OCR Image Sort: Image Sorting and Classification via Clustering, Text Detection, and Text Recognition

Nond Hasbamrer
Department of Electrical Engineering
Columbia University
New York, NY 10027, USA
hasbamrer.nond@columbia.edu

***Abstract*— In this paper, I propose a system for automatically sorting images into folders by using text detection and text recognition to extract descriptors from key images. The input dataset must be a sequence of images where an image containing the text descriptor for a subset of images precedes the subset in the overall sequence. The system uses k-Means clustering on grayscale image histogram data to filter potential key images from the dataset. Potential key images are then passed into the EAST deep learning CNN model and OpenCV to detect text regions in key images. These text regions then go through Tesseract's LSTM deep learning engine for text recognition. Afterwards, Python wrappers use the output from Tesseract to create folders on disk and move images into the appropriate folders. The OCR Image Sort system has been evaluated on a 3.9 GB dataset of 1433 images taken in North American refineries resulting in a precision of 83.1% and recall of 90.2%. Of the correct true positive key images, 81.1% had perfect text recognition and 18.9% had text recognition errors.**

Keywords— Computer vision, optical character recognition, k-Means clustering, deep learning, text detection, text recognition, image sort

I. Introduction

As part of my job, I visit refineries across the country and survey numerous pieces of existing equipment in the refinery. During these surveys, my team and I will take thousands of photos of numerous pieces of equipment and spend hours sorting through the photos after the survey. Each equipment typically will have a mounted tag or nameplate. A typical set of photos for a piece of equipment will start with a photo of the equipment tag followed by a series of photos of the equipment itself. For example, if each photo is denoted by p_j^i where *i* is the equipment number and *j* is the image sequence for that equipment, then photos for three pieces of adjacent equipment may follow:

$$photo\ set = \{p_1^1, p_2^1, p_3^1, p_1^2, p_2^2, p_3^2, p_4^2, p_1^3, p_2^3\}$$

In order to organize and label the photos, my team has to manually look at each photo, find the photo with the equipment tag, and label all the following photos with that equipment tag until a photo of a new equipment tag comes up. The goal of this project is to find a way automate this process of labeling and sorting photos by tag.

Works by Chandel and Mallick [1], Shrimali and Mallick [2], and Rosebrock [3] demonstrated OpenCV performing text detection using Zhou et al.'s EAST model [5] and text recognition using Tesseract [4] on single still images. The algorithms for text detection and text recognition could help automate the process of finding and reading equipment tags from key photos in a sequence of images.

II. System Scope

The proposed system processes a sequential set of equipment images and identifies key images. A key image is a photo that contains the tag or nameplate of an equipment and is the first photo in a photo set of that equipment. In the above example, key photos are $p_1^1, p_1^2, and\ p_1^3$. The system is built to run on a personal desktop or laptop computer, but not on a mobile platform such as a phone or camera. The system will sort all the photos associated with the key photo into a single folder (see Fig. 1 below).

The system is based on six main technologies: Python 3 [9], Tesseract [4], the EAST model [5], OpenCV [6], PyTesseract[7], and scikit-Learn [8]. Python3 is the general wrapper for all system components and interacts with the operating system to create folders and move image files around. Image filtering to identify key images is done via scikit-learn's k-Means clustering algorithm. Text detection is done by OpenCV with the EAST model. After OpenCV identifies the text regions, Tesseract does the text recognition and outputs a string back to Python3. PyTesseract provides a Python wrapper for Tesseract. See figure 2 below for a diagram of the system components.

III. Limitations on the Image Dataset

The images in the dataset are limited to in-focus photos taken with ample lighting such that any relevant text in the photo can

Columbia University COMS E6735 Spring 2019 Course Project
Professor John R. Kender (jrk@cs.columbia.edu)

