CSI 4106 RealClimb - : Rock image classification for attempt at optimal climbing route prediction

Final Report

Group 38

Cheryl Tollola/8317298
Xiuzhu Li/8571645
Suzie Oh/300151071

Table 1: Time spent on Project per team member

Team member	Task	Time spent	Challenge
Suzie Oh	Dataset research (which classes can we define), manual data collection	3 hours	Many sources to filter through: -Moonboard -Investigated rock climbing crowd sourced data of routes on phone apps and websites
	Further dataset collection	4 hours	Learned Bing Blind Image Search API -Searched with several queries ie: “ Bread Loaf Climbing Pinch Holds” and “Climbing Pinch Hold Slug” -Ran the search about 15 times with a result of 50 - 75 useable images per run -Manually deleted unusable images which required knowledge of rock climbing hold types (had to be learned)
	Feature vector research and collection for KNN	5 hours	New to image processing with OpenCV, spent too many hours researching and fiddling with minimum bounding rectangles and circles, looking at identifying blobs for holes in rocks, simple shape detection, centroids.
	Project research	2 hours	To find tutorials for KNN and CNN and OpenCV
Cheryl Tollola	Cropping images from Bing Blind Image Search results to one rock in an image	3 hours	There were 6 classes and 648 photos, they were cropped, rescaled and renamed.
	CNN	8 hours	Prepare the jupyter notebook for cnn training
Xiuzhu Li	Dataset research (which classes can we define), manual data collection	3 hours	Manual images found from various sellers of indoor rock climbing holds
	KNN	7 hours	Convert the tutorial using the iris dataset to use rock feature vectors

Introduction

The problem for our project is to classify an image of an indoor rock climbing "hold" also known as a "rock" into 6 different classes: volume, sloper, pinch, crimp, pocket, and jug. There is limited research into the classification of rocks to predict how an athlete should climb a route in indoor Sport Climbing.

Most of the data from previous research that related AI and rock climbing used Moonboard which only contained data of how an overall wall is difficult with certain routes on the re-configurable board using a limited and known list of rocks, and no images of the actual rocks themselves to classify these 6 classes of rocks in a new setting.

The two main sources of research our group found were on topics quite different from our goal:

1. *Optimizing climbing routes:*
   http://cs229.stanford.edu/proj2017/final-reports/5232206.pdf

2. *Strange Beta: An Assistance System for Indoor Rock Climbing Route Setting Using Chaotic Variations and Machine Learning*
   https://arxiv.org/pdf/1110.0532.pdf

The algorithm will be partially used in an application for the Tokyo Olympics 2020 Open Innovation Challenge focusing on the category of "Sport Climbing".

Sports Climbing combines 3 disciplines: speed climbing where a competitor climbs as high up a 15m wall as fast as they can, lead climbing where a competitor must climb up a 15m wall with difficult rock holds within a time period with a rope, and bouldering where a competitor must climb a 4m wall with difficult hand holds without a rope.

We wanted to understand if image classification with machine learning algorithms such as the K-nearest neighbours or Convolutional Neural Networks would produce a usable result to identify rocks on one of the walls rock climbers will compete on in the Olympics to teach the audience rock climbing techniques.

Dataset description / Features & Feature engineering

The dataset features photos of 6 types of indoor rock climbing holds: jug, pinch, crimp, pocket, volume, sloper.

The about 40 images of our data came from internet searches for suppliers of indoor rock climbing holds that were then cropped and manually and had the background or things other than the rock removed using https://www.remove.bg/.

- The official partner and supplier of Sport Climbing in the Tokyo 2020 Olympics: https://www.entre-prises.us/
- Other hold sellers:
  https://www.bluepill-climbing.com/products/competition-holds/
  https://holdtopia.com/brands
  https://www.metoliusclimbing.com/climbing_holds.html
  https://www.elementclimbing.com/

We also used Bing Image Search API using our python program to grab more images per class with multiple. We ran the search about 15 times with a result of 50 - 75 useable images per run. After querying, we had to manually delete images where the main hold of interest was too small, hidden on a wall with others attached to it, had a hand blocking it.
With these two sources we had 108 images per class in one folder per class, so there was no separate test and train sets.

Feature engineering

For the KNN, we attempted to grab feature vectors using OpenCV:
https://hub.packtpub.com/opencv-detecting-edges-lines-shapes/
https://docs.opencv.org/3.4/de/d62/tutorial_bounding_rotated_ellipses.html
Which is computer vision software to calculate features and output them to a CSV file such as calculating the minimum bounding rectangle or circle of each hold per image.

