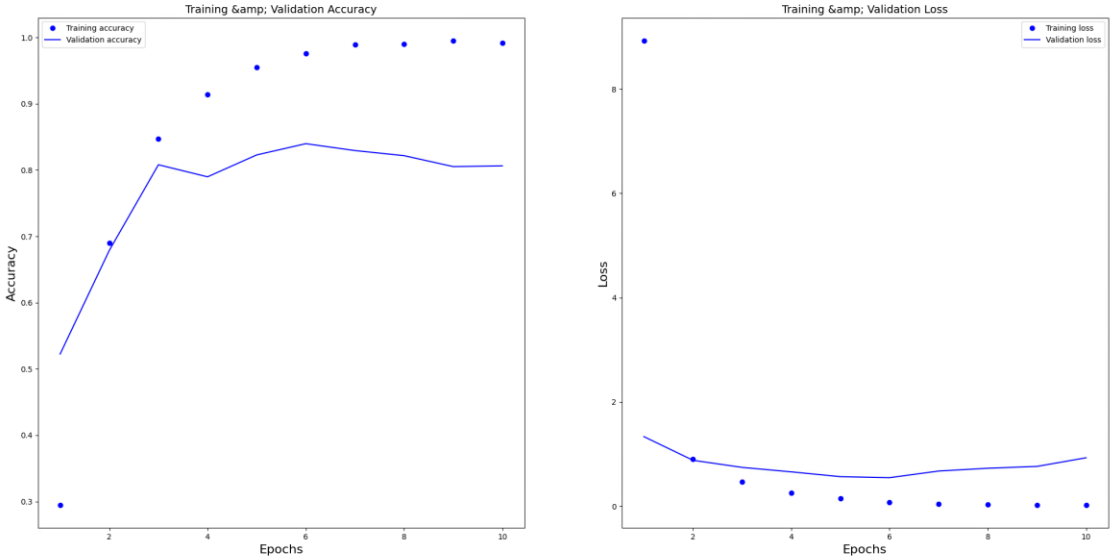
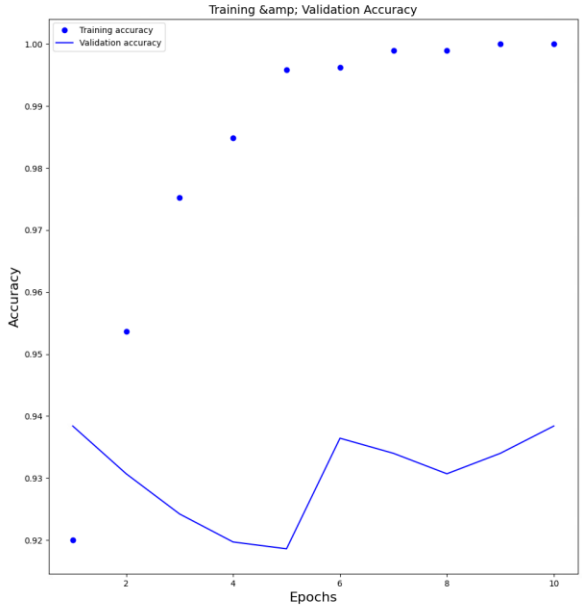
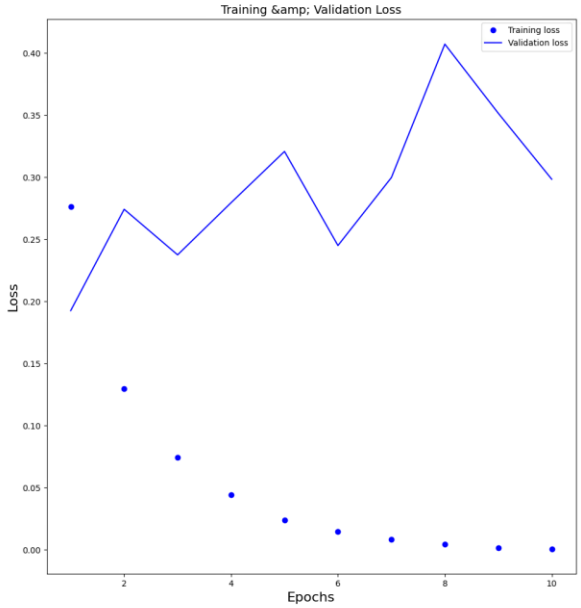
We both agreed that a CNN would work well and that it was something that we could get done in the time we both had. We wanted to make the TACO and Mask_RCNN work but we were wasting too much time and we could not even get a model working. So, we decided to try out making our own CNN and just using a histogram as our feature vector. In hindsight we already knew this was a bad idea so we should have searched better image pre-processing methods earlier. However, we eventually found the ImageDataGenerator package in keras and thought that it looked very user friendly and luckily it was. We found out very quickly through basic experimentation that adam was by far the best optimizer. The rate of convergence was so much faster than all the other optimizers. When we looked up popular optimizers for CNN, we saw that most people use the adam optimizer, so we just stuck with it. The methods for splitting up the images into directories was done hastily and was not done well. Had we more time that would be something we could clean up quite easily. The way we did it was not memory efficient at all, nor was it clean.

Now we can discuss some of the interesting bits about the actual model performance. We found out very quickly that our model is very prone to being over trained and it begins to happen around the 7th or 8th epoch. Below is a graph demonstrating this



What we did to solve this issue is to split the data into new training and validation sets and then continue learning on the new sets. Looking at the next graph you see that there is a major improvement to the validation loss and accuracy with even just one reshuffle of the training and validation sets. Additionally, I think it becomes clear that if you reshuffle and then retrain the model to the new training sets enough times both validation loss and training loss become very close to 0 and accuracy stays around 99.5%.





If you would like to see if you can produce these graphs yourself then do the following.

1. Make sure to comment out everything in the for loop in Final_Model.py
2. In the terminal run python3 Final_Model.py writeup
3. In the terminal run python3 load_model.py writeup.h5 writeup2

If we had more time, we would also debug another annoying problem we were having. For some reason we were finding it very difficult to append new data about accuracy and loss from the model to an array so that we could graph it all as one continuous line. If you look inside the for loop of Final_Model.py you will see all that code commented out.

I believe that our model is insufficient to identify each object in an image and if that is the goal of a project one must use an additional form of image segmentation. For example, the mask_RCNN project on github https://github.com/matterport/Mask_RCNN. I believe that ImageDataGenerator is a powerful image pre-processing tool however that github project is on another level and can cleanly identify objects in the foreground and background. You can find a link to how mask_RCNN was used to identify baseball fields in OpenStreetMap images on the github link above. This leads me to believe that the algorithm is generic enough to be applied to this problem.