Apache License Version 2.0 © 2019 Nond Hasbamrer

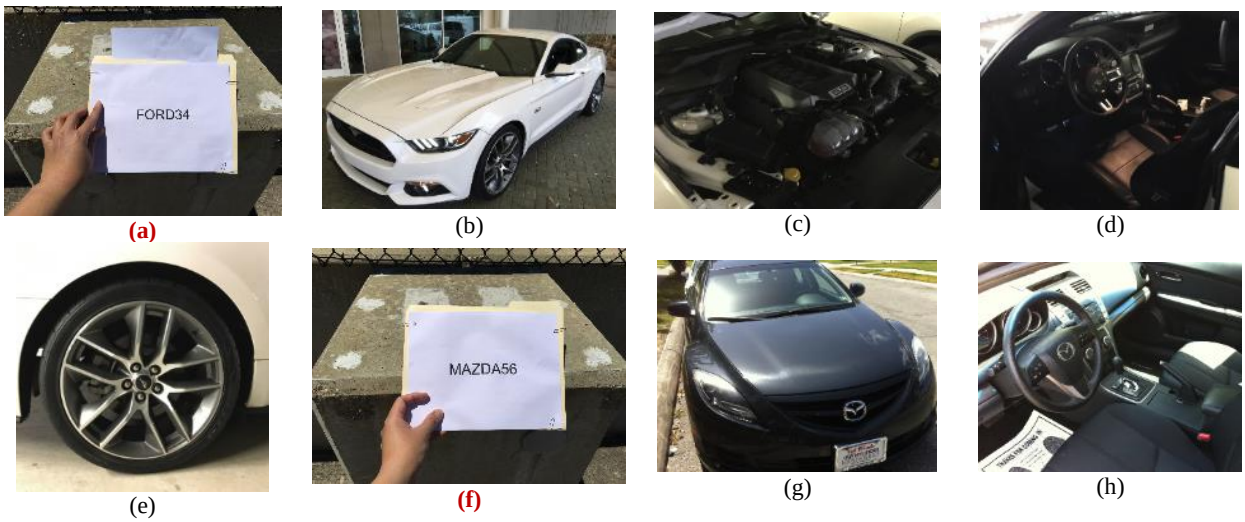


Fig. 1. In this example set of 8 sequential car images, the key images are image 'a' and image 'f'. The system will detect the descriptor in image 'a' and put images 'a' through 'e' into a common folder. When the system sees a text descriptor in image 'f' and identifies this as a key image, the system will create a new folder based on the label in image 'f' and sort all images after 'f' into that folder.

be easily read by a person viewing the image on a 24inch 1080p TN panel computer monitor. The images are stored in JPEG (PNG works too) with a resolution of approximately 4608 pixels by 3456 pixels. Depending on the content of the image, the size of these images may range from approximately 1.5MB to 4.3MB.

In order to increase the success rate of key image identification and successful optical character recognition, the content of key images and non-key images is limited. Key images must contain a single label in black text on white background. The label area is centered in the middle of the image, right side up, in focus, and in the same plane as the camera. Note that the label must be printed using a common font and cannot be handwritten. The version of Tesseract used in this program was only trained on printed text and was not trained to recognize handwritten alphanumeric characters. In order to avoid false-positive key photo classification non-key images may contain text, but the text must not be prominent and not centered, occupy less than 10% of the image, or have different colored lettering on different colored background.

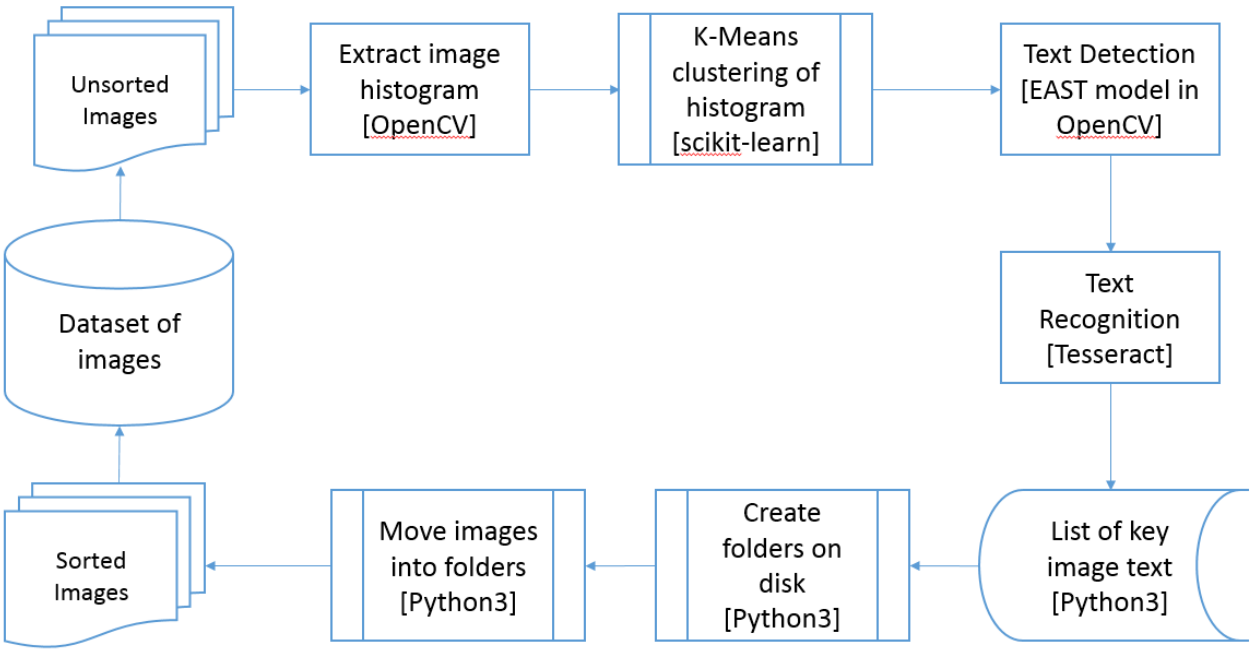


Fig. 2. An overview of the different system components of the image sorting and classification project.

The images in the small test dataset (34 images) were captured using an iPhone 6s at full resolution using the front 5 megapixel camera. These images have a resolution of 4032 by 3024 pixels and are stored in JPEG format.

The images in the large evaluation dataset (1433 images of equipment in North American refineries) were captured using a FujiFilm FinePix XP140 camera at full resolution. These images have a resolution of 4608 by 3456 pixels and are stored in JPEG format.

IV. Development Environment

The entire system was developed using the Spyder IDE (version 3.3.3) [PK5] contained in the Anaconda package manager (Anaconda Distribution version 2019.03 with Anaconda Navigator version 1.9.7) [PK4] running on a Windows 7 personal computer. The Anaconda platform allows the user to create virtual environments that contain specific versions of Python and auxiliary packages that are isolated from other virtual environments. The OCR Image Sort system uses Python 3.7.3, OpenCV version 4.1.0, Tesseract version 4.0.0.20190304, PyTesseract version 0.2.7, and scikit-learn version 0.20.3.