Our ideal candidate feature vectors were:
1. Number corners (important for volumes)
2. Number of Flat edges (important for crimps and volumes)
3. Degree it sticks out from wall (important for slopers, jugs and volumes)
4. Max length of 1 dimension is bigger than
max length of other dimension (important for pinches)
6. Has holes that are not the ones for bolting onto a wall (important for pockets)

We ended up identifying that jugs might also have “handles”, where a climber’s fingers could wrap around the rock, whereas for a pocket, they would go inside a hole.

The OpenCV method worked to find the contour, simple shapes, and bounding box of an image, greyscale histogram, but was not viable for the other feature vectors we wanted.

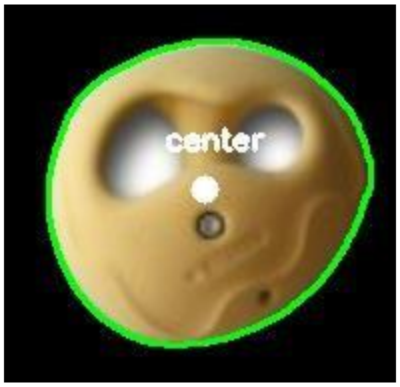


Figure 2: Picture of a contour with CannyEdge detection

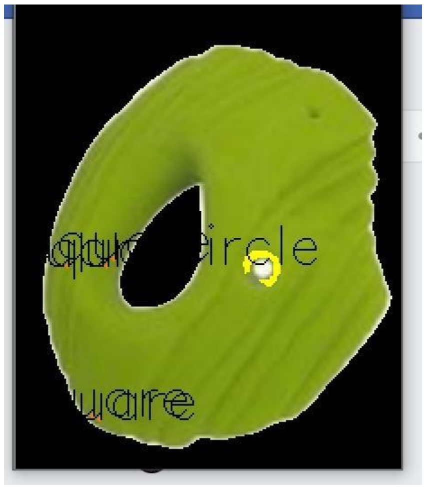


Figure 3: Picture of simple shape prediction

But only and we were running out of time to make a dataset for the KNN, so we manually made a csv file with 15 samples annotated from the images we found, and 7 feature vectors per sample.

1. Number of holes
2. Number of corners
3. Number of edges (number of not round edges)
4. Positive (if it stuck out of the wall more than an average rock, the height of the rock was sizeable)
5. Circular or rectangular (if a bounding rectangle or bounding circle to the edges of the shape would contain more of the shape
6. Length bigger than width approximately (also known as inertia in OpenCV)

picture number	rock type	number of holes	number of corners	number of not round edges	positive (y/n)	circular or rectangular (c/r)	length bigger than width approx (y/n)	handle (for jug) (y/n)

Figure 4: Headers in the dataset .csv file from manual finding of feature vectors

Details of the methods used

KNN

a. First of all, the K-nearest Neighbors (KNN) calculates the Euclidean Distance between every training set and testing set.
   Then it gets the Nearest Neighbors of each testing set like below.
   1. Sort all of the records in the training dataset by their distance.
   2. Select the first 'K' entries to return as the most similar neighbors

   Then it makes predictions with the most similar neighbors from the training dataset.
b. We choose KNN because it is simple and easy to implement compared to other algorithms. Also, it relies on how well the feature vectors define the rocks, which is important to see if this type of project can produce an accurate prediction of new rocks not seen in the data set, and to see if it is viable for a machine learning application.

