Report for Foundations of Machine Learning-Project00

First A. Author, *Hao Sun*

***Abstract*—This project contains a code to find all red cars in a given picture. Main methods includes threshold value of HSV, k-nearest neighbors (KNN) algorithm, operating about Euclidean distance. Firstly, this report simply describes how to implement my code including using of HSV, KNN algorithm and how my functions work in each file. Experiments part explains how to construct train dataset, test dataset, principle and methods of some experiments and the results such as eliminating error points and removing close points. Also, several photos are attached to illuminate the consequence and efficiency of each experiment. Writing a code may face with many reality problem such as run time and memory overflow. And the main requirement is implemented but not so perfect. Because time and methods are limited, although the result seems to be good, there are some problems. I think it can be better if we can use convolution neutral network (CNN), OpenCV or something like that.**

***Index Terms*—Euclidean Distance, HSV, KNN, Threshold**

I. Introduction

IN this project, I write four *.py files. The main procedure is according to characteristics of HSV (hue, saturation, value) space and Gaussian probability density function to get threshold value and find some "red pixel." Then use k-nearest neighbors (KNN) to eliminate some undesired points such as orange ground and red roofs to make the results more accurate. Finally, remove close data and return locations of each red car. Here I refer to Wikipedia for knowledge of HSV. Below is its website. https://en.wikipedia.org/wiki/HSL_and_HSV

II. Implementation

The whole project contains about three parts.

Firstly, get red cars' RGB (red, green, blue) value according to "data_train.npy" and "ground_truth.npy." Then convert these RGB value to HSV space and calculate mean and standard deviation (std) of H, S and V. I use HSV instead of RGB because it is more explicit about color. H represents angle of color, S represents amount of grey and V represents brightness. Here assume each parameter is Gaussian distribution and remove some points far away from mean according to std. So we roughly get red pixels by restricting some threshold value.

Because the amount of data in "ground_truth.npy" is small and monotonous, we need more data especially some data indicating the characteristics of a pixel which is not a red car. So I create "error_pixel.npy" and "red_car.npy." RGB value in "error_pixel.npy" comes from error points after HSV filter. Import them and merge them into train dataset. Initialize KNN labels according to the sequence of your train data. Use KNN to fit data and predict our red pixels got from former part. Then return points meeting requirements.

Finally, we should remove red pixels indicating same cars. We calculate their Euclidean distance and remove close points. Consequently, we get unique location of each red car.

III. Experiments

At the very beginning, the result is not good and I designed some experiments to make it more accurate.

A. Train test and validation dataset

1) Train section

In fact, in this project I use two train datasets. At first, I just use offered "ground_truth.npy." I convert RGB to HSV and choose three 1-D Gaussian distribution and create thresholds to filter points far away from mean, But the outcome is unacceptable. I plot my predicted points in the picture and find many orange ground and red roofs are included. Then I realized that the train dataset is too small. So I decide to train again at the base of former result. So I select points of orange ground and red roofs and add them into "error_pixel.npy." I just add RGB value for convenience. Then for accuracy of KNN, I add some points of red cars into " red_car.npy" and merge them into one train dataset. Also, I initialize labels for train dataset. Then train them use KNN.

2) Test section

In test module, firstly convert all test data to HSV and use threshold parameters to roughly get red pixels. I changed threshold parameters several times and compare their accuracy by plotting them in the picture and get a relative accurate parameters. Then use KNN's predict function to remove error points and return satisfied points. Finally I should remove close points. I think a red car must own several continuous red pixel and their distance is limited by car's length. I observed the plot and find a red car's diagonal length is about 10 pixel, so I use 10 as distance threshold.

This report is written on October 7th, 2018

F. A. Author is a graduate student in University of Florida, Gainesville, FL 32608 USA (e-mail: haosun@ ufl.edu).

Here use "calculateddistance" function to calculate distance of each rows (each row represents a location of red pixel). Returned argument [i, j] represents i-row and j-row in my red pixel dataset. If my i, j occurs, it shows they are close to some points, we use this way to get rid of some noise red pixels. Then find repeated rows exists in i and j simultaneously and remove repeated rows and return to your picture to get locations of our predicted points.

3) Validation section

When training data, I use cross-validation by changing the value of M and get a high accuracy train dataset. When testing and improving my code, the main method I used to validate my result is plotting the predicted point in the picture. Observe the accuracy and add some error points into error_pixel and train again.

B. Experiments Design

1) Choose threshold value of H, S, V

In fact at the very beginning, this is the only method in my code. In this experiment I want to get an acceptable range of H, S and V. Here I use threshold because of characteristics of HSV space. And we can calculate cumulative distribution function between my threshold, the outcomes are about 0.5. If a point is far away from my mean, its probability is small. I do not use probability density function because it is hard to get a suitable threshold. At first, I use 3-D Gaussian module, the value is too small (about 10^{-9}) to distinguish a red pixel from my data. Meanwhile, I do not add a class which contains not red cars' RGB. So this is the only way I can do. Naturally, the result is not satisfying. As Fig. 1 shows, I got many red car but also some error points such as orange ground and red roofs.

