ASL classification with Graham Scan and Pixel Comparison

Connor Studebaker
CPSC 450
Fall 2024

Summary

This project tackles the problem of correctly identifying the yes and no symbol in American Sign Language (ASL). The 2 algorithms used were a simple Pixel Comparison algorithm [1,2] (naive), and a Convex Hull algorithm based on the Graham Scan algorithm [3,4] (complex). These algorithms were implemented using python due to ease of use, and the easy integration of the OpenCV module [6] used for taking in a picture input. The 2 algorithms were rated on their accuracy of correctly identifying the ASL gesture and the time it took to run the algorithm. The naïve pixel counting algorithm was able to perform the classification almost 10x faster than Comple Convex Hull algorithm depending on the image but the Pixel Comparison algorithm lagged behind the Convex Hull in complexity and robustness.

1. ALGORITHM SELECTED

The problem for this project was to correctly identify the ASL sign of yes or no on a blank white background. The ASL yes sign resembles a fist, while the no sign is what can be described as a pinching symbol (Figure 2 & 3) The naïve approach used a Pixel Comparison algorithm where you simply counted the pixels of the sign. The fist being bigger and closed allowed for more pixels to be counted vs no sign which has a gap between the thumb and index finger. This algorithm isn't based on any major preexisting algorithms, it's a simple naïve amalgamation influenced by [1,2], a straightforward algorithm to count pixels in an image.

Pseudocode – Pixel Comparison
Input: A gesture with a white background
Result: image either contains yes, no, or invalid
begin

```
1 convert image to greyscale
2 threshold the image
3 count white pixel in image
4 classy pixel count
```

end

Complexity Analysis – Pixel Comparison
O(n), where n is the total number of pixels in the image. The algorithm performs a single pass through the pixel values.

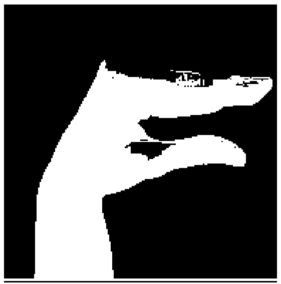
Figure 1: Threshold Image of a No symbol

The complex approach constructs a Convex Hull of the input gesture using the Graham Scan algorithm [3,4] then compares it to the Convex Hull of the training data. If the algorithm is unable to classify the gesture a second method is then used where the distance of the bottom right hull point is measured and depending on the length a gesture is determined. This can work because the convex hull of a no symbol includes the distance from the tip of the thumb to the base of the forearm. This will almost always be larger than that of the yes symbol, which is just the side of the hand. So based on the distance between the two points the image can be classified, a visualization of this can be seen comparing figure 2 and 3 to each other.

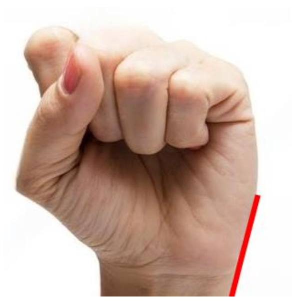
Figure 2. ASL yes symbol

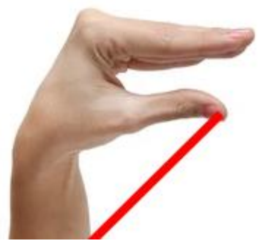
Figure 3. ASL no symbol

Pseudocode – Convex Hull
Input: A gesture with a white background
Result: image contains yes, no, invalid, or leans toward yes or no symbol
begin

```
1 Train Convex Hull data, preprocess images
2 for images in training data
3     Blur image
4     Threshold image
5     Identify contours and points
6     call Graham Scan to build convex hull
7 Redo steps 3-6 for input image
8 compare input image hull to training hull
9 if no gesture run 2nd classification
10  find rightmost hull distance to classify
```

end

Pseudocode – Graham Scan
Input: Contour points
Result: Convex Hull
begin

```
1 sort points by y-coordinate
2 start_point = points[0]
3 sort points by polar angle
4 stack=[start,points[1],points[2]]
5 for i from 3 to len(points):
6   while(stc)>1&&stc[-1],pnts[i])!= X:
7     stack.pop()
```

```
    8   stack.push(points[i])
    9 return stack
X = cross product
```

Complexity Analysis – Convex Hull with Graham Scan

Sorting the points has a complexity of O($n$log$n$) this is from the built in python sort function. Scanning and constructing the hull has a complexity of O(n) because it has to iterate through each point once. Finally for the Hull comparison it has a complexity of O(n) this is by calculating the distance between all the points which is a linear operation. Comparison also interpolates the hulls to the input image to match the test image, it only compares with the points in the Hull which is a linear operation. This brings the total complexity of the algorithm to O($n$log$n$).

