OPTICAL CHARACTER RECOGNITION USING OPENCV AND TESSERACT

VAISHNAV S ANAND
18BCE0109
COMPUTER SCIENCE ENGINEERING
VIT UNIVERSITY VELLORE
asvaishnav9999@gmail.com

N THARUNKUMAR REDDY
18BCE0368
COMPUTER SCIENCE ENGINEERING
VIT UNIVERSITY VELLORE
tharrunreddy@gmail.com

M RAGHUNANDAN REDDY
18BCE0364
COMPUTER SCIENCE ENGINEERING
VIT UNIVERSITY VELLORE
m.raghunandan3@gmail.com

***Abstract*— Optical character recognition (OCR) is the electronic recognizing and advanced digital encoding of composed or printed text by methods for an optical scanner and specific software. Utilizing OCR programming allows a PC to peruse static pictures of text and convert them into editable, searchable and accessible data or information.**

Character recognition is a prominent area of research in the field of pattern recognition. There is an enormous interest for OCR on transcribed, sign boards and so forth., documents in Image processing. Despite the fact that, enough studies have performed in foreign scripts like English, Arabic, Chinese and Japanese, just a not many works can be followed for handwritten character recognition. OCR framework improvement for content and pictures has numerous application regions like safeguarding original copies and antiquated literary works written in a several scripts and making advanced digital libraries for the documents. Feature extraction and classification are basic steps of character recognition process influencing the overall precision of the popularity system. This paper presents a brisk overview of Digital Image Processing methods like Feature Extraction, Image Restoration and Image Enhancement.

Keywords—OpenCV, tesseract, OCR, blobbing, insert

I. INTRODUCTION

Utilizing OCR software permits a PC to peruse static pictures of text and convert them into editable, accessible data. The fundamental procedure of OCR includes analyzing the text of a document and making an interpretation of the characters into code which will be utilized for processing. OCR is normally likewise referenced as text recognition. OpenCV was begun at Intel in 1999 by Gary Bradsky, and the primary release turned out in 2000. Vadim Pisarevsky joined Gary Bradsky to deal with Intel's Russian software OpenCV team. In 2005, OpenCV was utilized on Stanley, the vehicle that won the 2005 DARPA Grand Challenge.

The essential procedure of OCR includes examining the content of a document and making an interpretation of the characters into code which will be utilized for processing. OCR is typically likewise referenced as text recognition. Tesseract is an optical character acknowledgment (OCR) framework. It is utilized to convert picture archives into editable/accessible PDF or Word documents. Tesseract is viewed as one of the most precise open source OCR engines right now accessible and its development has been sponsored by Google since 2006.

II. OBJECTIVE

A. *OPTICAL CHARACTER DETECTION*

Optical Character Detection is done with the help of python module OpenCV. What OpenCV actually does is that it matches the input images with the already trained neural network, the same have been imported to the program. It is able to detect multiple texts at the same time and by mapping to the neueral network. The Regoin of Interests are ex tracted in this step. Now the next step is to recognise the text present in these regions of interest.

B. *OPTICAL CHARACTER RECOGNITION*

The Tesseract is an open source text recognition (OCR) Engine, accessible under the Apache 2.0 permit. It tends to be utilized legitimately, or (for developers) utilizing an API to separate printed text from pictures. It bolsters a wide assortment of dialects. Tesseract doesn't have an inherent GUI, yet there are a few accessible from the 3rdParty page. The region of interests extracted from the previous step is added to it to find the text. Tesseract tends to be utilized with the current design investigation to perceive text inside a huge record.

III. LITERATURE SURVEY

• In 1951, M. Sheppard invented a reading and robot GISMO which will be considered because the earliest work on modern OCR. GISMO can read musical notations also as words on a printed page one by one. However, it can only recognize 23 characters. The machine also has the potential to could copy a typewritten page.

• J. Rainbow, in 1954, devised a machine which will read uppercase typewritten English characters, one per minute.

• The early OCR systems were criticized due to errors and slow recognition speed. Hence, not much research efforts were placed on the subject during 60’s and 70’s.

• The only developments were done on government agencies and large corporations like banks, newspapers and airlines etc. Because of the complexities related to recognition, it had been felt that three should be standardized OCR fonts for alleviating the task of recognition for OCR.

