images look really weird, and that’s why I was initially so inclined to try out dynamic neural networks. We decided that this would be challenging to implement and again, we had to focus on going from basic to complex. We would need an actual understanding of the algorithm to be able to implement it, unlike what we had thought earlier. Generally, we thought any ML technique should be straightforward to implement without much knowledge of how it works, but unfortunately we saw that this was not the case. We then begin to shift our thinking from simple → complex. Apparently, resizing images is easy to do with the Python Imaging Library (PIL) with a method called “Image.resize(w, h)”. We decided to develop code that can take a folder of images as input, and as the output, the same folder of image, but all images standardized to the width x height we decide on. We still have not decided on width and height yet, so we decided to leave our code as a method, with two parameters “width” and “height” so that we could pass on whatever we come up with later.