2. IMPLEMENTATION

Both algorithms were implemented using python due to past familiarity with using image processing and utilizing OpenCV in python [6].

In the Pixel Comparison OpenCV was used for image preprocessing such as grayscale conversion and thresholding (thresholding is when an image is converted into binary of either black or white values). NumPy arrays were used to hold pixel values [7]. The main change from the pseudocode was a timer added to track how long it takes for the algorithm to finish. Pixel Comparison uses the following libraries: CV2 for image handling, numpy for data handling, and time for tracking completion time. The main challenge faced was handling shadows and dark spots on the gesture this led to complications with thresholding and converting the hand part to white pixels. Since not all of the hand would be able to be counted and classified.

For the Convex Hull algorithm after the image was preprocessed the contour points were saved in a numpy array [7], and the convex hull uses a stack just like the Graham Scan specifies [3,4]. All calculations for polar angles and cross products uses arrays to handle data. The main deviation from the pseudocode is the filtering out of garbage data, such as blank images with no gestures. Interpolation was also added to normalize hull size when comparing hulls. The biggest challenge was adding the interpolation, I believe that its implemented correctly but its complex and hard to understand. That also stands for the polar angle geometry needed in order to implement the graham scan when having to sort points.

A series of tests was developed to test the implementation, evaluate its robustness, and ensure performance. The following table 1 summarizes the unit test cases and results:

Test Name	**Purpose**	**Input Description**	**Results**
Basic Functionality	Ensures algorithm can accept input file	a valid jpg file	Pixel Count ✓ Convex Hull ✓
Blank white image	Correctly filters out invalid gesture	Empty white jpg file	Pixel Count ✓ Convex Hull ✓
Blank black image	Despite only being implemented with a white background in mind, this test handles unexpected behavior	Empty black background jpg file	Pixel Count ✓ Convex Hull X
Yes symbol with no shadows	In perfect lighting and conditions correctly classify the image	Picture of fist with white background	Pixel Count ✓ Convex Hull ✓
Yes symbol with some shadows	With shadows in play correctly classify the image	Fist with some part darkened	Pixel Count ✓ Convex Hull ✓
No symbol with no shadows	In perfect lighting and conditions correctly classify the image	No symbol with white background	Pixel Count ✓ Convex Hull ✓
Peace gesture	Tests a peace gesture to see if gesture will be classified as invalid	Peace gesture with white background	Pixel Count X Convex Hull ✓
Cat Picture	Tests if able to filter out invalid image	Cat on a white background	Pixel Count X Convex Hull X
Yes symbol with darker skin tone	Tests if classify gesture with dark skin tone	Yes gesture with darker skin tone	Pixel Count ✓ Convex Hull X
Out of center yes symbol	Tests whether the gesture can correctly be identified if its not centered in the image	Yes symbol skewed to the right	Pixel Count ✓ Convex Hull ✓

Table 1: unit tests completed

The following tests were required to be passed in order for the algorithms to be deemed completed, 1,2,4,5,6. Tests 1 & 2 ensured that OpenCV was working correctly and that both algorithms were able to detect if there was an object present in the image. Tests 4, 5, & 6 were detrimental to the accuracy of the algorithms, they were made to test the accuracy of the classification. Test 10 was also important because it helped test the robustness of both algorithms; by having the symbol out of the center it tests the Graham Scan's ability to build a convex hull when points differ in the location from the training data. To run the unit tests for yourself simply select the image you would like to test from the `/testing_data` folder then replace the image name in the python file. For `PixelComparison.py` it is located on line 70 and for `ConvexHull.py` it is located on line 202. Both files also include the `argpasre` library this is used for the turning on the performance test which is discussed in detail in section 3.