V. System Set Up and Dependency Installation

To set up the system, first install Tesseract for windows by downloading the installer from the link in [PK8]. Then add Tesseract to the windows PATH variable. Install the latest version of Anaconda [PK4]. Open Anaconda Prompt, and run the following commands **in order** to install all supporting packages:

1. conda install pip
2. pip install tesseract
3. pip install pytesseract
4. pip install opencv-contrib-python
5. pip-install imutils
6. conda install tensorflow-gpu

VI. Code Overview

The software is contained in one class called ImageSort. The location of the EAST model, the unsorted photos folder path, and the sorted photos folder path is initialized with the class. The EAST model is also loaded into memory during class initialization.

```
class ImageSort:
    def __init__(self):
        self.folderPath='./input'
        self.sortedPath='./output'

self.eastPath='./frozen_east_text_detection.pb'
        self.net=cv2.dnn.readNet(self.eastPath)
```

The ImageSort class has two main execution modes: text detection with OCR mode and an OCR only mode. The text detection with OCR mode is the default and recommended execution mode as this mode is faster and produces more accurate results. This mode uses OpenCV to extract a greyscale image histogram of each image, scikit-learn's k-Means clustering algorithm to identify potential key images based on the histogram of each image, OpenCV and the EAST model to detect text regions in images, and then passes the text regions to Tesseract for recognition. The OCR only mode passes the entirety of all images to Tesseract. This mode is slower as tesseract has to process the entire image instead of just the text region and may also produce more false positive key image identifications.

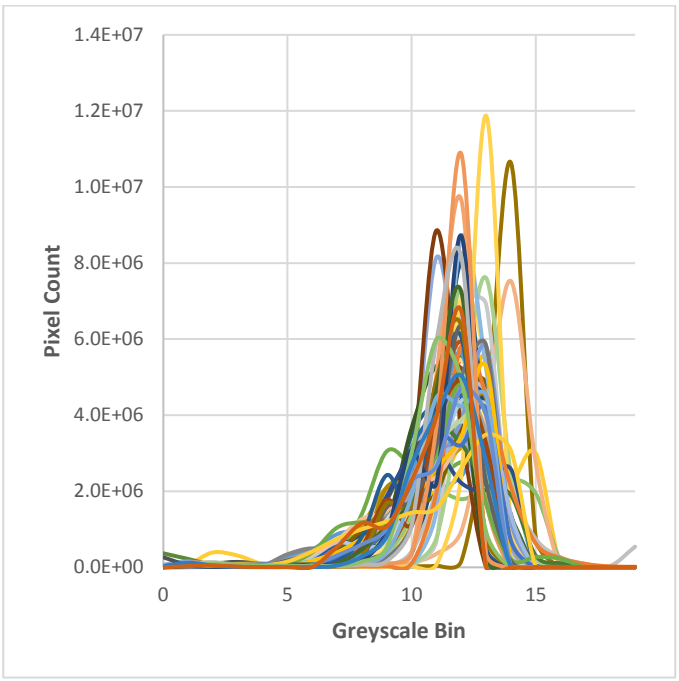
A. Text Detection with OCR mode:

The main function for the text detection with OCR mode is runDefault().

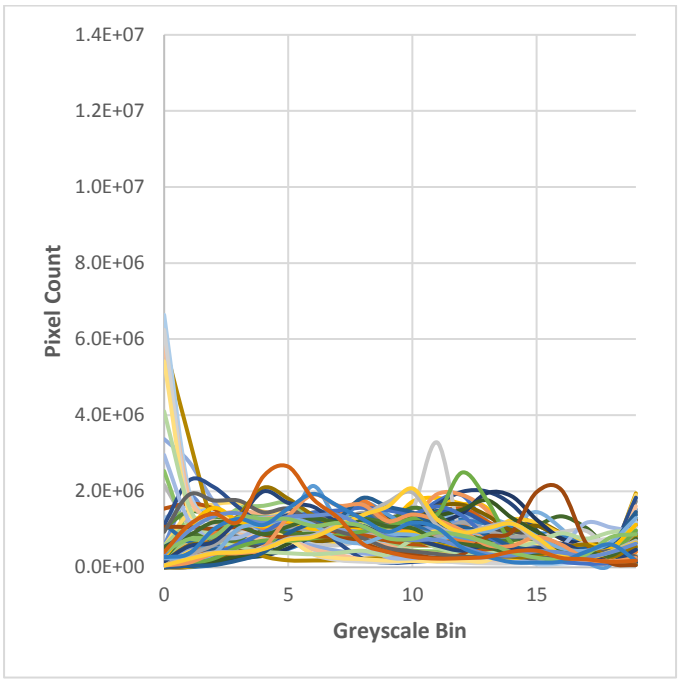
```
def runDefault(self):
    start=time.time()
    fileNames=self.getFileNames()
    print('Gathering histogram data...')
    histoAll=self.getHisto()
    print('Histogram Extraction Complete')
    print('Clustering...')
    clusterAssign=self.findKeyPhotos(histoAll)
    print('K-Means Clustering Complete')
    print('Begin text detection and OCR...')
    imgText=self.textDetectAndRecogAll \
    (clusterAssign)
    print('Sorting Complete')
    self.makeFolders(imgText)
    imgToFolder=self.folderMap(imgText)
    self.sortImages(imgToFolder,fileNames)
    end=time.time()
    #print(imgText)
    print('OCR Image Sort Complete')
    print('Execution time (s): ' + str(end-start))
    return
```

