

# Summary Report of Find Phone Task

Email: [linhai\\_li@hotmail.com](mailto:linhai_li@hotmail.com)

## Preface:

The purpose of this mini-project is to demonstrate my ability to solve a practical computer vision and machine learning problem. I did not try to tweak the model to its best performance to reach production criteria. Especially this relatively simple model seems to perform better than expected as compared to the accuracy indicated in your instruction document. Considering the small size of data set (main reason) and busy schedule with my current full-time job, it may not worth more effort to improve the model unless more data are available. Instead, I will point out a few directions of future improvements at the end of this report.

## Key results:

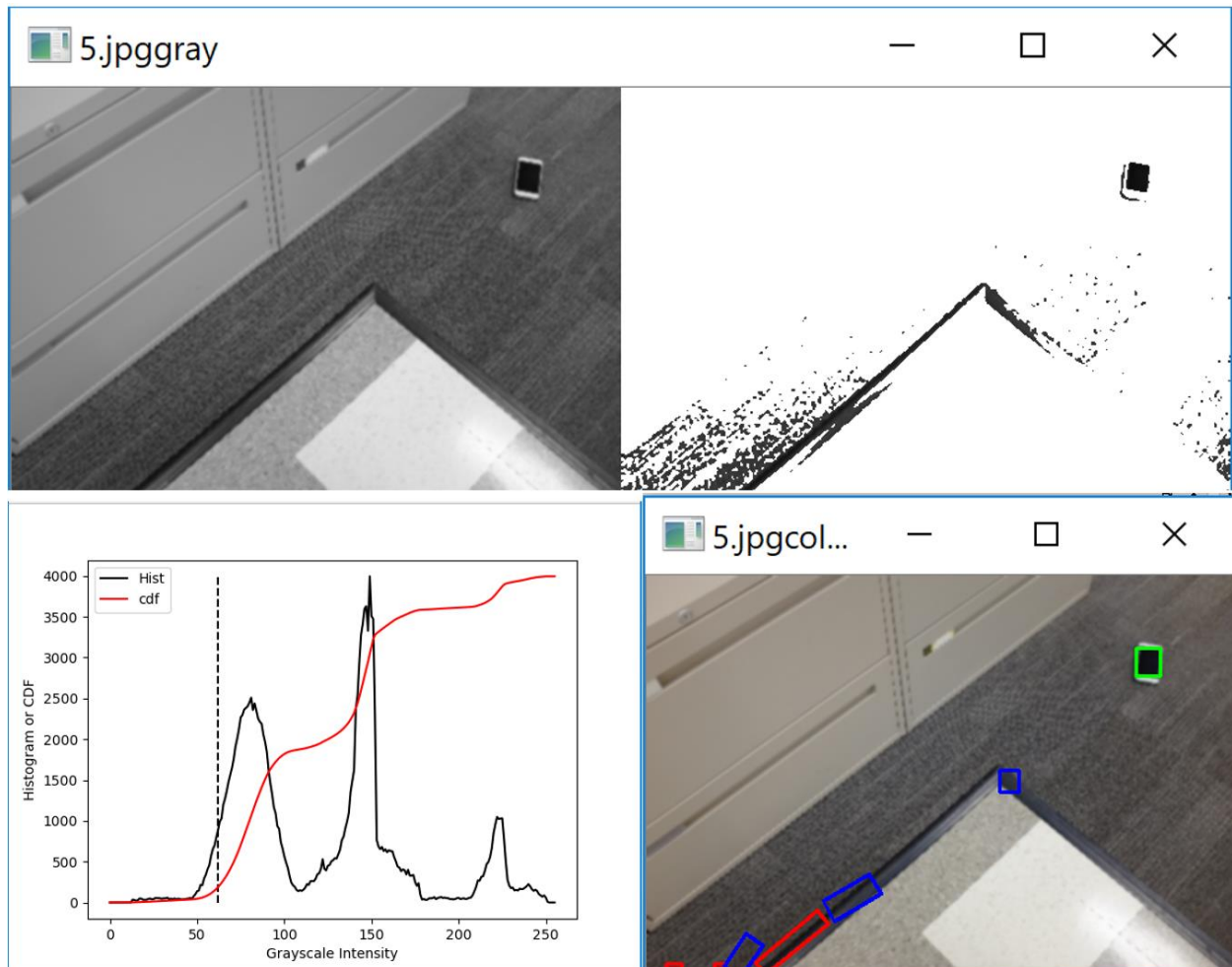
- Using this very simple model, I am able to detect the centers of 111 phones out of total 129 phones. It means 86% accuracy, when using all images in the training. The submitted model is the version with all provided images fed to the training.
- From 10 tests in which ten images were randomly selected and excluded in all the training process, I am able to successfully detect the center of phones with accuracy 100% (3 tests), 90% (3 tests), 80% (3 tests), and 70% (1 test). In average, I am able to detect the center of unknown phones with accuracy of 88%. Depending on the images in the test, this model can likely succeed more than 70% of times, but it may also fail completely on extreme cases (see below for analysis).
- It seems that this model has difficulty in detecting center of phones at certain scenarios based on a very shallow and brief analysis (see below).

