Project Three: Image Recogonition and the Use of Support Vector Machines

Gene Burchette

May 4, 2016

1 Introduction

For this project, the implementation was allowed to be fairly open-ended, focusing more on the results than the need for ingenuity or novelty. As long as the student could find a Python package or use a pre-existing machine learning package and could predict the label of a new image outside of the supplied sets of images with an accuracy above 90 percent, the task would be complete. Given that images represented as bitmap vectors, the most facile approach seemed to be to implement a support vector machine to analyze the patterns in the features (pixel RGB values). Initially, I watched a video series from a YouTube channel 'sentdex' on an introduction to image processing in Python and learned a few tricks about thresholding RGB values thus converting to black and white, blurring images to reduce sensitivity, and to look at new ways to make feature extraction more useful.

Figure 1: Stages of image preprocessing: Original, Black/White, Blurring

2 Approach

Starting this program mainly consisted of browsing the Internet for various python packages involved in image recognition like sentdex's video series, OpenCV, mathplotlib, and scikit-learn's SVM library. Once reaching the end of the video

series, the author mentioned that his simplistic model of feature extraction could be expanded by centering images, reducing noise, or detecting the most important features of an image and emphasizing them (like edges).As he shared his imagination improving the feature extraction, he did not offer concrete ways to execute these ideas. So, after several unsuccessful attempts, I realized that trying pixel to pixel comparisons using statistical correlation would not suffice and settled on using an SVM that would take in the bitmaps and the classifications. After reading the sklearn documentation on SVC, specifically LinearSVC, I started preprocessing the X and y numpy ndarrays so that svc.fit(X, y) would set the best values for the SVC instance.

3 Accuracy

Cross Validation - Variations on k				
k-fold	No. Right Predictions	No. Wrong Predictions	Total	Accuracy
20	362	38	400	90.5%
10	348	42	400	87.0%
n	383	25	408	93.9%

* Accuracy was calculated by right predictions divided by total predictions.

To try to improve the accuracy, I tried varying the level of blur of the images and using these new bitmaps in my validation method. Sigma is the variance of the Gaussian kernel applied to the bitmap using numpy's ndarray gaussian_filter method. Conclusively, a sigma between 1.5 and 2 resulted in the most accurate predictions.

Cross Validation - Variations on sigma (blurring coefficient)				
sigma	No. Right Predictions	No. Wrong Predictions	Total	Accuracy
0 (no blur)	350	60	410	85.4%
1	352	58	410	85.9%
1.5	353	57	410	86.1%
2	353	57	410	86.1%
2.5	352	58	410	85.9%
3	352	58	410	85.9%

4 Conclusion

This program was very exciting between trying to load all the files dynamically in separate folders, learning about how to shape multidimensional numpy arrays to fit a format for a library, and watching a program learn as if it was a child. If I had more time, I would have experimented with other kernel functions like rbf and played with the C and gamma fields to improve my accuracy. I found a very interesting research paper (1) on the use of bitmaps and SVM and would like to have improved my SVM implementation using his research. Namely the results of using different kernel functions on bitmaps perform from worst to best in order: Linear, Polynomial d=2, Polynomial d=5, and RBF. Although the RBF kernel gave my program *0%* as opposed to *92%* using the Linear kernel, I imagine the right gamma would push my program over 95% accuracy. This program was a fun and extremely educational project, teaching me not only how to decipher and make valuable use of research, but also trying to incorporate machine learning into a tangible, working program.

(1) http://olivier.chapelle.cc/pub/intern98.pdf