This function first starts the execution timer then calls getFileNames() which scans the folder ./input/ and returns a list of all the image filenames in that folder. Next, the function calls getHisto() which uses OpenCV to produce a an array containing greyscale image histograms of all images (see Fig. 3). The histogram contains 20 bins with each pixel value ranging from 0 to 256. These parameters were determined experimentally. The array of histograms is passed into the findKeyPhotos() function which fits a k-Means clustering algorithm with 2 clusters to the histogram data. Each bin in the histogram data is considered a feature in the k-Means clustering algorithm. The function returns the cluster assignments of each image.

Under the assumption that there are less key images than non-key images in the dataset, the function textDetectAndRecogAll() loops through all image files in the smaller cluster and calls the textDetection function. The textDetection function uses the EAST model to identify regions of text within the image and then passes these text regions into Tesseract for text recognition (see Fig. 4).



(a) Image Histogram of 74 Key Images



(b) Image Histogram of 74 Non-Key Images

Fig. 3. The greyscale image histogram contains 20 bins ranging from black at bin 0 to white at bin 19. The image histogram for key images (a) peak between bins 10 and 15 while the image histogram of non-key images (b) are much flatter and do not have peaks as prominent as the key images. Since key-image histograms have a characteristic shape while non-key image histograms are much noisier, the pixel counts per bin can be used by a k-Means clustering algorithm to classify potential key images. Note that peaks in (a) are consistent with the constraint that key images must be a picture of printed black text on a white background. Due to the lighting conditions of the refinery plants, the key images are not completely saturated at bin 19, but peak at an off-white shade instead.

The text output from Tesseract gets appended to a list of all text strings within that image and the longest string of text is returned as the most likely label for that potential key image. This list of key image text is then used by the makeFolders function to create folders in the ./output/ folder named after each key image text. The folderMap function takes the key image text list and maps where each image in the ./intput/ folder should go in the ./output/ folder based on which key image precedes each

(a)

(b)

Fig. 4. (a) Text detection bounding box from the EAST model. (b) Cropped image sent to Tesseract for OCR.

regular image. The sortImages function uses this map to move all of the images from the ./input/ folder into the appropriate subfolder in ./output/ (see Fig. 5).

B. *OCR only mode:*

The main function for the OCR only mode is runTextRecogOnly(). This function is similar to the main function for the text detection with OCR mode except for the function creates the list of key image text using readAllUnsorted() instead of textDetectandRecogAll(). In readAllUnsorted(), the function loops through all the images and calls Tesseract on the full image instead of on small cropped images passed from a text detector. After the key image text list is created, the program runs through the same functions as above to create folders and move image files.

VII. Code Execution Instructions

Create a folder called ./input/ and another folder called ./output/ in the same directory as the code. Make sure the folder paths in the ImageSort class initialization for the input and output folders point to the correct location. Check that the location of the EAST model is correct.

A. *Running the program from a Terminal:*

To run the program from the terminal, make sure that the terminal directory as the folder where imagesort.py, the EAST model, and the input and output folders are located. Run the program using:

> python imagesort.py

No additional inputs or options are required.

B. *Running the program from a Python 3 console for OCR only mode:*

First, comment out the default run script at the bottom of the imagesort.py file. Run imagesort.py to load in the code. Initialize the class with.

```
iSort=ImageSort()
```

To run the code with text detection + OCR (recommended), run the following command in the Python terminal after initializing the class:

```
iSort.runDefault()
```