• Hence, OCRA and OCRB were developed by ANSI and EMCA in 1970 that provided comparatively acceptable recognition rates.

• During the past thirty years, substantial research has been done on OCR. This has cause the emergence of document image analysis (DIA), multi-lingual, handwritten and omni-font OCRs.

• Despite these extensive research efforts, the machine’s ability to reliably read text is still far below the human. Hence, current OCR research is being done on improving accuracy and speed of OCR for diverse style documents printed/ written in unconstrained environments.

• There is no availability of open source or commercial software available for complex languages.

• Tesseract, a highly popular OCR engine, was originally developed by Hewlett Packard in the 1980s and was then open-sourced in 2005. Google adopted the project in 2006 and has been sponsoring it ever since.

IV. METHODOLOGY

Summary of the natural scene text detection challenges:

A. • Image/sensor noise: Sensor noise from a handheld camera is typically higher than that of a traditional scanner. Additionally, low-priced cameras will typically interpolate the pixels of raw sensors to produce real colours.

B. • Viewing angles: Natural scene text can naturally have viewing angles that are not parallel to the text, making the text harder to recognize.

C. • Blurring: Uncontrolled environments tend to have blur, especially if the end user is utilizing a smartphone that does not have some form of stabilization.

D. • Lighting conditions: We cannot make any assumptions regarding our lighting conditions in natural scene images. It may be near dark, the flash on the camera may be on, or the sun may be shining brightly, saturating the entire image.

E. • Resolution: Not all cameras are created equal — we may be dealing with cameras with sub-par resolution.

F. • Non-paper objects: Most, but not all, paper is not reflective (at least in context of paper you are trying to scan). Text in natural scenes may be reflective, including logos, signs, etc.

G. • Non-planar objects: Consider what happens when you wrap text around a bottle — the text on the surface becomes distorted and deformed. While humans may still be able to easily “detect” and read the text, our algorithms will struggle. We need to be able to handle such use cases.

H. Unknown layout: We cannot use any a priori information to give our algorithms “clues” as to where the text resides.

V. IMPLEMENTATION

A. • ALGORITHM:

- We call the algorithm "EAST" because it's an: **E**fficient and **A**ccurate **S**cene **T**ext detection pipeline.
- We import NumPy, OpenCV
- We load and copy our input image.
- Determine the ratio of the original image dimensions to new image dimensions
- Then we resize the image, ignoring aspect ratio
- The first layer is our output sigmoid activation which gives us the probability of a region containing text or not.
- The second layer is the output feature map that represents the "geometry" of the image — we'll be able to use this geometry to derive the bounding box coordinates of the text in the input image
- he output geometry map used to derive the bounding box coordinates of text in our input images
- And similarly, the scores map, containing the probability of a given region containing text.
- We start off by grabbing the dimensions of the scores volume and then initializing two lists:

1. rects: Stores the bounding box (x, y)-coordinates for text regions

2. confidences: Stores the probability associated with each of the bounding boxes in rects

- We need to filter out weak text detections by ignoring areas that do not have sufficiently high probability
- We then update our rects and confidences lists, respectively
- The final step is to apply non-maxima suppression to our bounding boxes to suppress weak overlapping bounding boxes and then display the resulting text predictions:

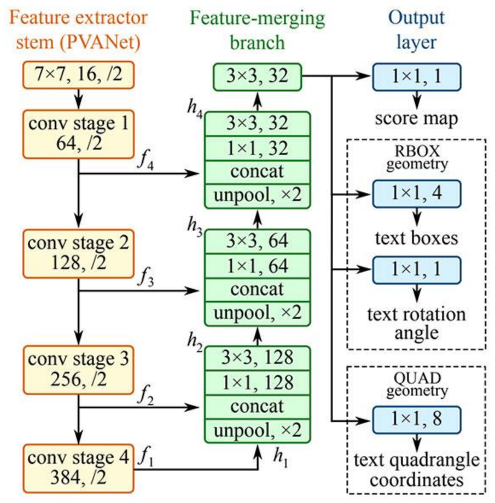


Figure 1Figure 1Efficient and Accurate Scene Text detection pipeline

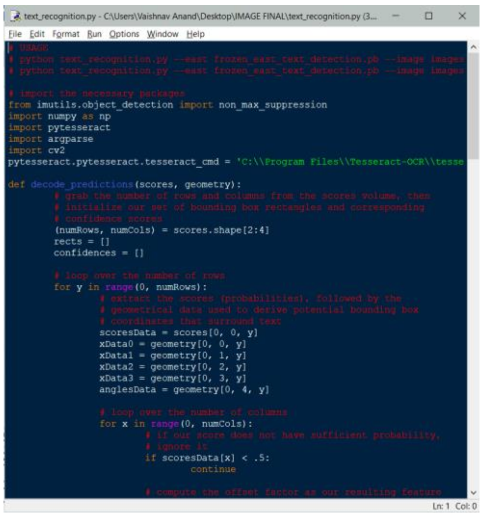
B. • CODE

```
# import the necessary packages
from imutils.object_detection import
non_max_suppression
import numpy as np
import argparse
import time
import cv2
# construct the argument parser and parse the
arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str,
help="path to input image")
ap.add_argument("-east", "--east", type=str,
help="path to input EAST text detector")
ap.add_argument("-c", "--min-confidence", type=float,
default=0.5,
help="minimum probability required to inspect a
region")
ap.add_argument("-w", "--width", type=int,
default=320,
help="resized image width (should be multiple of 32)")
ap.add_argument("-e", "--height", type=int,
default=320,
help="resized image height (should be multiple of 32)")
args = vars(ap.parse_args())
# load the input image and grab the image dimensions
image = cv2.imread(args["image"])
orig = image.copy()
(H, W) = image.shape[:2]
# set the new width and height and then determine the
ratio in change
# for both the width and height
(newW, newH) = (args["width"], args["height"])
rW = W / float(newW)
rH = H / float(newH)
```

```
# resize the image and grab the new image dimensions
image = cv2.resize(image, (newW, newH))
(H, W) = image.shape[:2]
# define the two output layer names for the EAST detector model that
# we are interested -- the first is the output probabilities and the
# second can be used to derive the bounding box coordinates of text
layerNames = [
"feature_fusion/Conv_7/Sigmoid",
"feature_fusion/concat_3"]
# load the pre-trained EAST text detector
print("[INFO] loading EAST text detector...")
net = cv2.dnn.readNet(args["east"])
# construct a blob from the image and then perform a forward pass of
# the model to obtain the two output layer sets
blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
(123.68, 116.78, 103.94), swapRB=True, crop=False)
start = time.time()
net.setInput(blob)
(scores, geometry) = net.forward(layerNames)
end = time.time()
# show timing information on text prediction
print("[INFO] text detection took {:.6f} seconds".format(end - start))
# grab the number of rows and columns from the scores volume, then
# initialize our set of bounding box rectangles and corresponding
# confidence scores
(numRows, numCols) = scores.shape[2:4]
rects = []
confidences = []
# loop over the number of rows
for y in range(0, numRows):
# extract the scores (probabilities), followed by the geometrical
# data used to derive potential bounding box coordinates that
# surround text
scoresData = scores[0, 0, y]
xData0 = geometry[0, 0, y]
xData1 = geometry[0, 1, y]
xData2 = geometry[0, 2, y]
xData3 = geometry[0, 3, y]
anglesData = geometry[0, 4, y]
# loop over the number of columns
for x in range(0, numCols):
# if our score does not have sufficient probability, ignore it
if scoresData[x] < args["min_confidence"]:
continue
# compute the offset factor as our resulting feature maps will
# be 4x smaller than the input image
(offsetX, offsetY) = (x * 4.0, y * 4.0)
# extract the rotation angle for the prediction and then
# compute the sin and cosine
angle = anglesData[x]
cos = np.cos(angle)
sin = np.sin(angle)
# use the geometry volume to derive the width and height of
# the bounding box
h = xData0[x] + xData2[x]
w = xData1[x] + xData3[x]
# compute both the starting and ending (x, y)-coordinates for
# the text prediction bounding box
endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
startX = int(endX - w)
startY = int(endY - h)
# add the bounding box coordinates and probability score to
# our respective lists
rects.append((startX, startY, endX, endY))
confidences.append(scoresData[x])
# apply non-maxima suppression to suppress weak, overlapping bounding
# boxes
boxes = non_max_suppression(np.array(rects), probs=confidences)
# loop over the bounding boxes
for (startX, startY, endX, endY) in boxes:
# scale the bounding box coordinates based on the respective
# ratios
startX = int(startX * rW)
startY = int(startY * rH)
endX = int(endX * rW)
endY = int(endY * rH)
# draw the bounding box on the image
cv2.rectangle(orig, (startX, startY), (endX, endY), (0, 255, 0), 2)
# show the output image
cv2.imshow("Text Detection", orig)
```

