Intro to AI: Image Recognition Project

Daniel Eppinger

May 4, 2016

At the start of the project, I had no idea how to begin the assignment on image recognition. I knew I wanted to code in Python, since that is the language I had become most familiar with, and Python would most likely have some extremely helpful libraries available. I started out with a simple Google search, which led to the creation of a working version for the project. After small amounts of testing, I realized that as much as I was proud of the work I had done, I needed to create something with more accuracy. I then switched to another method that was mentioned during class, which helped me to complete the project in a completely different way. The second version is the approach I decided to keep, as it is more efficient, accurate, and easier to use than the preceding version.

1 Approach

I ended up having two different approaches to the project. The first approach, which I have named pixel-by-pixel analysis (PPA), is particularly straightforward, but was difficult to program. The second approach, which uses a support vector machine (SVM), is far more complicated, but was much easier to program. As a result of this, the PPA approach was more interesting to code since most of it was from scratch; however, the SVM approach resulted in a more useful program.

1.1 Pixel-by-Pixel Analysis

I discovered this approach after searching the Internet for how to do image recognition in Python. One of the first links was a tutorial, which I followed to the end. Subsequently, I modified the code from the tutorial to fit the assignment I was given. Seeing as the tutorial had no name for the basic approach to image recognition it used, for the purposes of this report, I gave it the aforementioned name.

Pixel-by-pixel analysis works by comparing each individual pixel of a given image to the pixels of the images within the dataset. Before this, each image is put through a threshold function that reduces the image to pure black and pure white, no colors in between. Images in the dataset are then stored in a file so they can be easily used in subsequent program executions. Each image in the dataset falls into a category that has been predetermined by the user. The provided image is classified as the category with which it has the most pixels in common with. This is how the tutorial performed this approach; however, for the purposes of this assignment, this method was far too slow and vastly inefficient.

I improved upon this method by further reducing each category of images to a single matrix. Originally, each image was represented by a matrix, with each pixel as an array of values corresponding to the pixel's value. I reduced this so each pixel in an image is now represented by either a 1 or 0. A 1 means that pixel is black, and a 0 means that pixel is white. To simplify this further, I combined all the images into one matrix such that what originally represented a pixel is now a pair of numbers. The first number is equivalent to how many images in the category have a white pixel at this spot, and the second number is equivalent to how many images in the category have a black pixel at this spot. Each category matrix was saved to a file so they could easily be retrieved on subsequent runs. To compare the given image to the images in each category, the program checks each pixel from the given image to see if it is black or white. Then, based on the answer, the program retrieves the value for the corresponding pixel and color for the current category. It does this for each pixel, and sums the values. Therefore, at the end, the program has five values, one for each category. Each value is then divided by how many images were in it's corresponding category, so as to get an average value that is comparable to the other values. The given image is determined to fall in the category which has the highest value.

1.2 Support Vector Machine

I decided to pursue the SVM approach due to the low accuracy and efficiency of the PPA approach. The entire approach is based around the use of Python's support vector machine library, specifically the SVC (support vector classifier) function. The SVC function creates a support vector machine using the provided features and labels. Afterwards, a separate function can be called with a given value and the machine will predict which label it falls under. I modified a few of the functions from the PPA approach to be used for the SVM.

The threshold function is the only function that was significantly changed. Similar to PPA, the threshold function reduces an image to 1s and 0s; however, instead of a matrix, it is just an array, so there is no way to distinguish between rows and columns. Also, the function appends the array to a list of all the image arrays, as well as appends the name of the category to a separate list. These two lists are the features and labels provided to the support vector machine, respectively.

After the feature and label parameters are created, the support vector machine is initialized, and then trained using the features and labels. For the initialization of the machine, I used linear as the kernel type and 1.0 as the C value. These were listed as the standard values, so I decided to start with them, and because they worked well, I just left them alone. Afterwards, the support vector machine is pickled so that it may be easily used in future image recognition attempts. Finally, a given image is put through the threshold function, and is given to the machine, which returns the label which it believes the image would fall under.

2 Accuracy

For testing accuracy, I removed the last ten images from each category and used those fifty images as the input for each approach. The PPA approach did surprising well when it came to identifying images; however, it definitely struggled with some of the categories. The SVM approach also did very, and it far surpassed what PPA was able to accomplish.

2.1 Pixel-by-Pixel Analysis

The PPA approach had an overall accuracy of seventy percent. It missed two smiles, one hat, six hashes, five hearts, and one dollar. I found the results to be pretty surprising, because I did not expect the program to work that well. Pure random guessing should provide an accuracy of about twenty percent, and this program provided a much higher accuracy, so I was quite happy with the results. Still, the accuracy is not as high as it could be if another approach was used, which is why I decided to use the SVM approach instead.

2.2 Support Vector Machine

The SVM approach, across the fifty test trials, had an overall accuracy of ninety percent. It missed one hash, three hearts, and one dollar. Clearly, it seemed to struggle most with hearts. This was somewhat surprising since hearts are simple shapes, but it would make sense how this could cause confusion between them and more complex shapes. The program mistook them once as a hash and twice as a smile. The heart certainly seems most similar to a smile, given the shape of each; however, it does not seem similar to a hash, so I was surprised to see that mistake occur. Provided a larger training set, the SVM program would certainly have a much higher accuracy as it is able to better identify the differences between the categories.

3 Conclusion

Overall, the SVM approach clearly dominated in terms of performance and efficiency compared to the PPA approach. On the other hand, this was to be expected, since PPA is a very basic implementation of image recognition, and an SVM is far more complex, though still relatively simple compared to other techniques. It seems like the only situation where the PPA approach would be used over the SVM approach is in a class setting where students are being taught about basic image recognition; however, the SVM approach definitely seems usable in a larger variety of situations. The situations would either be where the data can be separated by a line, or where accuracy is not of high importance.