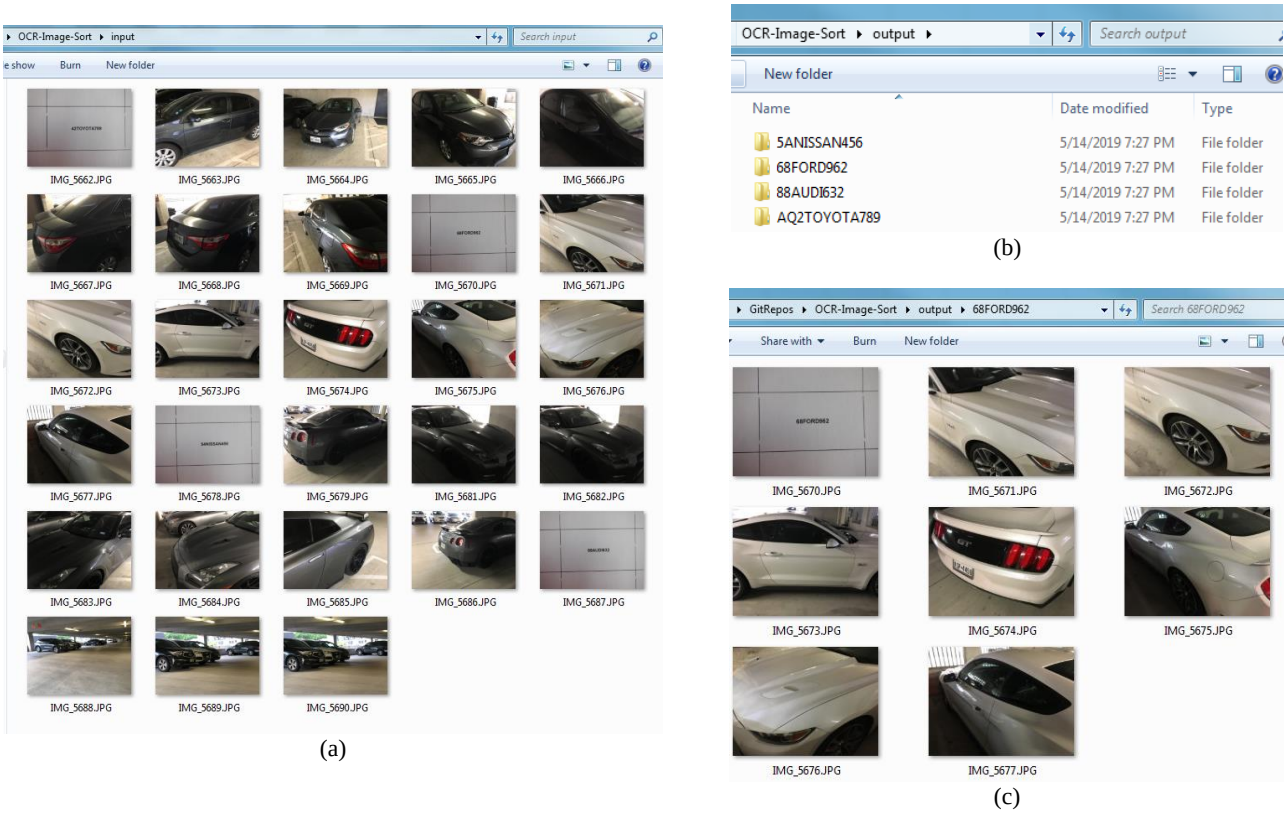


Fig. 5. (a) Input folder with 28 sequential images. Key images are the 1st, 9th, 17th, and 25th image. (b) Folders automatically created based on the text detection and OCR results. (c) The key image and all sequential images after that key image up to the next key image placed in the same folder.

To run the code with OCR only (not recommended), run the following command in the Python terminal after initializing the class:

```
iSort.runTextRecogOnly()
```

VIII. System Evaluation Environment

The prototype was tested on a personal computer with the following specifications:

- OS: Windows 7 Service Pack 1 64-bit
- CPU: Intel Core i5-4690K 3.50 GHz Quad Core
- GPU: Nvidia GeForce GTX 1080
- Ram: 16.0 GB
- OS Drive: Samsung SSD 850 EVO
- Program Drive: Toshiba SSD OCZ Trion 150
- Motherboard: MSI Z97 Gaming 5

During the test, the computer was air cooled and not overclocked. The ambient temperature of the testing room was approximately 70-72°F.

I used Python's native time.time() method to test the execution time. Note that the time it takes to load the EAST model into memory is not included in these measurements since the model is loaded into memory when the class object is created and not reloaded each time the script is executed.

IX. System Evaluation Results

For the full system evaluation, I used the OCR Image Sort system to sort a 3.9 GB dataset of 1433 images of over 100 different pieces of equipment (sensors, control valves, piping, enclosures, etc) taken in North American refineries and compared the system's automatic sort to a manual sort conducted by a subject matter expert. Each image has a resolution of 4608 by 3456 pixels, is stored in JPEG format, and ranges from 1.5MB to 4.3MB in size. The system sorted 1433 images in 327.984 seconds (0.229 seconds per image). The results are in Table 1 below.

TABLE I. Confusion Matrix for the Full System Evaluation

Trial n=1433	*Actual key photo*	*Actual non-key photo*	
Predicted Key photo	74	15	89
Predicted non-key photo	8	1337	1345
	82	1352	

Based on this confusion matrix, the system yielded a precision of 83.1% with a recall of 90.2%. Of the 74 correct key image identifications, 60 images (81.1%) had perfect text recognition and 14 images (18.9%) had text recognition errors. These results are a significant improvement over the prototype system. The prototype system had a precision of < 50% and a recall of ~75%. The prototype also took 0.478 seconds per image to process images at a lower resolution.

The large increase in both precision and recall comes from the addition of image histogram k-Means clustering before text detection and recognition. The clustering step filters key images from the dataset much more precisely than the prototype's simple heuristics on the number of text bonding boxes detected. Extracting image histogram data and performing k-Means clustering on the entire dataset is also much faster than performing text detection on the entire dataset.

X. Conclusions

This paper combines k-Means clustering, text detection, and text recognition to create an automatic photo sorter based on text within key images among a sequence of images. The system uses a combination of Python 3, OpenCV, EAST model, and Tesseract to extract key text, create and label folders on-disk, and move images into the appropriate folder. Testing on a 3.9 GB dataset of 1433 images yielded a total execution time of 327.984 seconds (0.229 seconds per image) with a precision of 83.1% and a recall of 90.2%. Of the correct true positive key images, 81.1% had perfect text recognition and 18.9% had text recognition errors.

Previous systems combining text detection and text recognition have been employed for extracting text from single still images or in real time from a camera. However, the combination of text detection and recognition have not been exploited as an image sorter for a sequence of still images.