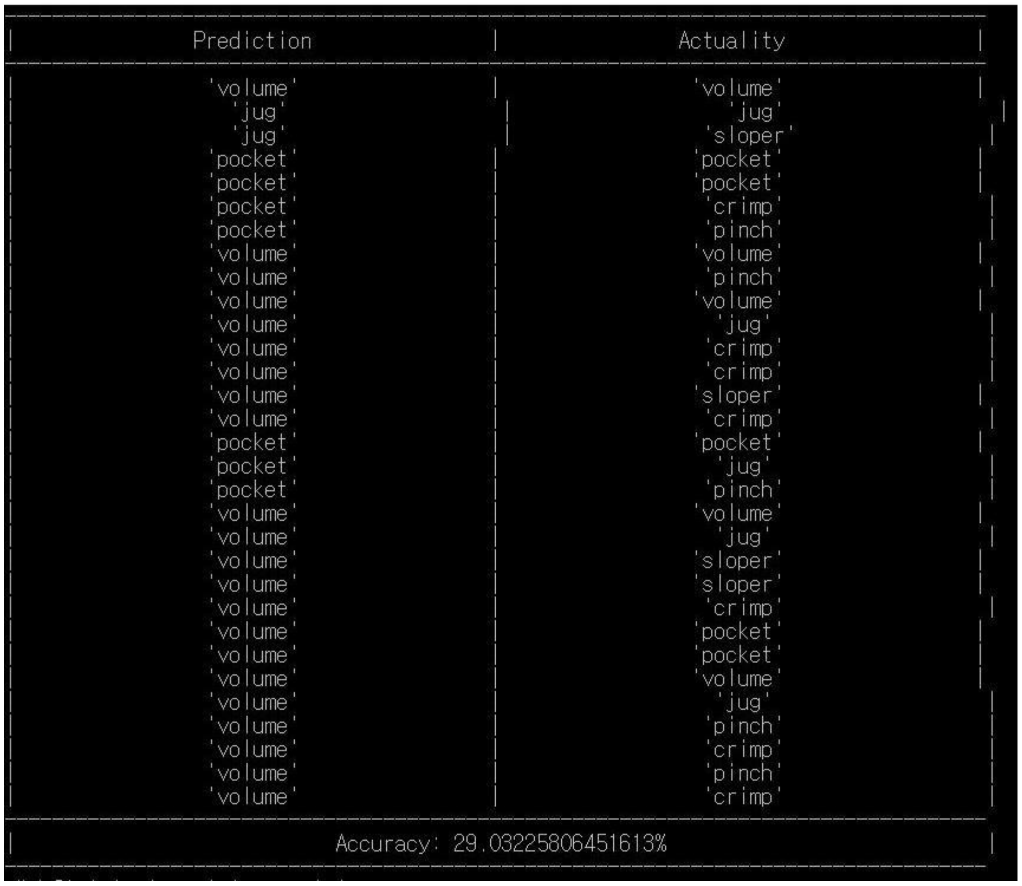
CNN

a. The method used in this section is called convolutional neural network training. In image classification, it is easier to throw all the data into a training network than classifying all the feature vectors manually.
b. We choose CNN because it is a widely used and commonly approved way of AI recognition method. We can find plenty of examples in kaggle.com and learn from those fairly quickly.
c. Preparing the dataset caused a lot of pain. Since the topic is very narrow, we cannot find pre-existing data that's been circling in kaggle or anywhere else, instead, we have to find the data by ourselves. After cropping, renaming and formatting, the data was finally usable. I chose jupyter notebook to write the notebook, as we already know how to use it. Library wise, I chose keras. Setting up the training model was not a big deal, after some minor bugs, it runs smoothly. However, in the end the CNN code went into trouble that I could not be able to fix. It was a bug complaining in the fit model section.

Overall

Individual results

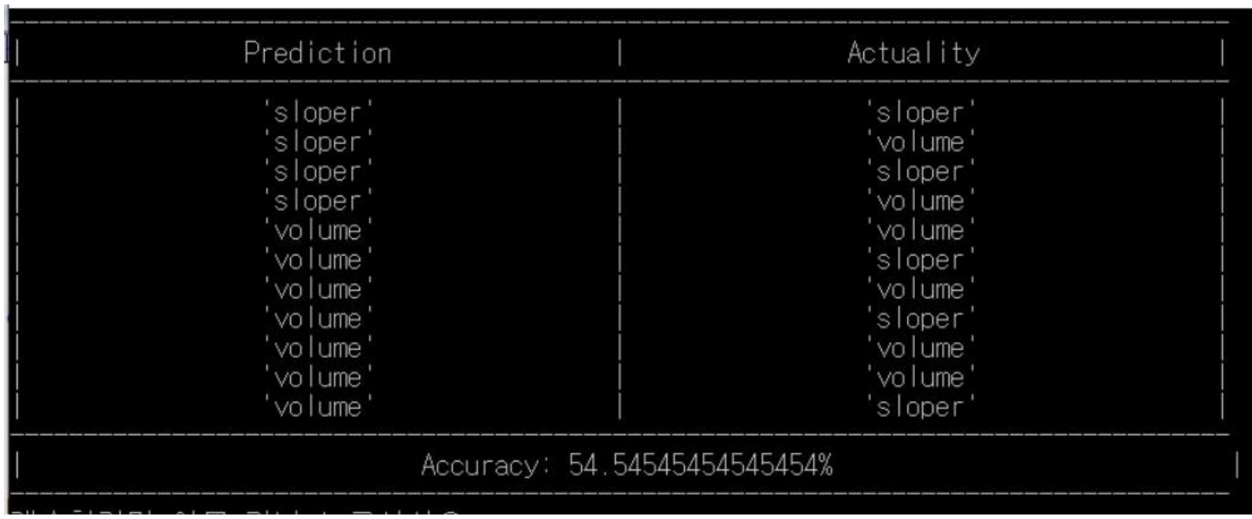
KNN



```
-----------------------------------------------------------------------------------
|              Prediction              |              Actuality              |
-----------------------------------------------------------------------------------
|               'volume'               |               'volume'              |
|                'jug'                 |                'jug'                |
|                'jug'                 |               'sloper'              |
|               'pocket'               |               'pocket'              |
|               'pocket'               |               'pocket'              |
|               'pocket'               |               'crimp'               |
|               'pocket'               |               'pinch'               |
|               'volume'               |               'volume'              |
|               'volume'               |               'pinch'               |
|               'volume'               |               'volume'              |
|               'volume'               |                'jug'                |
|               'volume'               |               'crimp'               |
|               'volume'               |               'crimp'               |
|               'volume'               |               'sloper'              |
|               'volume'               |               'crimp'               |
|               'pocket'               |               'pocket'              |
|               'pocket'               |                'jug'                |
|               'pocket'               |               'pinch'               |
|               'volume'               |               'volume'              |
|               'volume'               |                'jug'                |
|               'volume'               |               'sloper'              |
|               'volume'               |               'sloper'              |
|               'volume'               |               'crimp'               |
|               'volume'               |               'pocket'              |
|               'volume'               |               'pocket'              |
|               'volume'               |               'volume'              |
|               'volume'               |                'jug'                |
|               'volume'               |               'pinch'               |
|               'volume'               |               'crimp'               |
|               'volume'               |               'pinch'               |
|               'volume'               |               'crimp'               |
-----------------------------------------------------------------------------------
|                     Accuracy: 29.03225806451613%                              |
-----------------------------------------------------------------------------------
```

