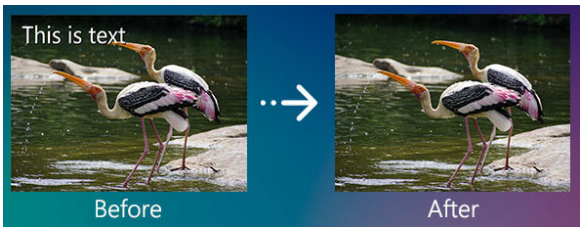
ROBT310 Final Project: Removal of the text in image

Zakhar Semenov, Alimzhan Mukushev, Aidar Amangeldi
Nazarbayev University

***Abstract*— In this project, we plan to develop a program that detects the text in the image and removes it with consideration of the background. In other words, the picture must retain its content but without the text. The motivation behind the project is to remove informational noise from the image or to hide unwanted text for censorship reasons which can be interesting for certain authorities. The approach will include several steps: detail enhancement in the image, text recognition, text removal, and background restoration.**

***Index Terms*— Text detection, digital image processing, pixel replacement**



I. Introduction to the Topic

Computer Vision has a wide range of use cases in the modern world. It provides many life improvement features to day-to-day life of people. This improvements include: simultaneous text translations, text-to-speech conversion for people with eyesight problems, augmented reality methods to enhance human-computer interaction and improve productivity.

II. Literature Review

As it has been mentioned earlier text recognition from images is active research in the field of Computer Vision. To address the issues related to text recognition many researchers have proposed different approaches. In the forthcoming section, we present an overview of approaches proposed to handle the issues related to text recognition. The first paper that we came across is published in 2015 by Pratik Madhukar Manwatkar and Dr. Kavita R. Singh from Yeshwantrao Chavan College of Engineering, Nagpur [1]. It is called A Technical Review on Text Recognition from Images. While the paper does not present a specific algorithm to solve the problem, it was a good start to dive into the topic of Text Recognition as it presents more than 20 research articles. In this way, now our team can navigate through the topic since the paper provides some of the useful datasets, open-source algorithms and software that are helpful in developing the project. Moreover, this paper devotes a separate section for the practical applications of text recognition.

Another research made by Yang et al. [2] is called A fast adaptive binarization method for complex scene images. The work includes methods (e.g Wavelet decomposition, Low-pass filter, Wavelet reconstruction, etc.) that are used to manipulate with image so that the in resulting image it is easier to read the text for OCR. The paper presents a novel adaptive binarization method based on wavelet filter, which shows comparable performance to other similar methods and processes faster, so that it is more suitable for real-time processing and applicable for mobile devices. The proposed method is evaluated on complex scene images of ICDAR 2005 Robust Reading Competition.

Pradeep et al. [3] have proposed neural network based classification of handwritten character recognition system. Each individual character is resized to 30 by 20 pixels for processing. They are using binary features to train neural network. However such features are not robust. In post processing stage, recognized characters are converted to ASCII format. Input layer has 600 neurons equal to number of pixels. Output layer has 26 neurons as English has 26 alphabets. Proposed ANN uses back propagation algorithm with momentum and adaptive learning rate.

We believe that Neural Networks would be particularly helpful to solve our issue, so we reviewed yet another CNN-based approach. In paper [4], multiple neural network models are used to recognize the Kannada language characters and the period during which the script was written. This involves two phases. The first phase of the work incorporates an Artificial Neural Network for identifying the base character. The second phase consists of a Probabilistic Neural Network model designed for the identification of age pertaining to the base character. Characters dated from century BC to the present day are used for analysis and experimentation results.

Another research conducted by Wagh and Patil [5] investigated the removal of the text from the image. Their method was divided in three procedures: text localization, text extraction, and text removal using inpainting. Inpainting is an image restoration technique used to restore the damaged parts of an image.

To simplify the process of text extraction, Jain and Yu [6] proposed an Automatic Text Location method that extracts the regions of the image containing text. These regions can then be directed to the Optical Character Recognition (OCR) model for further processing. Such approach is better in computational cost and accuracy.