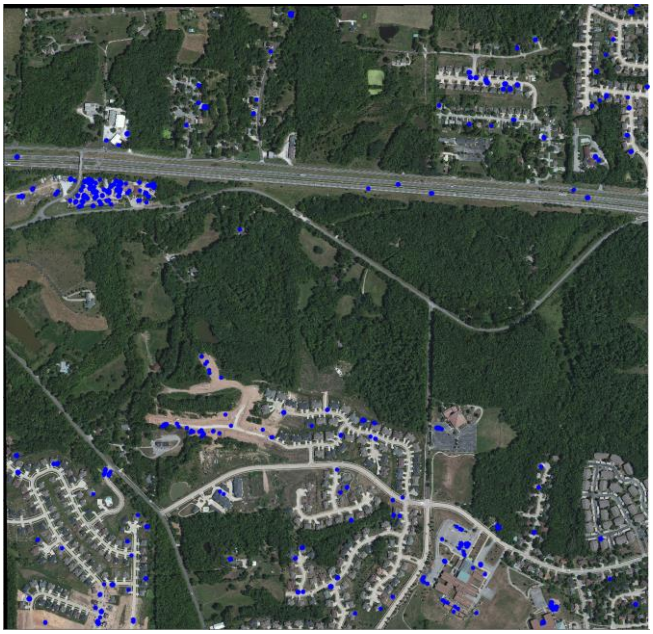

Fig. 1. Result of HSV threshold value. Blue points show locations of predicted red cars.

2) KNN experiment

The result of experiment (1) is obviously far from my expectation, so I designed another experiment to improve my result. The goal of KNN is to remove some error points such as red roofs. I plot points got from experiment (1) and find error points. Then add these points' RGB value into my file "error_pixel.npy." Also, I add some red cars' RGB into my file "red_car.npy." Next, utilize KNN function to fit and predict my points. I plot the result point again and compare them with the result of experiment (1). The result illustrates in Fig. 2. It successfully removes some red roofs and noise points at street and ground. Meanwhile, I changed my value of K and get a relatively good result. But some roofs looks like a red car can not be filtered, if I stick to removing them, I will lose some red cars' locations.

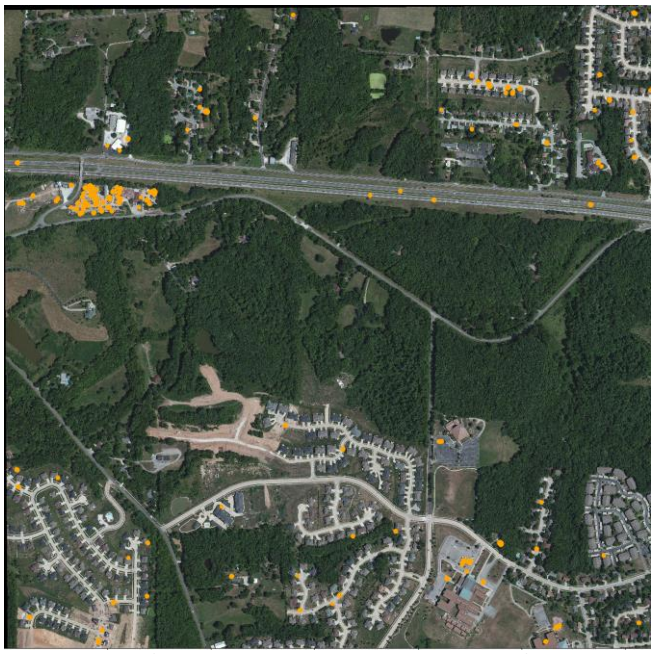

Fig. 2. Result of KNN. Orange points indicate locations of predicted red cars.

3) Remove close points

After finding apposite red pixels, I need to remove repeated points indicating same cars. I think calculating Euclidean distance is a good choice. I write a function to calculate distance of each rows in a matrix. Input matrix (m*n) and matrix (m*n) itself and return a matrix (m*m). The i-row j-column of new matrix indicates distance between i-row and j-row in original matrix. When I have a huge dataset, it can accelerate my code drastically. According to observation, I find a red car has some stable features. Firstly, it contains several red pixels. Secondly, the distance of red pixels can not exceed the length of a car. And I find the length of a car is about 10 pixels, so I use 10 as my distance threshold. The first thing I do is to find points whose distance with others is less than my distance threshold. Then I should remove points close to other points and get my locations of red cars. Again, I plot these points. Because full photo is too big, we can hardly discovery differences between result of HSV threshold and KNN, I plot a partial photo shows in Fig. 3. Although I removed close points, I find if two red cars are

very close, for example, at parking lot, I may lose some locations. It is caused by this algorithm and if I set the distance threshold too small, I will get two points even three or four points in a single car. Unfortunately I can not come up with a better algorithm to find a balance.