Even though the OCR Image Sort system does not have perfect precision and recall, users can still use the system to automatically sort images then make manual corrections to the automatic sort errors. Sorting 1433 images in less than 6 minutes automatically and then spending an hour correcting the sort errors is much quicker than spending 5-6 hours sorting images solely by hand.

This system could also be used for other applications outside of refinery image sorting that involve segregating a sequence of images by key images in the sequence. School yearbook photographers that take photos of students where the first photo of each student is the student holding a name card could use this system to automatically sort and label photos from the shoot. Car dealerships could use this system to automatically sort photos of taken of incoming car inventory. Travelers could use this system to sort photos of cities, towns, and tourist attractions that they visit based on photos of different signage.

XI. Future Work

The current version uses a text based approach to execute the algorithm and provides no way for the user to verify whether the predicted key image is correct. In version 2.0 of OCR Image Sort, a GUI could be built that allows the user to verify the key images before the text detection and text recognition stage. This will help reduce the number of false positive key photos. The GUI could also allow the user to correct any text recognition errors before the program creates the folders and move the images. With this approach, the user won't have to click through each folder and compare the folder name to the key image to verify the text recognition results. Since some engineering disciplines prefer to hand write notes for key images instead of using a printed list, the next version of OCR Image Sort could incorporate handwriting recognition into the OCR portion of the program.

XII. Lessons Learnt from the Project

Completing this project taught me that designing my computer vision algorithm with domain conscious feature selection can have a very big impact on the success of the algorithm. Initially, I wrote my algorithm as a general purpose image sorter without thinking about the unique image properties of the type of images that I was analyzing. I kept focusing on optimizing my algorithm based on the content of the image, but not the data of the image. I realized that I was trying to tune the computer based on human notions of the data. Once I started to consider the characteristics of the images as pure numerical data and abstract away from the content of the image, I was able to apply k-Means clustering on the image histogram. This shift in thinking significantly increased the success rate of my program and reduced the execution time by an order of magnitude. I also learnt that Tesseract's optical character recognition, though good, can still make mistakes on photos of printed text. Tesseract worked very well on flat scans of text, but slight variations in camera tilt and roll can cause OCR errors.

Acknowledgements

Thank you Professor John Kender and Teaching Assistant Zhou Zhuang for feedback and guidance on this project.

References

[1] V. Singh Chandel and S. Mallick, "Deep Learning based Text Recognition (OCR) using Tesseract and OpenCV," *Learn OpenCV*, 2018. Available: https://www.learnopencv.com/deep-learning-based-text-recognition-ocr-using-tesseract-and-opencv/

[2] V. Shrimali and S. Mallick, "Deep Learning based Text Detection Using OpenCV," *Learn OpenCV*, 2019. Available: https://www.learnopencv.com/deep-learning-based-text-detection-using-opencv-c-python/

[3] A. Rosebrock, "OpenCV OCR and text recognition with Tesseract," *pyimagesearch*, 2019. Available: https://www.pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/

[4] R. Smith, "An Overview of the Tesseract OCR Engine," presented at the Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02, 2007. Available: https://github.com/tesseract-ocr/tesseract

[5] X. Zhou et al., "EAST: an efficient and accurate scene text detector," in Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2017, pp. 5551-5560. Available: https://github.com/argman/EAST

[6] *The OpenCV Library*. (2000). Accessed: February 28, 2019. [Online]. Available: https://opencv.org/

[7] M. Lee et al., "A Python Wrapper for Google Tesseract," *Python Tesseract*. (2019). Accessed: May 13, 2019. [Online]. Available: https://github.com/madmaze/pytesseract

[8] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of machine learning research, vol. 12, no. Oct, pp. 2825-2830, 2011.

[9] Python Software Foundation. Python Programming Language, version 3.7.3. Available: https://www.python.org/

Main APIs and Software Package Links

[PK1] https://github.com/opencv/opencv

[PK2] https://github.com/tesseract-ocr/tesseract

[PK3] https://github.com/madmaze/pytesseract

[PK4] Python 3.7.3 |Anaconda custom (64-bit)| via https://www.anaconda.com/

[PK5] Spyder IDE: https://www.spyder-ide.org/

[PK6] EAST Text Detection model: https://www.dropbox.com/s/r2ingd0l3zt8hxs/frozen_east_text_detection.tar.gz?dl=1