In [7] they made a thorough overview to the topic of text detection in the image bringing in state-of-art papers classifying them by methods they use. The work presents step-wise

approaches to the problem, useful datasets and challenges. This paper is useful for us in the way it gives us an idea about how the text can be detected and localized as it happens to be the most important art of our project. Moreover, the papers presents problems that may appear with the text recognition, namely: scene complexity, uneven lightning, blurring and degradation, aspect ratios, distortion, fonts and multilingual environment.

Ohya et. al [8] has proposed a method of adaptive thresholding and relaxation operations using the text features such as Color, scene text style for text localization and recognition. Character recognition takes place using: high similarity between character pattern candidates and standard patterns taken from dictionary. They conducted experiments on 100 images which involves characters of different size and formats under uncontrolled light. The method developed can be used to read variously formatted characters with unknown size, font, and gray-level under uncontrolled lighting.

Scene-Text-Eraser [9] made the first attempt to erase text from real scenes using a deep neural network. To solve the data-deficiency problem of scene text removal, that is, the fact that non-text versions of real data do not exist, Scene-Text-Eraser generated 229 pseudo-background images as the ground truth from the ICDAR13 dataset by masking the original images with the given stroke annotations and inpainting with the third-party algorithm. Then, Scene-Text-Eraser used pairwise input images and pseudo-background images to train a deep neural network, which produced text-removed images. However, the scene text removal performance heavily relied on the heuristics of the pseudo-background generation. Although Scene-Text-Eraser dilated the given text stroke mask to cover ambiguous text boundaries in the pseudo-background generation, it still could not deal with more complex wild text.

III. Feasibility Study

This project can be divided into several parts:

1) Image restoration and detail enhancement
2) Text recognition
3) Text removal
4) Background restoration

The first part requires us to apply certain image processing algorithms to enhance the details and rid of sugar-pepper noise. The details can be enhanced using histogram equalization algorithms, while the sugar-pepper noise can be avoided by implementing Gaussian filter. The second part, as it was said, requires us to use an external Google Cloud Vision API to detect and extract text from the image. The next step can be accomplished using the information provided by the Google Cloud Vision API from the previous step. The API outputs the regions of the image that contain text, printed or handwritte. Since the last step - background restoration - does not have an available open-source library, we have to implement one of the approaches described above. Due to the big amount of different algorithms for this purpose, we need to conduct a performance evaluation of each of those algorithms to create the optimal approach.

We are planning to use the Google Cloud Vision API library for OCR in this project. This will require to create a free account with trial to gain access to the API. The API supports over 50 languages for recognition and many more languages as experimental features. Additionally, the API can detect several different handwritten scripts, including Latin, Japanese, and Korean.

IV. Project Implementation

A. Image preprocessing

Noise reduction. So before applying text recognition algorithms, the image has to be pre-processed from different types of noises and disturbances and adjust the contrast. First, the removal of the noise, in particular salt-and-pepper noise, is an important step as it facilitates and increases the accuracy of the text detection algorithm. Salt and pepper noise refers to a wide variety of processes that result in the same basic image degradation: only a few pixels are noisy, but they are very noisy. The effect is similar to sprinkling white and black dots—salt and pepper—on the image [10].To do so, there exist two methods - median filter and morphological filter. Since the scope of our course covers the first, we decided to implement median filter.

Listing 1: Median filter function

```
import numpy
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.cm as cm

def median_filter(img, filter_size):
  temp = []
  indexer = filter_size // 2
  img_final = []
  img_final = numpy.zeros((len(img),...
  len(img[0])))
   for i in range(len(img)):
     for j in range(len(img[0])):
       for z in range(filter_size):
         if i + z - indexer < 0 or ...
         i + z - indexer > len(img) - 1:
           for c in range(filter_size):
             temp.append(0)
         else:
           if j + z - indexer < 0 or ...
           j + indexer > len(img[0]) - 1:
             temp.append(0)
           else:
             for k in range(filter_size):
               temp.append(img[i + z...
               - indexer][j + k - indexer])
       temp.sort()
       img_final[i][j] = temp[len(temp) // 2]
       temp = []
  return img_final
```

