Olivia Dill
9-14-2017

IDEAS FSS-Vis Write Up

For my final visualization project, I wrote a piece of to allow a user to manually annotate an image of a face with 68 points, marking 68 facial features, and save those annotations to a .csv file. This project was motivated by my work this summer with a set of ~150 portraits from 2nd century AD Egypt. I had used automatic facial recognition to extract the locations of 68 facial features for about 75 of the portraits with the goal of analyzing their shape and placement to asses chronology, artistic attribution, idealization and naturalism. The automatic facial recognition, which was trained to work on photographs did not work for the majority of my portraits and I needed a way to extract the information fro those which had failed.

I used Bokeh to develop this tool. The code needed to do four things: load and format an image to be displayed, set up a data source to store the location of the points, set up callbacks to read and store the location of mouse clicks and other actions, and write out the final data. Each part had its own complications. When loading in the images I found I had to reshape and reformat the image data, due to the fact that Bokeh expects four channel values, instead of the three that jpegs support. I had worked with java and python before, but had never seen JavaScript, so understanding how data sources would update, and how to format JavaScript callbacks was challenging, but among one of the largest takeaways from this project. My lack of familiarity with javascript and with console logging also made the process of accessing the final data in a .csv format challenging. Another complication was deciding how the points were selected. Currently each time a user single clicks, the point closest to the mouse is added to the array. This comes with the constraint that users must click through in a fixed order, which can be complicated if a user makes a mistake or loses track of where they are, and that the user must gauge the appropriate spacing between points which is not always obvious. I compensated for the potential for mistakes by adding callbacks to remove the most recent value or wipe away all values, and by displaying the index of each marked point. If I were to develop this tool in the future I would want to introduce more regularity to how points are selected. One might, for example, have the user outline where the face is and then algorithmically select which points along that line are evenly spaced or belong to the appropriate landmark.

The impact of this tool on my research is twofold. First, I now have a resource that I can use to generate training data, should I chose to retrain the automatic facial recognition code, for painted portraiture instead of photos. Second, within the scope of this project, I tested my tool by generating data for 15 portraits. I developed some code, also in python, using matplotlib and the Seaborn library, to compare the results of the new fits to old fits. I can now quantify my guesses about what the failures of the existing facial recognition algorithm are. The original algorithm tends to draws jawlines and mouths too large. Because the initial algorithm was trained on photos, and was trained to optimize the average fit of all the points, these failures, hold art historical weight in and of themselves. They explain where artists were deviating from photorealistic proportion, e.g. in foreshortening of the jawline to suggest perspective, and in decreasing the size of the mouth.