Notes

My Approach using Template Matching

(The solution was coded on Google Colaboratory platform and verified on the dataset. The only package needed to be installed is opencv to import cv2 object. Please contact me at asrikanthan3@gatech.edu or 4703120779 if there are any issues with running the program. I have tested it on my system.)

The problem is solved using template matching method. "0.jpg" is used to create the template image. The labels are extracted from labels.txt and stored in a dictionary. The location of the phone is identified using the label and the template image is cropped out of the main image and saved as "template.jpg".

Template matching method has been implemented using OpenCV where a window size equivalent to the size of the template image is used to slide across image to obtain the location of maximum overlap. This is basically similar to 2D convolution using a filter where the filter in this case is the template itself. We choose the location with maximum value of the residual obtained after template matching and store this as the top left corner of the bounding box, if drawn around the phone. The mid-point of the bounding box is chosen to be the location of the phone. This proved to work well and accurately predicts the locations of the phone in approximately 124 out of 134 images in the dataset giving an accuracy of prediction around 92%. This method also requires minimal setup – installing opencv and python. It is not computationally expensive and has a small run time. Since this task involves only detecting a single object from the image and it was mentioned that the phone is unique and the problem of interest is only based on this particular phone, this method provides a simple and elegant solution.

Further Improvements

Currently, there are state-of-the-art methods that use Faster-RCNN, RetinaNet and FPN networks to perform Object Detection. However, they are computationally expensive and require a large dataset. These methods should be used only when necessary since a lot of time goes into preparing the data and installing dependencies. I don't think this approach is very elegant, but I can't rule out the possibility that they might work better.

The template matching method used in my approach can be improved by considering other evaluation metrics on the template to make it more accurate in predicting the location of the phone. In certain cases, the background artifacts result in inaccurate identification of the phone's location. This can be handled by adding more features to identify the phone.

If there were other types of phones or multiple objects in an image, then a classifier can be designed based on each template. For a classifier, an SVM or VGG16/VGG19 which is the current state-of-the-art method for object classification can be used depending on the problem specification.

Suggestions for Data Collection

I think it is very important to have data that is labelled continuously without any breaks. This dataset had a few elements missing and had to be hard coded in the for loop to be accounted for. Also, to make the problem more interesting and challenging, acquiring images with multiple phones and images with different types of phones would be a good idea.

Code

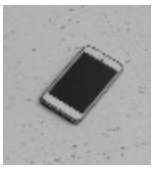
train_phone_finder.py

```
# Visual Object Detection using Template Matching
# Code to train on training dataset
## There is no machine learning involved in this method and it works
really well. I've attached more details and suggestions
## in a pdf file named "Notes"
### !apt-get -qq install -y libsm6 libxext6 && pip install -q -U opencv-
python (To install OpenCV on Google Colab)

import cv2
import numpy as np
import argparse
from pathlib import Path

# Accept path from command line
parser = argparse.ArgumentParser()
parser.add_argument("file_path", type=Path)

p = parser.parse_args()

path = p.file_path

# Store the labels in a dictionary
newDict = {}
with open(str(path)+"/labels.txt", 'r') as f:
        lines = f.readlines()
        k = 0
        for i in lines:
          words = i.split()
          temp = words[0]
          img_name = int(temp[0:len(temp)-4])
          coordx = float(words[1])
          coordy = float(words[2])
          newDict[img_name] = (coordx, coordy)

# Create Template Image
img = cv2.imread(str(path)+"/"+"0.jpg", 0)
imgx, imgy = img.shape[::-1]
mx, my = newDict[0]
mx = int(mx*imgx)
my = int(my*imgy)
temp = img[my-40:my+40, mx-37:mx+37]
cv2.imwrite(str(path)+"/template.jpg", temp)

# Create a variable to store labels after template matching
label = np.zeros((135, 2))

# Iterate over each image in the folder and generate the label
```

```
for i in range(0, 135):

    # These images were missing in the given dataset
    if i in [2, 19, 21, 28, 56, 65]:
      label[i][0] = 0
      label[i][1] = 0

    else:
      img = cv2.imread(str(path)+"/"+str(i)+".jpg", 0)
      imgx, imgy = img.shape[::-1]

      template = cv2.imread(str(path)+'/template.jpg', 0)
      w, h = template.shape[::-1]

      methods = 'cv2.TM_CCOEFF'

      method = eval(methods)

    # Apply template Matching
      res = cv2.matchTemplate(img, template, method)

      min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
      top_left = max_loc

      bottom_right = (top_left[0] + w, top_left[1] + h)

      mid_point = (float(top_left[0] + w/2), float(top_left[1] + h/2))

      label[i][0] = mid_point[0]/(imgx)
      label[i][1] = mid_point[1]/(imgy)

      cv2.rectangle(img, top_left, bottom_right, 255, 2)
      #cv2.imwrite(str(path)+"/output_images/"+str(i)+"_out.jpg", img)
      cv2.imwrite(str(path)+"/"+str(i)+"_out.jpg", img)

# Compute the total valid matches
count = 0
for i in range(0, 135):
  if i in [2, 19, 21, 28, 56, 65]:
    newDict[i] = (0, 0)
  accx, accy = label[i] - newDict[i]
  if(accx < 0.05 and accy < 0.05):
    count += 1

# Accuracy of prediction in training
print(100*count/134)
```

find_phone.py

```
# Visual Object Detection using Template Matching
# Code to test on test dataset
# Please ensure that the template image created by the previous script
exists in the current working directory
import cv2
import numpy as np
import os
import argparse
from pathlib import Path

# Get path to the current working directory which should contain
template.jpg
cwd = os.getcwd()

# Get path to image to be tested
parser = argparse.ArgumentParser()
parser.add_argument("file_path", type=Path)
p = parser.parse_args()
#print(p.file_path, type(p.file_path), p.file_path.exists())
path = p.file_path

# Load the image
img = cv2.imread(str(path), 0)

# To hold the coordinates of the phone
label = np.zeros(2)

imgx, imgy = img.shape[::-1]

template = cv2.imread(str(cwd)+'/template.jpg', 0)
w, h = template.shape[::-1]

methods = 'cv2.TM_CCOEFF'
method = eval(methods)

# Apply template Matching
res = cv2.matchTemplate(img, template, method)
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

top_left = max_loc
bottom_right = (top_left[0] + w, top_left[1] + h)

mid_point = (float(top_left[0] + w/2), float(top_left[1] + h/2))

label[0] = mid_point[0]/(imgx)
label[1] = mid_point[1]/(imgy)

#print(label[0] label[1])
print(" ".join([str(label[0]), str(label[1])]))
```



An example of the Template Image obtained from the main image that is used for matching

File Layout

File named Srikanthan_Anusha.ipynb contains the complete code in a jupyter notebook. The primary folder has been named Srikanthan_Anusha which contains 4 files. The two python scripts namely “train_phone_finder.py”, “find_phone.py” and a pdf file called “Notes” containing a writeup and both the scripts in a jupyter notebook named “Srikanthan_Anusha.ipynb”.