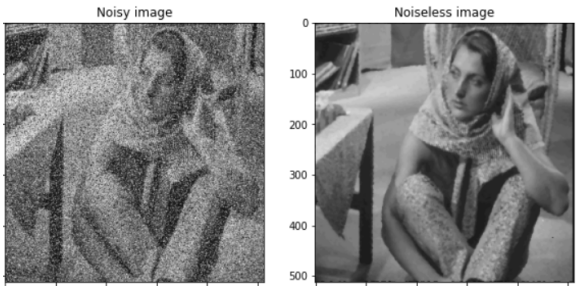


Fig. 1: Results of applying median filter

As you can see from the figure above, the median filter restores large amount of the image that is severely corrupted by defective pixels.

Histogram equalization.

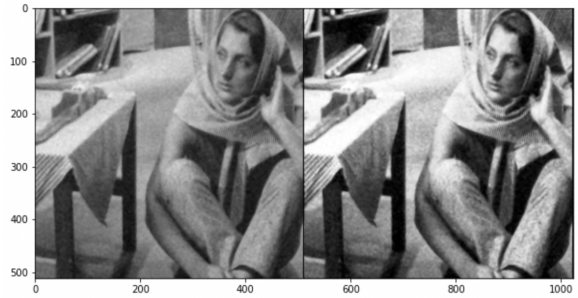


Fig. 2: Image before and after equalization

To make the recognition process more effective we next perform histogram equalization. To balance the image, we use OpenCV library. To visualize the histogram, we use numpy.hist function and calculate the CDF to test.

Listing 2: Histogram equalization function

```
import cv2
new_img = new_img.convert('L')
new_img_array = numpy.asarray(new_img)
hist_before, bin_before =
numpy.histogram(new_img_array.flatten(),
256, [0,256])
cdf_before = hist_before.cumsum()
cdf_norm_before = cdf_before *
float(hist_before.max()) / cdf_before.max()

plt.plot(cdf_norm_before, color = 'b')
plt.hist(new_img_array.flatten(), 256,
[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram before'),
loc = 'upper left')
plt.show()

equ = cv2.equalizeHist(new_img_array)
hist_after, bin_after =
numpy.histogram(equ.flatten(),
256, [0,256])
cdf_after = hist_after.cumsum()
cdf_norm_after = cdf_after *
float(hist_after.max()) / cdf_after.max()
plt.plot(cdf_norm_after, color = 'b')
plt.hist(equ.flatten(), 256,
[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram after'),
loc = 'upper left')
plt.show()

res = numpy.hstack((new_img,equ))

fig = plt.figure(figsize=(10,10))
imgplot = plt.imshow(res, cmap=cm.gray)
ax.set_title('After equalization')
```

The following tables correspond for histograms before and after equalization. The CDF from skewed became linear, which means an image is balanced.