[PK7] https://github.com/argman/EAST

[PK8] https://github.com/UB-Mannheim/tesseract/wiki

APPENDIX A: OCR IMAGE SORT CODE

The full repository can be downloaded from https://github.com/Aneapiy/OCR-Image-Sort

```
# -*- coding: utf-8 -*-
"""
OCR-Image-Sort
Image Sorting and Classification via Clustering, Text Detection, and Text Recognition
Created by Nond on April 13, 2019
"""
import cv2
import numpy as np
import pytesseract
import os
import time
from imutils.object_detection import non_max_suppression
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

class ImageSort:
    def __init__(self):
        self.folderPath='./input'
        self.sortedPath='./output'
        self.eastPath='./frozen_east_text_detection.pb'
        self.net=cv2.dnn.readNet(self.eastPath)

    def readImage(self, imgPath):
        #read an image with OpenCV
        #this function is based on references [1,4]
        img=cv2.imread(imgPath) #default color image

        #config parameters for pytesseract
        #'-l eng' sets English as the language
        #'--oem 1' sets LSTM deep learning OCR Engine
        #'--psm 3' default PSM, fully automatic
        config=('-l eng --oem 1 --psm 3')

        #Run Tesseract OCR
        text=pytesseract.image_to_string(img, config=config)

        return text

    def getFileNames(self):
        #creates list of image file names
        fileNames=os.listdir(self.folderPath)
        return fileNames

    def readAllUnsorted(self):
        #reads all image text in the unsorted folder
        fileNames=self.getFileNames()
        numOfImgs=len(fileNames)
        imgText=[]
        for i in range(numOfImgs):
            imgPath=self.folderPath+'/'+fileNames[i]
            imgText.append(self.readImage(imgPath))
        return imgText

    def makeFolders(self,imgText):
        #make folders based on image text
```

```
        for i in range(len(imgText)):
            if len(imgText[i])>0:
                folderName=self.sortedPath+'/'+imgText[i]
                try:
                    os.makedirs(folderName)
                except FileExistsError:
                    #directory of the same name already exists
                    pass
        return

    def folderMap(self,imgText):
        #creates a map of what folders which image goes to
        imgToFolder=[]
        for i in range(len(imgText)):
            if len(imgText[i])>0:
                folderName=self.sortedPath+'/'+imgText[i]
                imgToFolder.append(folderName)
            elif len(imgToFolder)>0:
                imgToFolder.append(imgToFolder[i-1])
            else:
                print("No Key Images")
                return
        return imgToFolder

    def sortImages(self,imgToFolder,fileNames):
        #takes the folder map and moves the images to that folder
        #the folder map and the file name list must be in the same order
        for i in range(len(imgToFolder)):
            source_file=self.folderPath+'/'+fileNames[i]
            destination_file=imgToFolder[i]+'/'+fileNames[i]
            os.rename(source_file,destination_file)
        return

    def unsortImages(self,imgToFolder,fileNames):
        #moves all images back to the unsorted folder
        for i in range(len(imgToFolder)):
            source_file=self.folderPath+'/'+fileNames[i]
            destination_file=imgToFolder[i]+'/'+fileNames[i]
            os.rename(destination_file,source_file)
        return

    def decode_predictions(self,scores,geometry):
        # this function is copied directly from reference [3]
        # with minor modifications
        #---(start of function from reference [3])-------------------
        # 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
            min_confidence=0.6
            for x in range(0, numCols):
                # if our score does not have sufficient probability,
                # ignore it
                if scoresData[x] < 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])

    # return a tuple of the bounding boxes and associated confidences
    return (rects, confidences)
    #---(end of function from reference [3])-------------------

def textDetection(self, imgPath):
    #this function is based on references [2,3,5]
    img=cv2.imread(imgPath) #default color image
    origImg=img.copy()
    #resize image for EAST model.
    #EAST requires width and height to be multiple of 32
    (origH, origW)=img.shape[:2]
    (newW, newH)=(320,320)
    #ratio of scaled image to original image
    rW=origW/float(newW)
    rH=origH/float(newH)
    img=cv2.resize(img, (newW, newH))
    (H, W)=img.shape[:2]
    outputLayers=['feature_fusion/Conv_7/Sigmoid','feature_fusion/concat_3']
```

```
        #Note: EAST model is loaded when this class initiatlizes
        blob=cv2.dnn.blobFromImage(img,1.0,(W,H),(123.68, 116.78, 103.94),True,False)
        self.net.setInput(blob)
        (scores, geometry)=self.net.forward(outputLayers)
        (rects, confidences)=self.decode_predictions(scores,geometry)
        boxes=non_max_suppression(np.array(rects),probs=confidences)
        croppedImages=[]
        for (startX, startY, endX, endY) in boxes:
            #add padding to the bounding boxes, so we don't clip letters/nums
            padding=0.8
            padX=int((endX-startX)*padding)
            padY=int((endY-startY)*padding)

            startX=max(int(startX*rW)-padX,0)
            startY=max(int(startY*rH)-padY,0)
            endX=min(int(endX*rW)+padX,origW)
            endY=min(int(endY*rH)+3*padY,origH)

            cv2.rectangle(origImg,(startX,startY),(endX,endY),(0,255,0),8)
            croppedTextBox=origImg[startY:endY,startX:endX]
            croppedImages.append(croppedTextBox)

            #Uncomment the 4 commands below to show the image with text detection boxes
            '''
            cv2.namedWindow('cropped',cv2.WINDOW_NORMAL)
            cv2.imshow('cropped',croppedTextBox)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            '''
        #Uncomment the 4 commands below to show the image with text detection boxes
        '''
        cv2.namedWindow('Text Recognition',cv2.WINDOW_NORMAL)
        cv2.imshow('Text Recognition',origImg)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        '''
