Charles Papandreou

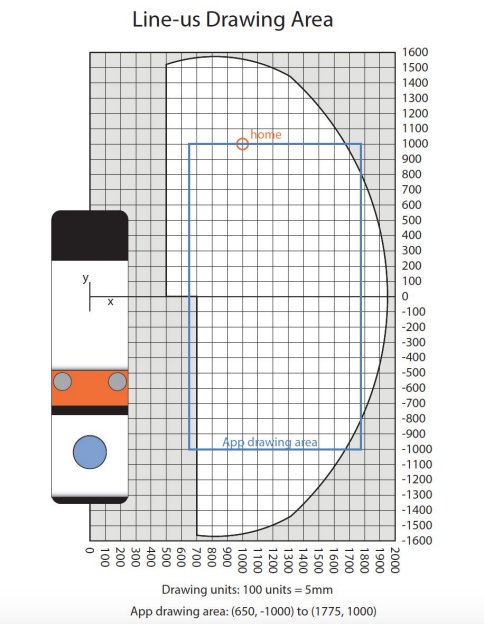
Duke University - IoT Lab

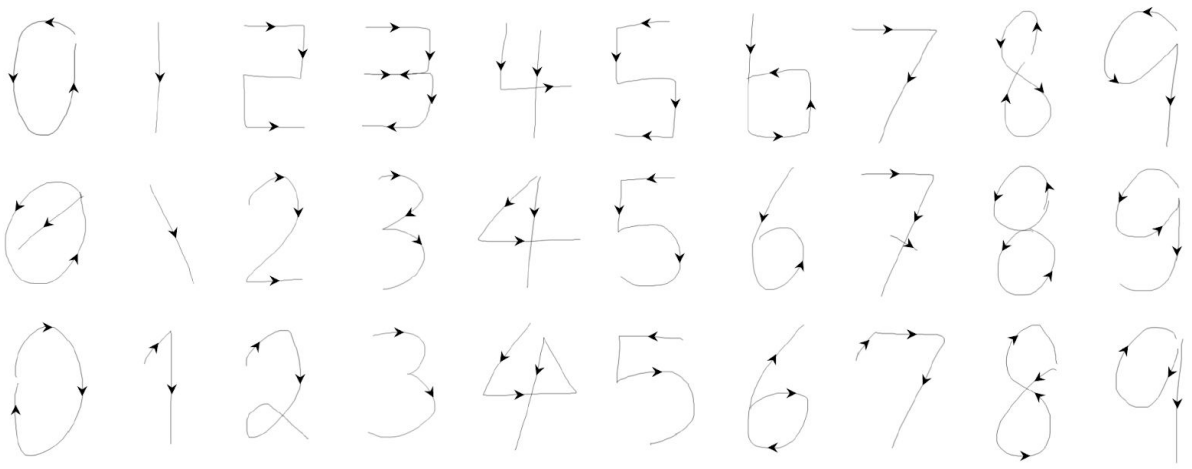
26 July 2019

Basic Machine Learning Classification for DMA-Based Motion Sensing

Line-Us & Digit Drawing Section:

In order to mimic the drawing motions and gestures of various characters, we used Line-Us. Line-Us is a small wifi connected robotic arm controllable with a TCP socket API. As a result, we were able to have the robot act on a python script and draw various characters. Initially, we focused on simply the numeric characters 0 through 9. Line-Us functions relative to a grid-based coordinate system. This grid is comprised of x-values ranging from 0 to 2000 and y-values ranging from -1600 to 1600. Using coordinates and some simple Python code, we were able to replicate gestures and movements representative of each of the digits. After considering the Mnist database for inspiration for different variations of each character, we managed to create three different portrayals of the motion behind each digit. It is important to note that since we are focusing on the motion for drawing each of the ten digits, it is the path taken by the robotic arm rather than the resulting picture that is important; Although two variations of a digit might look identical, the motion taken by the robot is different.