***NOTE:** I developed this model in Windows but attempted to simulate Linux environment in Git Bash shell. The code was written in Python 3.5, which may cause problems in Python 2.7. However, it shouldn't be a big problem as long as the used packages are up to date. I already tried to deal with print function difference, but please read the comment in "find\_phone.py" and uncomment the last line if problem occurs. Please ask me if you have any problems caused by difference in OS and Python version. I guarantee the code works with minor changes if needed.*

## Steps/thinking process to solve the problem:

- (i) I started this challenge by reading the instruction. It gave me an understanding about the background, objective, and evaluation criteria.
- (ii) The very first idea I had was to use train a convolutional neural network (CNN) to detect the center of the phone or the bounding box of the phone. CNN may be better at handling the bounding box, which means I actually need to label each phone with the coordinates of their bounding box. If I do it manually, it will take too much time and it is not ideal. Then I came up with the idea in (iii) and attempted to find the bounding box of the phone automatically.
- (iii) I then studied the original images shown in "find\_phone" folder. It made me think that all phones have a black screen, which can be used to find the bounding box of the phone screen. A very simple strategy was applied: the bottom 5% pixels with lowest pixel intensity on a

converted grayscale image were kept and pixels with higher intensity were excluded. A morphology approach was then used to remove small areas (e.g. very small black spots) and connected separated parts to a bigger block (e.g. two separated phone screen block caused by glares become a bigger screen), after which I used contour detection to form a mask for all suspected regions. Not all regions are phone screen, but phone screen should have some dimensional features (e.g. shape, size, and aspect ratio). Based on the given coordinates given in "labels.txt", I was able to extract the bounding box for the true screen and got the range of the length, width, area, and aspect ratio, which were later used to exclude the regions that are not likely to be a phone screen. See an example shown in Figure 1.



**Figure 1.** **Top-left:** Converted grayscale image for 5.jpg; **Bottom-left:** Histogram of grayscale intensities (black line) and the 5% cutoff value (dashed line); **Top-right:** The remaining pixels after the cutoff operation; **Bottom-right:** The final detected suspected regions. Red rectangles suggested the regions were excluded after applying the phone dimensional check; blue rectangles are regions still considered as phone screen; and green rectangle is the actual phone screen.