Figure 5: Result of the KNN predicting all 6 classes

The KNN had about 29% accuracy, even worse than guessing. If the dataset had around 1000 images per class and more useful feature vectors, we could expect a better result that our 15 samples per class. This would have been possible if OpenCV methods used were more accurate.



```
--------------------------------------------------------------------------------
|               Prediction               |               Actuality              |
--------------------------------------------------------------------------------
|               'sloper'                 |               'sloper'               |
|               'sloper'                 |               'volume'               |
|               'sloper'                 |               'sloper'               |
|               'sloper'                 |               'volume'               |
|               'volume'                 |               'volume'               |
|               'volume'                 |               'sloper'               |
|               'volume'                 |               'volume'               |
|               'volume'                 |               'sloper'               |
|               'volume'                 |               'volume'               |
|               'volume'                 |               'volume'               |
|               'volume'                 |               'sloper'               |
--------------------------------------------------------------------------------
|                        Accuracy: 54.54545454545454%                           |
--------------------------------------------------------------------------------
```

Figure 6: Prediction of the KNN between two classes

The prediction of KNN between two classes is slightly better, even for slopers and volumes which can be fairly similar, as they are noticeably "positive". In this case, the KNN accuracy is about 54.54%.

Conclusion

For KNN part, as I know about KNN in advance, it was good to be simple to implement and intuitive to understand. However, the accuracy of the result was not high, and I guess it's because there were not many data samples compared to other KNN programs online since we had to get them by ourselves. Also, it was difficult to find the best K to produce the best accuracy. If I get a change next time, I want to use more data sets to improve accuracy.

For CNN part, in short, things were fairly acceptable. The code runs almost all the way to the end despite the value error bug in fit section. I assume it has something to do with the number of types which is too much. The code is supposed to handle just a two-type classification, but we have 6 different types. And the slight difference between those different rock types is hard even for untrained people to distinguish, not to mention neural network. They are a lot trickier than differentiate written numbers or traffic lights. But in general, I've learned a lot more in the process. This elective course, surprisingly, has the most workload and the most outcome compared to other courses taken this semester.

References

KNN code
https://towardsdatascience.com/knn-using-scikit-learn-c6bed765be75

CNN code
Image classification
https://www.kaggle.com/uysimty/get-start-image-classification

How to (quickly) build a deep learning image dataset
https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/

OpenCV
https://docs.opencv.org/3.4/de/d62/tutorial_bounding_rotated_ellipses.html
https://hub.packtpub.com/opencv-detecting-edges-lines-shapes/
https://stackoverflow.com/questions/55587820/how-to-get-the-only-min-area-rectangle-on-a-multiple-contours-image-with-cv2-min
https://www.pyimagesearch.com/2014/10/20/finding-shapes-images-using-python-opencv/

On climbing:
https://www.dailyclimbing.com/the-ultimate-guide-to-the-different-types-of-climbing-holds/
https://toplogger.nu/blockhaus/my-gym
https://www.sboulder.com/arkose/montreuil?b=EBZcWRF64WQdrqomm
https://www.vertical-life.info/
https://mpora.com/rock-climbing/climbing-holds-climb#58wqpGkpGXG4d0pU.97
https://blog.earthtreksclimbing.com/volumes-the-new-dimensions-of-indoor-climbing