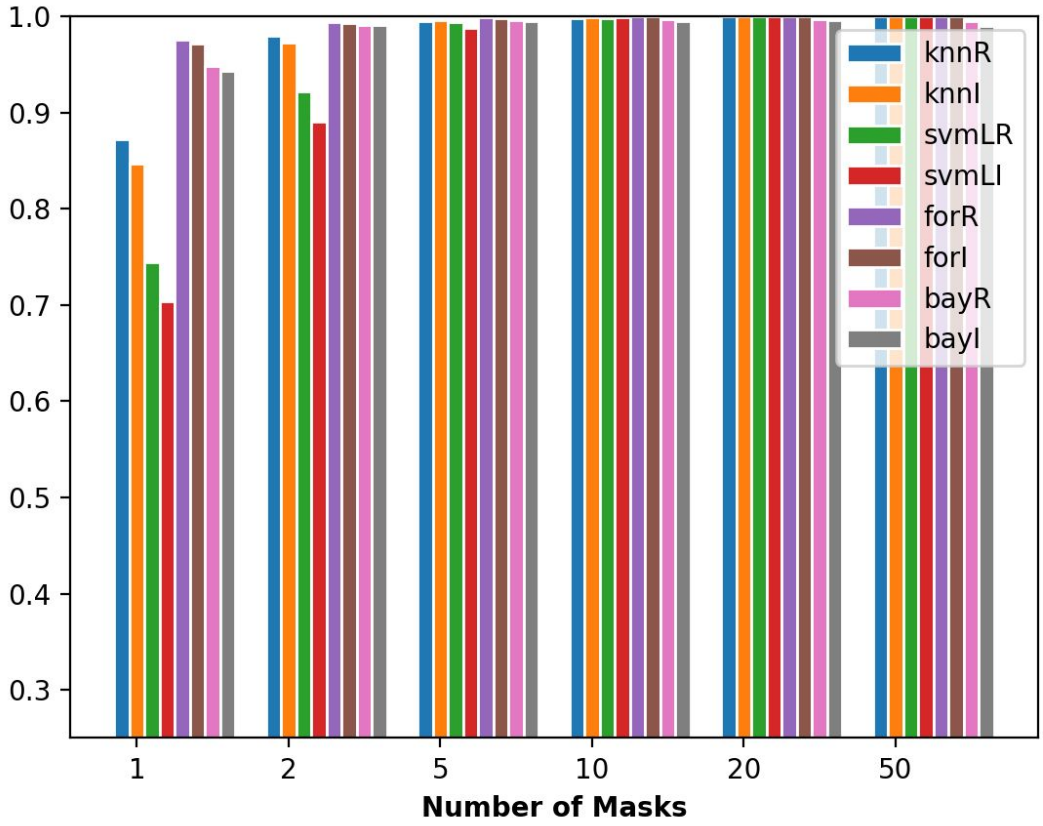


For example, although variations *0a* and *0c* create identical pictures of 0, *0a* starts on the right hand side and draws in a counter-clockwise motion, while *0c* starts on the left hand side and draws in a clockwise motion.

Initial Results:

Initially, only the first variations of each digit were looked at. We extracted four features from each time series, including peak factor, wave factor, coefficient dispersion, and auto-correlation coefficient. In total, this gave a resulting 200 features for each sample due to the 50 masks each sample involved. After performing feature extraction, we examined the average accuracy of four different learning algorithms with both real and imaginary values specifying for different numbers of masks. These measurements were averages after

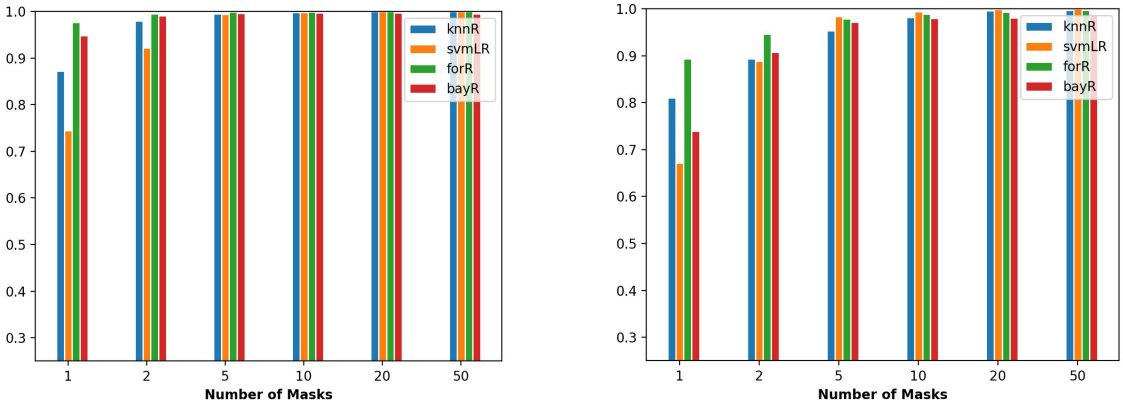
performing 10-fold cross validation along a sliding window of all masks for the specified range.