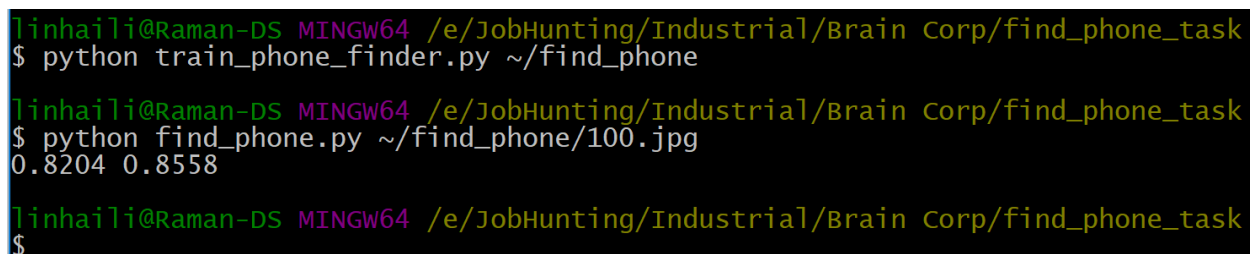
(iv) After initial testing, I found that in about 40 out of total 129 images the phone screen is the only detected region, in 84 out of 129 images there are more than one regions detected, and finally in 4 out of 129 images no region is detected at all. Then I started to think it might be

better to just train a classifier to recognize if the detected sub-regions are phone screen or not. It would be more reasonable than training a CNN to detect a very small area in the image, especially there are not sufficient data for CNN. As a result, I abandoned the CNN idea to test my other idea.

- (v) All the detected sub-images in (iv) were then extracted and saved in folder "train\_img" for training a classifier to recognize phone. In total, I extracted 314 sub-images from 129 original images and all sub-images were resized to 50 by 50 pixels. There is many ways to classify these sub-images into phone vs. non-phone. I decided to try an easy one, as I cannot build a production-ready model based on such small data set anyway. I computed the HOG feature of each sub-image and fed HOG features to train a SVM classifier. It works reasonably at the first place. I tried different kernels and hyperparameters in SVM and determined the "best" ones. It was found that the SVM is very sensitive to different kernels but not sensitive hyperparameters. In general, both linear and poly kernels work well but not RBF and sigmoid kernels. (Note: I suppose I can further optimize SVM classifier but again I did not do extensive tweaking the model because it works really well already and it is not intention to come up with the "best" model for this mini-project.)
- (vi) I have all the components ready. When a new image is fed to the model, it first detects the suspected regions in the image, followed by calculation of HOG feature for each suspected region, then being classified by the trained SVM, and finally the center of suspected region with highest probability as a phone is returned. In case there is only one suspected region detected, the center of that region is returned directly without being passed to SVM classifier.

### Examples of accurate detection:

Figure 2 shows the screenshot that I ran the model in Git Bash shell. I first trained the model and then test the model using "100.jpg". The model then outputs the coordinate of the phone center as "0.8204 0.8558".