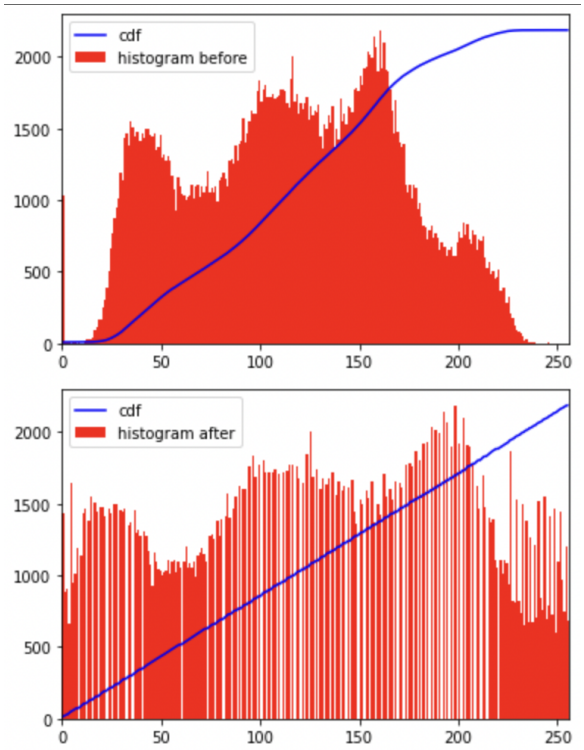


Fig. 3: Image before and after equalization

B. Text recognition

Fig. 4: Sample image for text recognition

As was mentioned previously, text recognition is handled by an external Google Cloud Vision API. Using the following code we were able to obtain the boundaries of the zones containing text in an image:

Listing 3: Text recognition

```
import os, io
from google.cloud import vision

os.environ[
  'GOOGLE_APPLICATION_CREDENTIALS'
  ] = r'token.json'

def text_rec(path):
  client = vision.ImageAnnotatorClient()

  with io.open(path, 'rb') as image_file:
    content = image_file.read()

  image = vision.Image(content=content)

  response = client.text_detection(
    image=image
  )
  texts = response.text_annotations
  print('Texts:')

  for text in texts:
    print('\n"{}"'.format(
      text.description
    ))

    vertices = (
      ['({},{})'.format(vertex.x, vertex.y)
      for vertex
      in text.bounding_poly.vertices]
    )

    print(
      'bounds:{}'.format(
        ','.join(vertices)
      )
    )

  if response.error.message:
    raise Exception(
      '{}'.format(
        response.error.message)
      )
```

By passing the image from Figure 4 to the *text_rec* function the program produces the following result:

Listing 4: Program output

```
Texts:

"white text"
bounds:(71,20),(519,20),(519,76),(71,76)

"white"
bounds:(71,20),(314,20),(314,76),(71,76)

"text"
bounds:(340,20),(519,20),(519,76),(340,76)
```

This output provides words with corresponding rectangular bounds from the image.

C. Text removal

Now that we have the bounding boxes for the text, we should come up with an algorithm that replaces those pixels in the bounding box so that it to produces a non-text restored background image. To do that, we have looked up to several methods form the related work and decided to resort to the one that is presented in [5] by Wagh and Patil. The method was encompasses three procedures: text localization, text extraction, and text removal using inpainting. Since we have already dealt with text localisation and text extraction in the Part B. We now focus on the last procedure - inpainting. In this stage, the bounding boxes are filled with appropriate colors present using nearest-matching neighborhood inpainting. To generate visually plausible region filling results, smoothing is carried out on the selected filled patches. According to the paper, "border of damaged region is searched and the pixel with smallest number of damage pixels in its 8-neighbors is used, with window centered on that selected pixel. In this step, The best matching pixel having more similarity to damaged one is searched using sum squared difference (SSD) criterion. Inpainting of selected pixel takes place from the color of best match pixel. But in case of high intensity change for order pixel is carried out using high intensity variation".

Listing 5: Text recognition

```
def text_rec(path):
# Get the client API instance
 client = vision.ImageAnnotatorClient()
# Input image
# Send text detection request to the API,
  get a response
 response = client.text_detection(image
          = image)
# Obtain text annotations
 texts = response.text_annotations
 print('Texts:')
# Cycle through text annotations, print
# the detected text and its boundaries
 for text in texts:
  print('"{}"'.format(text.description))

  vertices =(['({},{})'.format(vertex.x,
       vertex.y) for vertex in
       text.bounding_poly.vertices])

 print('bounds:{}'.format
      (','.join(vertices)))
```

D. Inpainting

In this stage, the bounding boxes are filled with appropriate colors present using nearest-matching neighborhood inpainting. Inpainting algorithm applies Otsu thresholding method. To generate visually plausible region filling results, smoothing is carried out on the selected filled patches.