3. PERFORMANCE TESTS

The performance test was conducted by having the algorithms attempt to classify 10 yes gesture images, 10 no gesture images, and 5 invalid images. This was then timed and rated on the accuracy of the classifications (see results figure 4). The performance testing was run on my local Windows 11 machine (i7-10750H 16GB ram). The exact timing data is relevant to my machine only but the trends you will find are universal. In order to run the performance test for both algorithms simply add "`--performanceTest 1`" in the console, this will automatically run the performance test and output the data into the console. The performance graph in figure 3 does not generate by itself and instead is made by me in excel based on the results from the performance test. The following will run the performance test for Convex Hull to run, for Pixel Comparison change file name to `PixelComparision.py`, of course make sure you are within the correct folder.

```
python ConvexHull.py --performanceTest 1
```

4. EVALUATION RESULTS

The results for the performance test described above were not that surprising due to how the test was structured. With limited "trick images" and ideal conditions the Pixel Comparison algorithm was suspected to be more accurate due to the complexity of correctly building the convex hull vs simply counting white pixels. The Pixel Comparison algorithm had an accuracy of 84% for the performance test images vs a 72% accuracy for the Convex Hull. The accuracy results of the performance test can be seen below in table 2. You can clearly see that the Pixel Comparison scored higher on classifying on both the yes and no gesture, this is most likely due to how the Convex Hull compares its training data to the input image. If the Convex Hull doesn't meet a certain accuracy the algorithm will be unable to correctly identify the image. Since hands vary in size, it's more difficult to correctly identify the gesture. This also explains why Convex Hull scored higher for the invalid gesture because the invalid gestures in the performance test looked nothing like a hand so naturally the Convex Hull wouldn't match.

	Pixel Comparison	Convex Hull
Yes gesture	80%	60%
No gesture	100%	80%
Invalid gesture	60%	80%

Table 2: Accuracy results of performance test.

Looking at figure 3 below we can see that the Pixel Comparison algorithm was able to finish the performance test 2x as fast compared to the Convex Hull algorithm. This timing difference was to be expected after taking their time complexity into account with Convex Hull being $O(n \log n)$ vs Pixel Comparison being $O(n)$. Timing duration between images varied slightly due to how both algorithms scale with image size, so larger pixel count images take longer to process. The images in the test were all similarly sized to decrease complexity for building the convex hull.

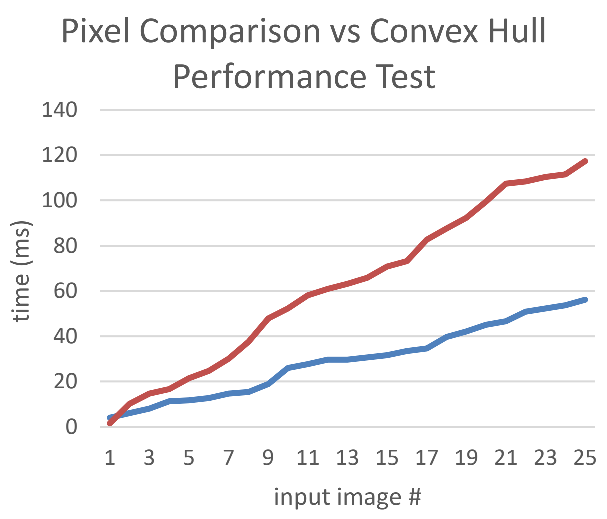

Figure 4: timing results for performance test (red is Convex Hull and blue is Pixel Comparison)

The performance test data heavily favored the Pixel Comparison, having a white background and most images having ideal lighting, this doesn't display a real-world scenario. The Convex Hull algorithm is more effective in a real-world scenario due to how its able to blur the background image and focus on the points of the hand to build a Convex Hull. If given a real-world scenario the Pixel Comparison algorithm would grayscale the entire image, threshold it then count the white pixels. Background artifacts would render the algorithm useless because it would count the white pixels from them messing with the classification.