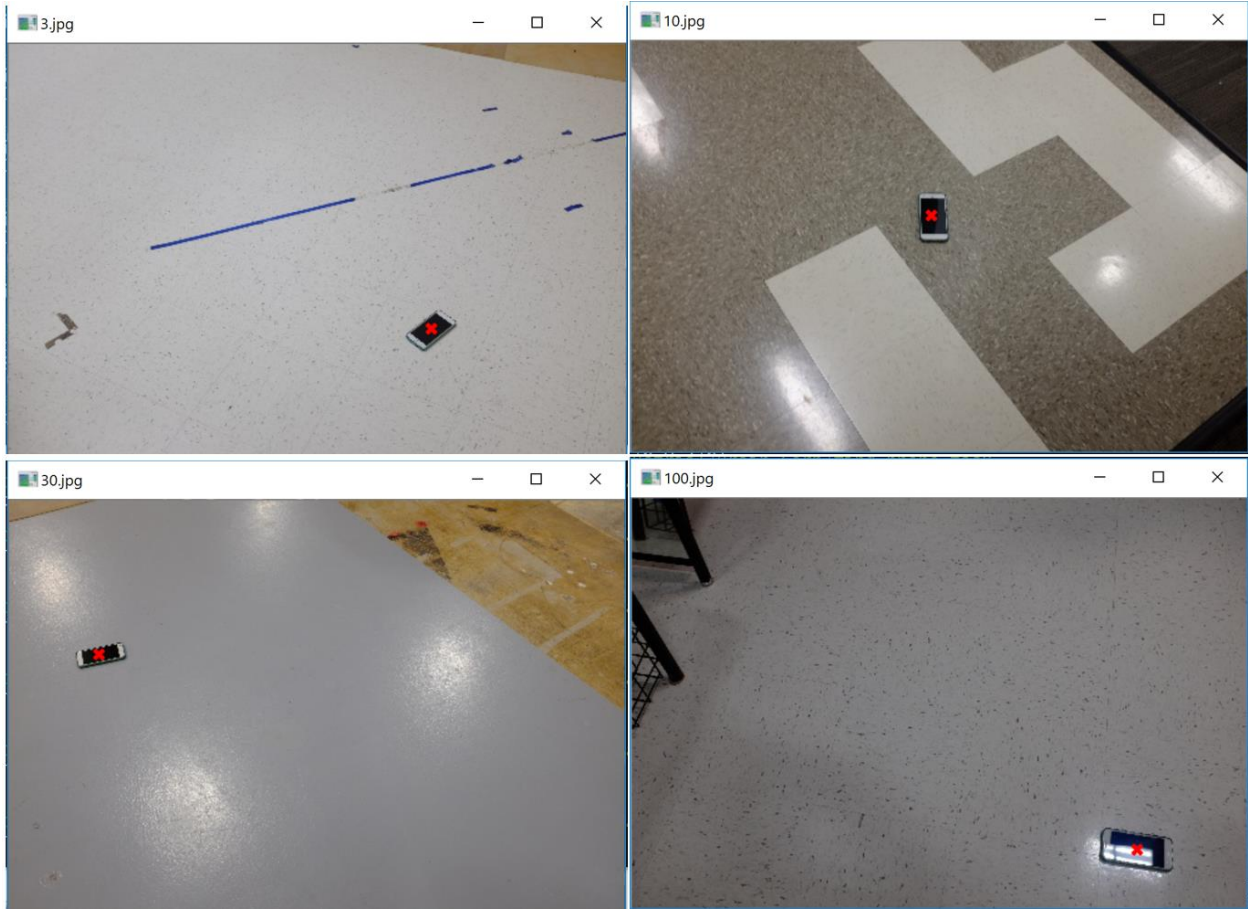
```
linhaili@Raman-DS MINGW64 /e/JobHunting/Industrial/Brain Corp/find_phone_task
$ python train_phone_finder.py ~/find_phone

linhaili@Raman-DS MINGW64 /e/JobHunting/Industrial/Brain Corp/find_phone_task
$ python find_phone.py ~/find_phone/100.jpg
0.8204 0.8558

linhaili@Raman-DS MINGW64 /e/JobHunting/Industrial/Brain Corp/find_phone_task
$
```

**Figure 2.** Screenshot of running the model in Git Bash shell.

Figure 3 shows the visual inspection of four examples for which the phone centers are accurately detected. I chose four images with different mixture of backgrounds. The images used in the test can also be seen from the top-left corner of each sub-figure.



**Figure 3.** Four examples with the phone center accurately detected. The red x's shows the detected center.

### Simple statistical analysis based 10 tests:

In order to better evaluate the model, I randomly picked 10 images that were excluded in the training phase and then these 10 images were used to test the model. I did such test for ten times. In each test, the training images and the test images are different. Therefore, I consider each test is independent. Table 1 shows the simple statistical analysis of these ten tests. Based on these ten tests, I suggest that there is no over-fitting or under-fitting problem, which is the reason that I decided not to tweak the model further.

