Find Phone Task

+1 551 260 0840 || Viramya Shah || shah.viram@husky.neu.edu

A short and informal note

This is a short report based on my intuitions and assumptions. The report mainly focuses on the methodology used and why was it used particularly. It is divided into as follows:

1. Dataset
2. Packages used
3. Intuition, Approach and Methods
4. Files and Results
5. Further work

Dataset

The dataset that was provided contained 129 images. The images were pictures of a phone lying on a floor. Along with that labels.txt was provided that provided the coordinates of the phone.

Although, it was not clearly mentioned whether the coordinates given were of a corner of the phone or did they represent the center. On eyeballing a few cases (using openCV for drawing point as per the given coordinate), it was clear that they belonged to center.

Packages Used

Following packages have been extensively used.

1. Numpy, Pandas for data handling and manipulation
2. Keras for modelling
3. openCV for image processing
4. gluonCV for using pre-trained model

Intuition, Approach, and Method

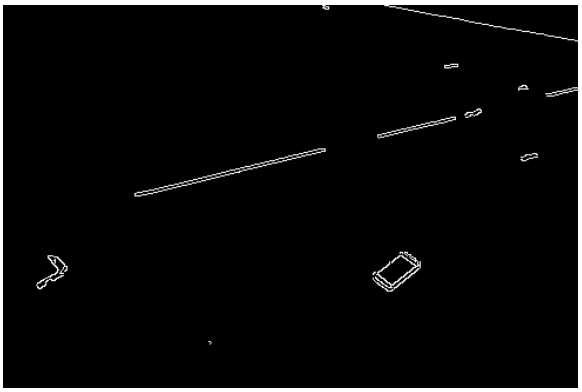
I had 2 different views on how to see the problem statement. One was using a state-of-art object detection model like SSD, Faster R-CNN or YOLO. Other was to make a own custom model.

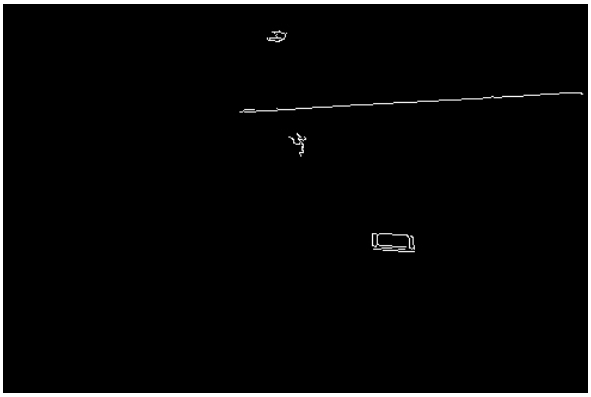
I made a loose analogy with class-agnostic and class-aware method; stating the pre-trained model would detect all the objects present in the image and after detecting, based on the class scores would, with certain confidence, predict the object. On the other hand, class-aware would mean the model knows what it has to look for. The latter made more sense as the problem statement clearly mentioned that 'every image has a phone and coordinates were to be predicted'.

Also, while using a pre-trained model, I needn't make and/or train my own model. I can directly use it for prediction. I used Single Shot Detector (requiring 512 px on the short side) pretrained on Common Objects in Context (COCO) dataset. That implementation can be found in ***find_phone_pretrained.py***.

For my custom model, I planned on giving the model least information about the surrounding and most information about the phone. So before I started with the custom model, I eyeballed the images. The images mainly involved a tiled/non-smooth floor design. Based on this observation, I applied a blurring

filtering using openCV. Particularly, I chose ***Bilateral Filtering***, as unlike others, it blurs while preserving the edges. Later, I applied ***Canny Edge Detection***.





Thus, I obtained images with minimum background information and maximum information about the phone.

Another problem was the size of the dataset. 129 images were too less for model to have considerable training. Thus, I simply augmented the dataset by flipping the images on both short and long side. While doing so, I also had to update the label.txt with newer coordinated for the new images.

Thus, I now had 129*4=516 images. The images, as we can see, are sparse.

The model architecture contains of 6 convolution blocks and 6 FCC layers. The kernel size (3x3) and pooling size (2x2) are same for all the convolutions. Every layer has a ***'Rectified Unit' (ReLU)*** as activation with last layer having a ***sigmoid*** activation. My intuition for choosing sigmoid was to squash the outputs in (0, 1) range. With ***Adam*** optimizer and ***Squared Log Error*** loss function, I trained model for 20 epochs.

```
.
.
.
Epoch 18/20
464/464 [==============================] - 3s 7ms/step - loss: 9.8504e-
05 - acc: 0.9634 - val_loss: 0.0033 - val_acc: 1.0000
Epoch 19/20
464/464 [==============================] - 3s 7ms/step - loss: 1.4130e-
04 - acc: 0.9634 - val_loss: 0.0036 - val_acc: 1.0000
Epoch 20/20
464/464 [==============================] - 3s 7ms/step - loss: 1.3158e-
04 - acc: 0.9720 - val_loss: 0.0034 - val_acc: 0.9808
```

Files and Results

The folder contains following .py files:

1. train_phone_find.py

   Custom model architecture that accepts the path to folder containing images and labels file via command-line

   - The file creates/saves following files:
     - my_model_log_loss_colab.h5: Saving the model
     - master_final_custom.xlsx excel sheet with results as explained below

2. find_phone.py

   Load the model and runs the images for co-ordinate prediction. It accepts the image path via command line as well

3. find_phone_pretrained.py

   This files runs SSD model trained on COCO dataset. The model returns a bounding box, but I have coded it in a fashion that would return co-ordinates of the center.

The last column in excel sheets generated have values of 1, 2 and 3. This is my own metric where 1 indicates that the prediction was within 5% error of the ground truth, 2 indicates < 15% error whereas 3 indicated >15% error. The performance of the model can be seen from this pivot table.

Row Labels	Count of x_acc	%	Count of y_acc	%
1	500	96.90%	504	97.67%
2	13	2.52%	7	1.36%
3	3	0.58%	5	0.97%
Grand Total	516	100.00%	516	100.00%

1 Distribution of errors on custom model

Row Labels	Count of x_acc	%	Count of y_acc	%
1	104	80.62%	107	82.95%
2	1	0.78%	5	3.88%
3	24	18.60%	17	13.18%
Grand Total	129	100.00%	129	100.00%

2 Distribution of errors on pretrained model

Further Work

I could think of following ways to improve upon:

1. Explicitly ask the customer to use a standard background.
    a. But that would incur overhead on the customer and they might not follow.
2. Using image processing technique such as Contour Detection, Masking, HSV Coloring
    a. The sole intention behind steps would be to reduce information in the surrounding and get more focus on the object i.e. phone