Images-Python

Patrick Seitz
Computer and Information Sciences Department
Indiana University South Bend
South Bend, USA
pseitz@iu.edu

***Abstract* — In this experiment we practiced image processing, computer vision, and knowledge-creation to classify MNIST pictures of handwritten digits 0-9 using Python with NumPy and Scikit Image. Frist, we used a set of features to develop an algorithm that would be able to test images and tell if they match. We also practiced inverting images using only the NumPy library.**

Keywords* — *Artificial Intelligence, NumPy, MSE, Binary Image

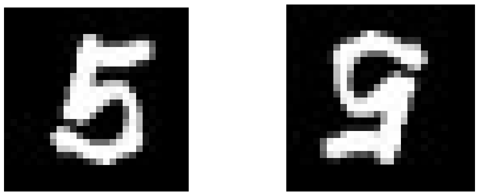
I. Introduction

To create Artificial intelligence (AI) one must first develop an algorithm to base it on. In this experiment a set of features were discovered that allowed the creation of a primitive AI capable of detecting images passing them or failing them based on a specified threshold.

II. Experiment

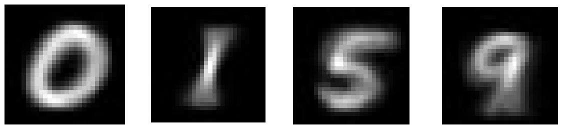
A. Inverting Image

First, before anything could be done 3 file variables were created holding different directories. This was done in order to reference multiple images when processing them in functions that will be described later. To do this "glob" was used to create file variables. One of the variables "fliplist" which contained one image of each number. This was then processed in a function called "def images_flip()". This function used a for each loop and flipped each image using np.flip and displayed it to the screen.



B. Composite Image

For the rest of the experiment, we will be comparing test images to a base image. A base image is a known good image template to test upon. In order to make the base image a composite image was made and then turned into a binary image. To make the composite image the function "def composite image" was used. The function takes the file list variable "dirlist" and computes a summation of all the digits of the same type. When finished the images were saved to a directory to be used in further functions. Here are some examples of the composites images that were created. The parts of the image where it appears to be whiter occurs because other images were in the same spot as the summation was computed. This means those areas were most similar among the images of each digit.

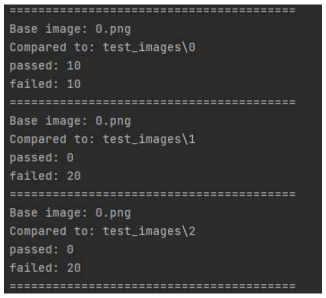


C. Image Learn and Test

Since we now have a base image to test upon, we can start to learn about the general performance of how well the test images compare to the base image. To test for comparison, we used Mean Square Error (MSE). MSE is the average squared difference between the estimated values and the actual value. This allows us to test how close an image is to one another. To use MSE SciKit Image was used as they have a MSE function under "skimage.metrics". The function outputs a floating point number "x". When "x" is equal to 0 they are the same image. As "x" increases in the positive direction the more dissimilar the images are from one another. Using this method comes with its drawbacks as we will discuss further in the analysis section.

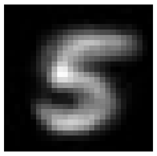
Furthermore, 80% of the test images where used to learn about the general performance of the testing. Metrics such as the max, sum, and average were recorded for the MSE of each digit. This information was used in the function called "def image_learning()" At the end of the test the averages were compared and a threshold of 0.13 was chosen as it seemed to be the most viable number to get what we want. If the result was below 0.13 the test passed, if it was above it failed.

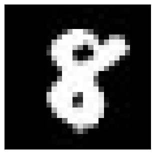
When the threshold of 0.13 was discovered, it was time to use this information and test the base images with the last remaining 20% of test images. The function to compute the results is called "def image_tester()". As described earlier each base image was tested by each number, 20 times for each number. This resulted in each base number being tested 200 times in total. Here is a sample picture of results that were recorded for a base digit of 0.



```
========================================
Base image: 0.png
Compared to: test_images\0
passed: 10
failed: 10
========================================
Base image: 0.png
Compared to: test_images\1
passed: 0
failed: 20
========================================
Base image: 0.png
Compared to: test_images\2
passed: 0
failed: 20
========================================
```

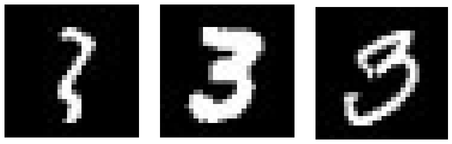
In the previous picture we can see that the results leave more to be desired, with an average 50% pass rate for when comparing to the same number type. This 50% pass rate was common among all other base image tests. A large amount of false positives were also registered. Some numbers that were similar in shape were more likely to register false positives than others. For example, a 5 and an 8 have the chance of registering a false positive as they can look similar.





III. ANALYSIS AND THOUGHTS

When using this method of MSE it was not sufficient in order to properly computer the similarities between images. With further reading MSE seems to work very well with images that are aligned exactly. In our data all our images were not aligned completely as each digit had a different shape. For example, consider these pictures of threes.



Continually, creating a composited image as the base image did help increase the accuracy of the result, but was not enough. Further processing in the test images will be needed to correct this issue for MSE to work properly.

Going forward, if I were to change my approach, I would explore the use of (SIFT) feature detector and descriptor extractor located in “skimage.feature”. With the use of (SIFT) it can draw upon similarities even with images that are drastically distorted from one another. This would solve the problem of images that are not aligned properly, thus increasing the likely hood of favorable results.