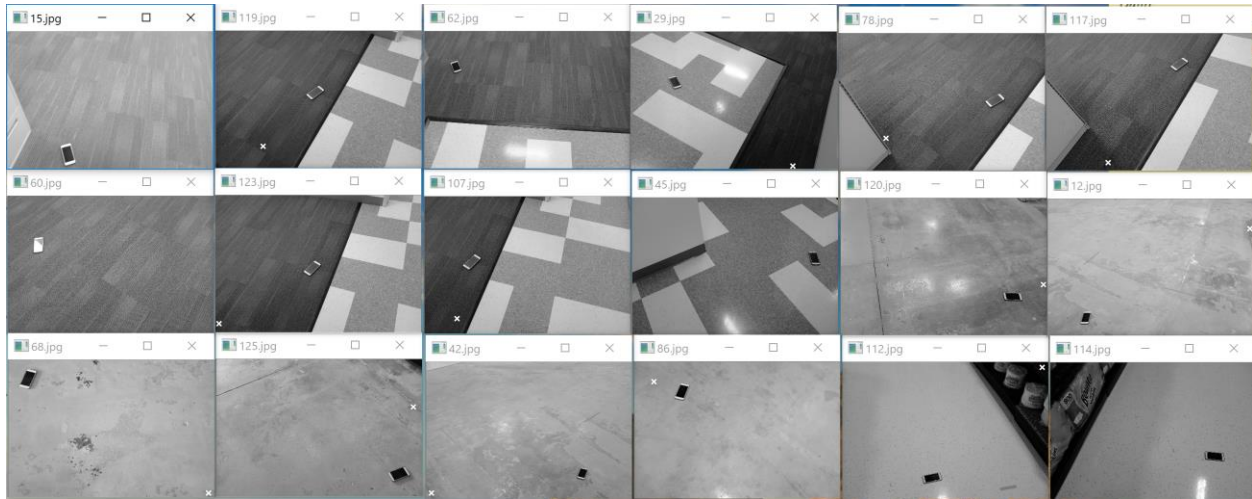
**Table 1.** The comparison of training and test accuracy based on ten independent tests.

No.	Train Accuracy	Test Accuracy	No.	Train Accuracy	Test Accuracy
1	85.7%	80%	6	85.7%	80%
2	86.6%	90%	7	86.6%	100%
3	86.6%	100%	8	85.7%	70%
4	85.7%	90%	9	85.7%	80%
5	86.6%	100%	10	86.6%	90%

### Analysis of the failed cases:

Regardless of the reasonable performance, there are 18 cases, as shown in Figure 4, not being able to be detected by the model after some improvements to the model. I suggest that these cases can be conquered with better strategy and more data. Again in this project, I am not trying to come up with a production-ready model. Therefore, I did not try to conquer these cases and dig deeper. Instead, I did a simple analysis based on initial thoughts.

First of all, 9 out of 18 cases are the scenarios where phone lay on black carpet. It suggests that the phone screen were not detected from its background, which implies that the 5% cutoff based on histogram may not work best for this scenario. The same reason is applicable to 3 cases where phone are lay on relatively dark (dirty) ground. Similar reasons can also go to 2 additional cases with big patches of very black background. For the remaining 4 cases, the reason is not very clear to me.



**Figure 4.** All eighteen failed cases because either the phone is not found or its center is not accurately detected. The white "x" indicates the predicted center of phone.

### Future improvements:

The proposed future improvements are based on current model framework and it may not be applicable to other types such as CNN.

- (i) A better strategy to segment the image into bright and dark pixels to more effectively detect a phone screen. I currently set a constant cutoff percentage, i.e. 5%, to binarize the images. If there are sufficient data, I expect to build an algorithm to predict a dynamic cutoff value for each image. This may solve problem in most of the above 18 failed cases.
- (ii) Get a more robust statistics on the phone dimensions extracted from the images. This will help us to exclude regions that are not phone screen and most likely keep the phone regions.
- (iii) Attempt to use the white patches around the phone screen explicitly although this feature may be already used in the SVM classifier.
- (iv) Try other classifiers other than SVM, like random forests, neural networks, etc. Ensemble learning technique from these individual models may improve the performance a little.