5. REFLECTION

The research conducted for this project highlights how it is both interesting but redundant and useless. No one uses Convex Hulls for image processing anymore, it's all computer vision and machine learning now. I could've had better results and less stress if I simply built a Neural Network in Tensor Flow to classify the images. But that's what's fun about these projects you get to learn about something new and hopefully apply it one day. I've never worked with contours or convex hulls before, but I'm very comfortable with polar math from when I took circuits and signals & systems. Being able to apply those skills in CS class was not something I expected, though it still was a challenge with implementing the entirety of the Graham scan algorithm because of the complex math. I was able to overcome this by consulting with my peers and touching up on my polar math knowledge.

The Pixel Comparison and Convex Hull algorithms are both P-TIME. There's no exhaustive or non-deterministic guessing that would make it NP-TIME, they both use structured and deterministic computations such as sorting and linear traversal.

If I had more time for this project I would like to improve the accuracy of the Convex Hull algorithm. I would do this by first adding more training data so that the convex hull shape can fit more sizes of a fist and pinching gesture. Then I would work on the accuracy of the convex build from the data points. The inaccuracy in the convex hull was why I had to come up with the secondary method of comparing the rightmost hull distance. If the convex hull was more accurate then the no symbol would not have a line from the thumb to the forearm. Instead, it would have multiple lines following the thumb to the base of the hand then finally to the forearm.

6. RESOURCES

For the Pixel Comparison algorithm, I used both [1,2] as resources. [1] explores various algorithms for pixel-based image processing, including techniques for finding regions and connected components. It provides pseudocode examples that were useful when implementing the Pixel Comparison. Shih's book [2] focuses on pixel classification and pattern recognition, both of which were crucial for the Pixel Comparison algorithm. The sections on pixel intensity analysis and thresholding informed the methodology for comparing pixel values in the naive algorithm. This resource also supported the testing phase by providing insights into expected outcomes for different patterns.

When implementing the Convex Hull algorithm [3] was pivotal for understanding the Graham Scan algorithm, a key component of the convex hull implementation. Without it I wouldn't have been able to build the convex hull from the contour points, it provided the grounding for the whole convex hull algorithm. This source [4] provided a practical introduction to applying convex hull algorithms to binary images for shape analysis. While the examples were in MATLAB, they were easily adapted to Python, specifically for identifying the convex hull of hand contours. The examples from this source clarified how convex hulls could be used for image classification tasks and supported debugging the algorithm during development. The last resource used for the Convex Hull algorithm was [5], it helped with the implementation of the Graham Scan algorithm and provided pseudocode which acted as a jumping off point. The comment section under the video also was informative will solving small edge cases, this resource was without a doubt the most helpful due to how eases the complexity of implementing Graham search.

The OpenCV library [6] was used extensively for image preprocessing, such as thresholding, and contour detection. The documentation provided essential guidance on using OpenCV functions efficiently in Python. Functions like `cv2.findContours` and `cv2.threshold` were directly applied based on the examples provided. The NumPy library [7] was critical for performing mathematical operations, such as calculating Euclidean distances, which were used to compare convex hull shapes. The documentation provided clear explanations of array manipulations, which were essential for working with point sets and hull data.

7. REFERENCES

[1] **Birchfield, S. T.** 2017 *Image Processing and Analysis*. (1st ed.) CENGAGE, Boston, MA

[2] **Shih, F. Y.** 2010 *Image Processing and Pattern Recognition: Fundamentals and Techniques* (1st ed.) Wiley, New York, NY

[3] **Gaham R. L.** 1981 *Finding the Convex Hull of a Simple Polygon.* Stanford Technical Report CS-81-887. Standford University, Palo Alto, CA

[4] **Eddins S.** 2011 Binary Image Convex hull – algorithm notes (October 2011). Retrieved November 15, 2024 from https://blogs.mathworks.com/steve/2011/10/04/binary-image-convex-hull-algorithm-notes/

[5] **Stable Sort.** (2020). Convex Hull Algorithm – Graham Scan and Jarvis March tutorial. Video. (April 2020). Retrieved November 25, 2024 from https://youtu.be/B2AJoQSZf4M?si=aolY_c0ByVtfLyXF

[6] **OpenCV. Open Source Computer Vision.** Retrieved from https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html

[7] **NumPy.** Retrieved from https://numpy.org/doc/2.1/reference/generated/numpy.squeeze.html