        #print(boxes)
        return croppedImages

    def textDetectAndRecogAll(self, clusterAssign):
        #reads all image text in the unsorted folder
        fileNames=self.getFileNames()
        imgText=[]
        #Assume that there are less key photos than non-key photos
        cnt=np.unique(clusterAssign, return_counts=True)
        keyPhotoNum=cnt[0][np.argmin(cnt[1])]
        for i in range(len(fileNames)):
            if clusterAssign[i]!=keyPhotoNum:
                imgText.append('')
                continue
            imgPath=self.folderPath+'/'+fileNames[i]
            croppedImages=self.textDetection(imgPath)

            #If there's too many text regions, this might be a
            #non-key photo with mulitple text regions
            '''
            if len(croppedImages)>4:
                continue
```

```
            '''
            croppedText=[]
            for j in range(len(croppedImages)):
                #config parameters for pytesseract
                #'-l eng' sets English as the language
                #'--oem 1' sets LSTM deep learning OCR Engine
                #'--psm 7' treat image as a single line of text
                config=('-l eng --oem 1 --psm 7')
                #Run Tesseract OCR
                text=pytesseract.image_to_string(croppedImages[j], config=config)
                text=''.join(filter(str.isalnum,text))
                #Tags are generally longer than 3 characters
                if len(text)>3:
                    croppedText.append(text)
            #pick the longest string found in the key image
            if len(croppedText)>0:
                imgText.append(max(croppedText,key=len))
            else:
                imgText.append('')
        #print(imgText)
        return imgText

    def getHisto(self):
        histBins=20
        fileNames=self.getFileNames()
        histoAll=np.zeros((len(fileNames),histBins))
        for i in range(len(fileNames)):
            imgPath=self.folderPath+'/'+fileNames[i]
            img=cv2.imread(imgPath,0)
            hist=cv2.calcHist([img],[0],None,[histBins],[0,256])
            histoAll[i]=hist.reshape(histBins)
            #Histogram with image
            #plt.subplot(121), plt.imshow(img,'gray')
            #plt.subplot(122), plt.plot(hist)
            #Histogram only
            #plt.plot(hist)
            '''
            plt.hist(img.ravel(),256,[0,256])
            plt.show()
            '''
            #Uncomment the 4 commands below to show the image with text detection boxes
            '''
            cv2.namedWindow('Grayscale',cv2.WINDOW_NORMAL)
            cv2.imshow('Grayscale',img)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            '''
        return histoAll

    def getColorHisto(self):
        histBins=20
        fileNames=self.getFileNames()
        histoAll=np.zeros((len(fileNames),histBins))
        for i in range(len(fileNames)):
            imgPath=self.folderPath+'/'+fileNames[i]
            img=cv2.imread(imgPath,0)
            hist=cv2.calcHist([img],[0],None,[histBins],[0,256])
            histoAll[i]=hist.reshape(histBins)
```

```
            #Histogram with image
            #plt.subplot(121), plt.imshow(img,'gray')
            #plt.subplot(122), plt.plot(hist)
            #Histogram only
            #plt.plot(hist)
            '''
            plt.hist(img.ravel(),256,[0,256])
            plt.show()
            '''
            #Uncomment the 4 commands below to show the image with text detection boxes
            '''
            cv2.namedWindow('Grayscale',cv2.WINDOW_NORMAL)
            cv2.imshow('Grayscale',img)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            '''
        return histoAll

    def findKeyPhotos(self, histo):
        #Use k-means clustering to group key photos together
        kmeans=KMeans(n_clusters=2,random_state=0).fit(histo)
        clusterAssign=kmeans.labels_
        #Try DBSCAN
        #Try Spectral Clustering
        return clusterAssign

    def runTextRecogOnly(self):
        start=time.time()
        fileNames=self.getFileNames()
        imgText=self.readAllUnsorted()
        self.makeFolders(imgText)
        imgToFolder=self.folderMap(imgText)
        self.sortImages(imgToFolder,fileNames)
        end=time.time()
        print(imgText)
        print('Execution time (s): ' + str(end-start))
        return

    def runDefault(self):
        start=time.time()
        fileNames=self.getFileNames()
        print('Gathering histogram data...')
        histoAll=self.getHisto()
        print('Histogram Extraction Complete')
        print('Clustering...')
        clusterAssign=self.findKeyPhotos(histoAll)
        print('K-Means Clustering Complete')
        print('Begin text detection and OCR...')
        imgText=self.textDetectAndRecogAll(clusterAssign)
        print('Sorting Complete')
        self.makeFolders(imgText)
        imgToFolder=self.folderMap(imgText)
        self.sortImages(imgToFolder,fileNames)
        end=time.time()
        #print(imgText)
        print('OCR Image Sort Complete')
        print('Execution time (s): ' + str(end-start))
        return
```

```
#Default run script:
'''
iSort=ImageSort()
iSort.runDefault()
'''
#test script
#testImagePath1='./unsorted/IMG_5662.JPG'
#testImagePath2='./unsorted/IMG_5678.JPG'
iSort=ImageSort()
start=time.time()
fileNames=iSort.getFileNames()
print('Gathering histogram data...')
histoAll=iSort.getHisto()
print('Histogram Extraction Complete')
print('Clustering...')
clusterAssign=iSort.findKeyPhotos(histoAll)
print('K-Means Clustering Complete')
print('Begin text detection and OCR...')
imgText=iSort.textDetectAndRecogAll(clusterAssign)
print('Sorting Complete')
iSort.makeFolders(imgText)
imgToFolder=iSort.folderMap(imgText)
iSort.sortImages(imgToFolder,fileNames)
end=time.time()
execTime=end-start
print('OCR Image Sort Complete')
print('Execution time (s): ' + str(end-start))

#Histogram stuff
#histTest=iSort.getHisto()
#clustersAssign=iSort.findKeyPhotos(histTest)
#iSort.runTextRecogOnly()
#output=iSort.runTextDetectAndRecog()
#iSort.unsortImages(imgToFolder,fileNames)
'''
imText=iSort.readImage(testImagePath2)
print(imText)
'''
```