The above results show average accuracy values for k-Nearest Neighbors, Linear Support Vector Machines, Random Forests, and Naive Gaussian Bayes. It became clear pretty quickly that these results were too good for such basic algorithms with only 1 mask (4 features). The conclusion was that the signals for different digits were too distinct; our algorithms were overfitting to the data and the classification task was not difficult enough. After this realization we decided to add variations to the digits for a more complex classification task.

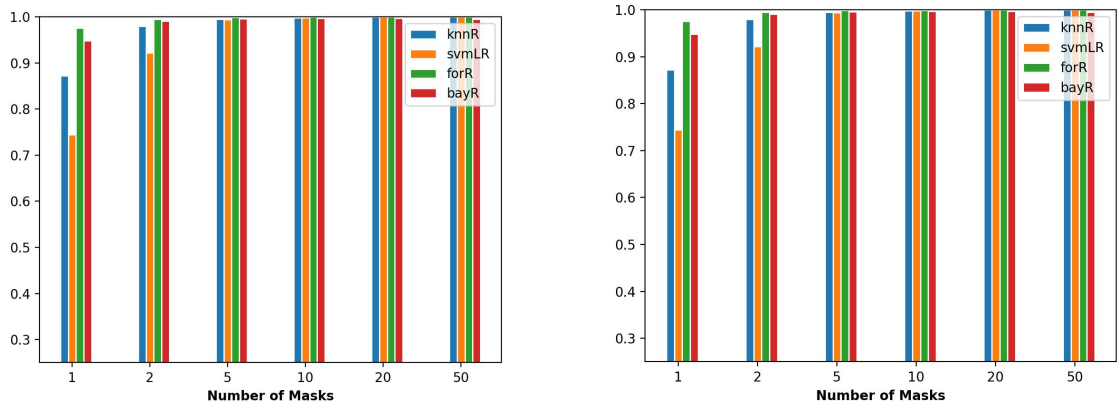
Effects of Data Size and Series Length on Accuracy:

While waiting for this second round of data collection, we started to look into how modifying the float type of the data might affect accuracy as well as how taking only some of the data points from the time series might affect accuracy. Initially, we decided to examine how accuracy for the different learning algorithms would change when looking at every time step compared with every 2nd, compared with every 5th… etc.



The image on the left shows accuracy values for each of the algorithms with every 1st time step taken into consideration when calculating and extracting features. The image on the right shows accuracy values for each of the algorithms with every 1000th time step (only 4 data points!) taken into consideration when calculating and extracting features. As we can see, this surprisingly does not have an enormous effect on the accuracy of the learning algorithms when performing classification. This seems to reaffirm the original issue that the signals are too distinct from each other; even when only considering four data points rather than 4096 to calculate features, the results remain similar. It is therefore unknown currently whether decreasing the number of time steps is a feasible option to decrease the amount of time once working with a more complex classification task.

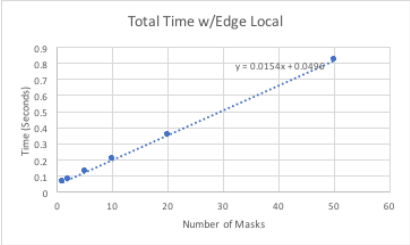
Next, we decided to look at the size of the float being used to calculate features. We examined float64, float32, and float16. They all had very similar accuracy values.