Fig. 3. Result of removing close points. Orange points indicate locations of predicted red cars after KNN. Blue points indicates locations of red cars after removing close points. To illustrate result more clearly, here is magnified picture.

IV. Conclusion

After the running project, I get relative accurate locations of red cars. Fig. 4 shows it has a high accuracy in picture data train. Here I do not use K-means because its classification is not stable, random center can be everywhere which will decrease the accuracy of result. Meanwhile, I have gotten labels for red cars and not red cars, I think it is a problem of supervised learning instead of an unsupervised learning. For regression, I can not imagine a line or surface can express the boundary perfectly. Probabilistic generative method is the first method I tried to use because an appropriate 3-D Gaussian distribution can almost express everything. But A Priori is a disturbing term when I calculate A Posteriori. And the result is too small. It is hard to distinguish.

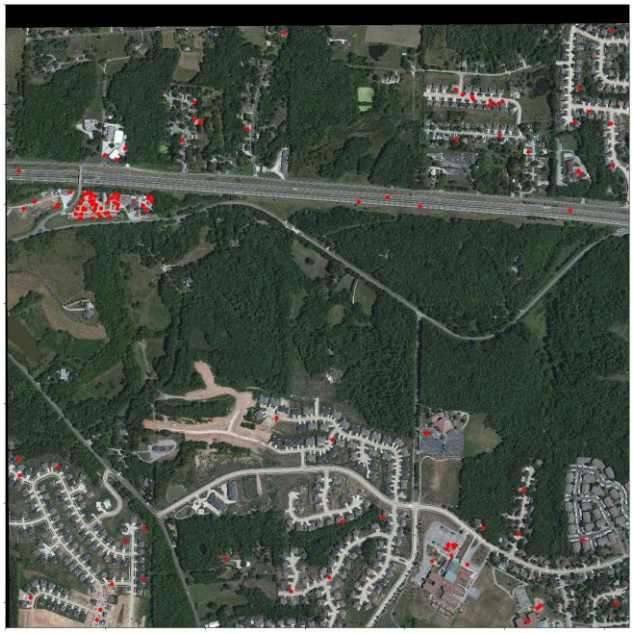

Fig. 4. Result of full data train picture. Red points indicate locations of predicted red cars.

I think this method is good at predicting new picture especially for those possessing less red roofs. I plot the predicted result for full test picture in Fig. 5 and a representative small part in Fig. 6. But undeniable it has some defects. Firstly, my train data is not abundant enough so I can not express all RGB values and distinguish them exactly. Secondly, some roofs' color is almost same with red cars. It is hard to distinguish them. When I remove some roofs, I lose some red cars' locations. This dilemma happens when I use KNN to remove error pixels. And what I can do is find a relative accurate balance. For the final part to remove close points, inevitably I will lose some red cars' locations. Because time is limited, I am sorry I can not make it better. Furthermore, I find running speed is an important factor when predicting such picture. I considered use KNN to replace my first part, but KNN is time-consuming, and if I use KNN at the first time, I need to add more train data. So I use HSV threshold value as a filter and KNN targeted eliminate error pixels.

If anything missed or uncertain, you can refer to README.txt or comments in code. Please contact me if these are any problems.

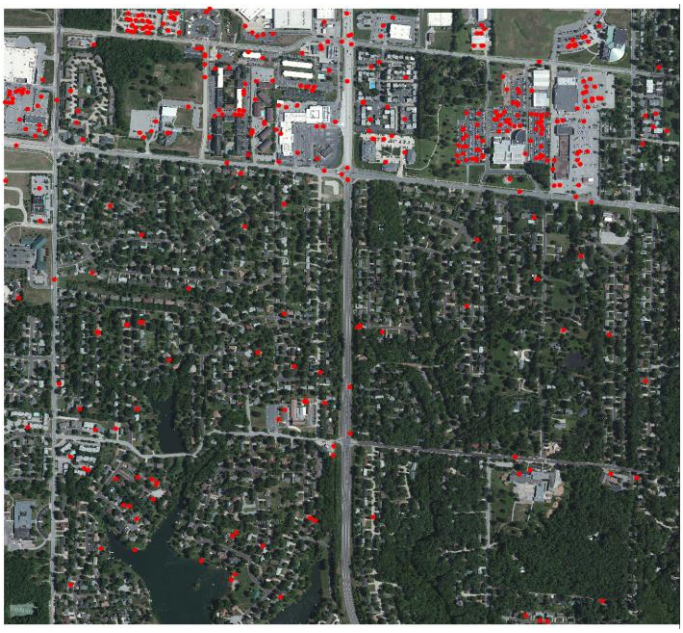

Fig. 5. Result of full data test picture. Red points indicate locations of predicted red cars.

Fig. 6. Result of a small part of data test picture. Blue points indicate locations of predicted red cars.

REFERENCES

[1] S. Patil, U. Bhangale and N. More, "Comparative study of color iris recognition: DCT vs. vector quantization approaches in rgb and hsv color spaces," *IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sept. 2017. DOI: 10.1109/ICACCI.2017.8126070.