cv2.waitKey(0)

VI. RESULTS



```
# USAGE
# python text_recognition.py --east frozen_east_text_detection.pb --image images
# python text_recognition.py --east frozen_east_text_detection.pb --image images

# import the necessary packages
from imutils.object_detection import non_max_suppression
import numpy as np
import pytesseract
import argparse
import cv2
pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesse

def decode_predictions(scores, geometry):
        # grab the number of rows and columns from the scores volume, then
        # initialize our set of bounding box rectangles and corresponding
        # confidence scores
        (numRows, numCols) = scores.shape[2:4]
        rects = []
        confidences = []

        # loop over the number of rows
        for y in range(0, numRows):
                # extract the scores (probabilities), followed by the
                # geometrical data used to derive potential bounding box
                # coordinates that surround text
                scoresData = scores[0, 0, y]
                xData0 = geometry[0, 0, y]
                xData1 = geometry[0, 1, y]
                xData2 = geometry[0, 2, y]
                xData3 = geometry[0, 3, y]
                anglesData = geometry[0, 4, y]

                # loop over the number of columns
                for x in range(0, numCols):
                        # if our score does not have sufficient probability,
                        # ignore it
                        if scoresData[x] < .5:
                                continue

                        # compute the offset factor as our resulting feature
```

Figure 3 CODE implementation

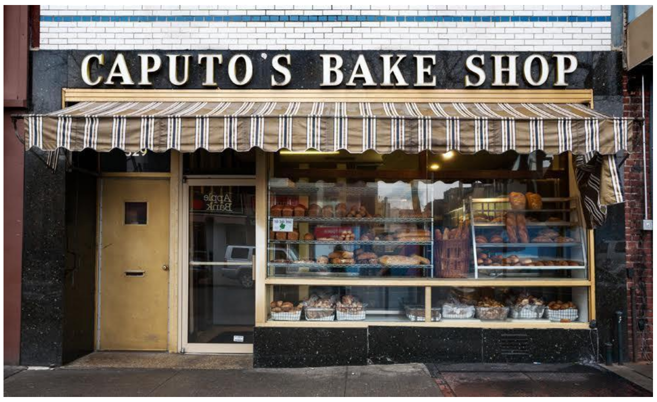


Figure 7INPUT IMAGE

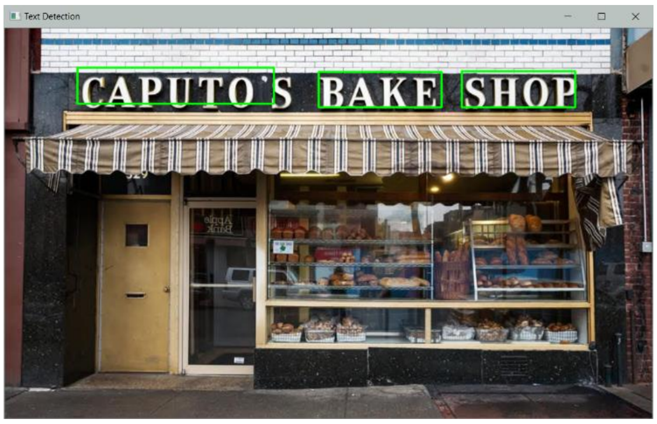


Figure 6 Character Detection

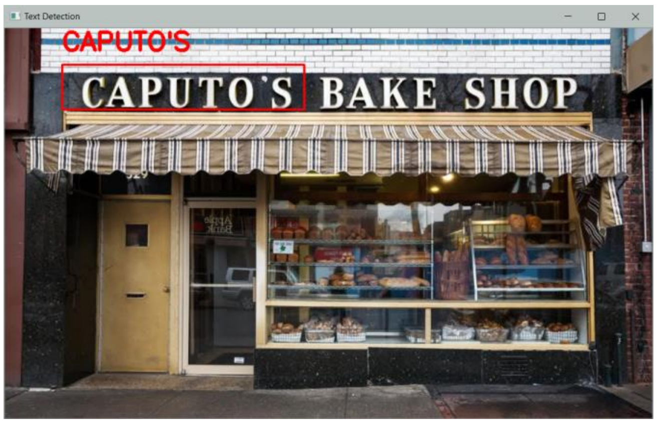


Figure 8Recognised words

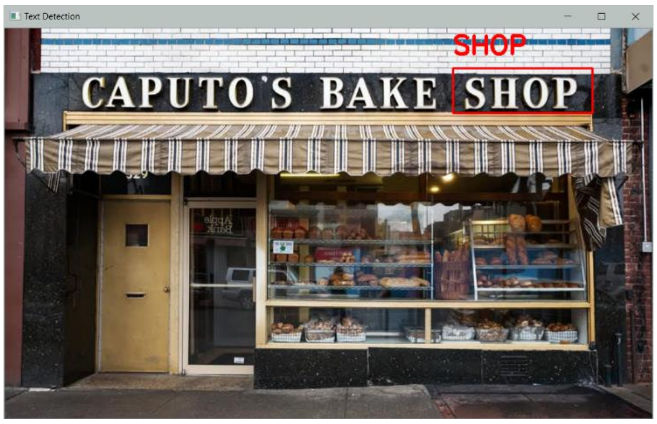


Figure 2Recognised Words

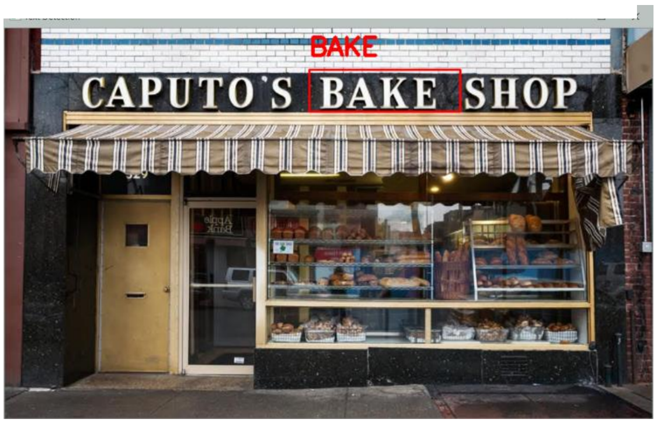
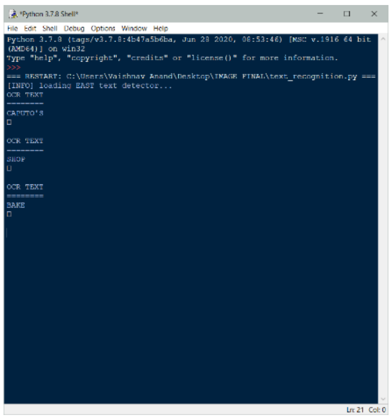


Figure 5Recognised Words



```
Python 3.7.8 (tags/v3.7.8:4b47a5b6ba, Jun 28 2020, 08:53:46) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\Vaishnav Anand\Desktop\IMAGE FINAL\text_recognition.py ===
[INFO] loading EAST text detector...
OCR TEXT
--------
CAPUTO'S
□

OCR TEXT
--------
SHOP
□

OCR TEXT
========
BAKE
□
```

Figure 4Output Characters

VII. CONCLUSION

- By using above proposed algorithm or the EAST detector we can obtain the accurate and speed output of the characters in the given or present natural scene.
- So, it may be useful in character recognition in the natural scenes such as recognition of sign boards, name boards and many in the real life day to day scenes.

VIII. REFERENCES

https://shodhganga.inflibnet.ac.in/bitstream/10603/4166/10/10_chapter%202.pdf

https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/

https://www.researchgate.net/publication/267426877_Facial_Recognition_using_OpenCV

,