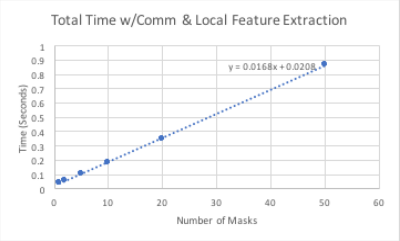


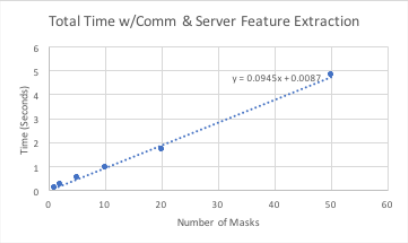
The image on the left shows float64 while the image on the left shows float16. As is pretty clear, there is little to no difference in the accuracy values, which is not entirely surprising. In terms of affecting the time for feature extraction or classification, decreasing the size seemed to affect time for classification, with smaller float size correlating with smaller time. One thing to note is that float32 is in fact quicker than float16. This is likely due to the fact that python and numpy don't support calculations in float16, requiring casting into float32 then back into 16 automatically whenever performing calculations with the values. As a result, it seems that using float32 for the entirety of this process would be ideal.

Edge Device Time Tests:

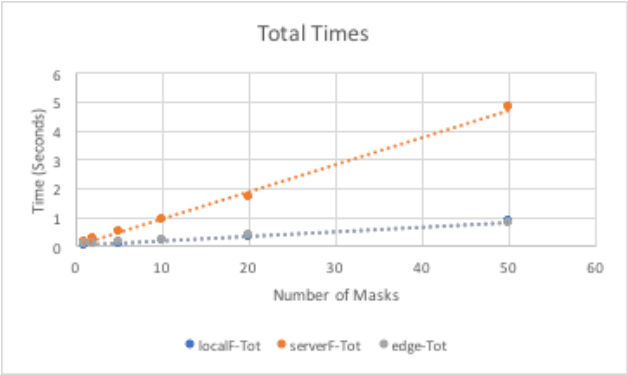
In addition to looking at how altering the quantity and size of data used might affect outcomes both in terms of accuracy as well as time and efficiency, we looked into performing different parts of this process both locally and in conjunction with a server.

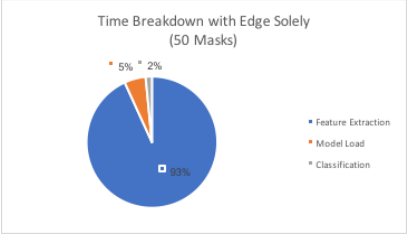


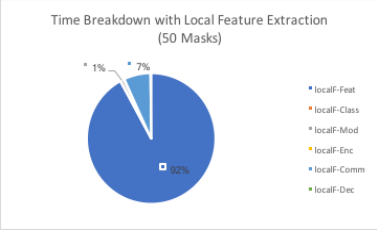


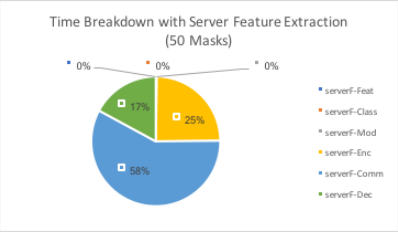


Looking at these three visualizations, we can see that the number of masks used when doing each of these different variations of this process all caused increases in time. Looking at the below image, we can conclude that it is quickest to perform the entire process on just an edge device rather than communicating with a server. A close second to that is to extract features locally and send the information to a server for classification. A large portion of time when adding communication with a server goes towards encoding, decoding, and sending this information, also as seen below.









Another observation was that communication time when communicating from the Raspberry Pi to the server increased with a polynomial relationship with the number of masks used when

sending raw data. It seems there are limitations on the Raspberry Pi's ability to send large amounts of information quickly (2.8 seconds for 50masks), as these issues did not persist when communicating with a server from a computer instead (.41 seconds for 50masks). It seems that there is a significantly larger amount of time required for sending the larger data package, which is a large part of the reason for the longer time required for server side feature extraction.