Listing 6: Inpainting

```
def inpaint(img_hsv, vertices_y,
    vertices_x, inpainting_radius):
 start_y = abs(min(vertices_y))
 end_y = max(vertices_y)
 start_x = abs(min(vertices_x))
 end_x = max(vertices_x)

 part_hsv = img_hsv[
   start_y:end_y,
   start_x:end_x]

# Get Hue channel
 a_channel = part_hsv[:,:,1]

# Automate threshold
 using Otsu method
 th = cv2.threshold(a_channel, 127, 255,
   cv2.THRESH_OTSU)[1]

# Replace the part
 img_hsv[start_y:end_y, start_x:end_x] =
   cv2.inpaint(part_hsv, th,
   inpainting_radius, cv2.INPAINT_TELEA)
 return img_hsv
```

Listing 7: Smoothing

```
def smoothing(img_hsv, vertices_y,
    vertices_x, space, kernel_size):
 spaced_start_y = abs(min(vertices_y) -
   space)
 spaced_end_y = max(vertices_y) + space
 spaced_start_x = abs(min(vertices_x) -
   space)
 spaced_end_x = max(vertices_x) + space

 to_blur_part = img_hsv[spaced_start_y:
   spaced_end_y, spaced_start_x:
   spaced_end_x]
   img_hsv[spaced_start_y:spaced_end_y,
   spaced_start_x:spaced_end_x] =
   cv2.GaussianBlur(to_blur_part,
   (kernel_size,kernel_size), 0)
 return img_hsv
```

V. Results

Fig. 5: Before the text removal

Fig. 6: After the text removal

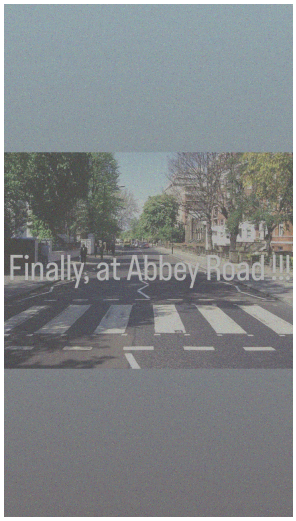


Fig. 7: Before the text removal

Fig. 8: After the text removal

Fig. 9: Before the text removal

Fig. 10: After the text removal

VI. Discussion and Conclusion

In this project we were able to achieve good results. The program successfully identifies and localized text, and finally removes the text. Although it is not perfect as some details around the text are lost, great portion of the content of images are preserved. It was noticed that, text removing algorithm performs better when given pictures with non-homogeneous background. We assume that it is because of the thresholding method.

References

[1] Manwatkar, Pratik & Singh, K. (2015). A Technical Review on Text Recognition from Images. 10.1109/ISCO.2015.7282362.

[2] Yang, Jufeng, Kai Wang, Jiaofeng Li, Jiao Jiao, and Jing Xu, “A fast adaptive binarization method for complex scene images,” 19th IEEE International Conference on Image Processing (ICIP), 2012.

[3] J. Pradeepa, E. Srinivasana, S. Himavathib, ”Neural Network Based Recognition System Integrating Feature Extraction and Classification for English Handwritten,” International journal of Engineering, Volume 25, May 2012.

[4] Kashyap, K., Bansilal, H. and Koushik, P. A., “Hybrid neural network architecture for age identification of ancient Kannada scripts”, Proceedings of the International Symposium on Circuits and System, Vol. 5, (2003), 25-28.

[5] P. D. Wagh and D. R. Patil, ”Text detection and removal from image using inpainting with smoothing,” 2015 International Conference on Pervasive Computing (ICPC), 2015, pp. 1-4, doi: 10.1109/PERVASIVE.2015.7087154.

[6] A. K. Jain and Bin Yu, ”Automatic text location in images and video frames,” Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170), 1998, pp. 1497-1499 vol.2, doi: 10.1109/ICPR.1998.711990.

[7] Ye, Qixiang & Doermann, David. (2015). Text Detection and Recognition in Imagery: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence. 37. 10.1109/TPAMI.2014.2366765.

[8] J.Ohya, A. Shio, and S. Akamatsu, “Recognizing Characters in Scene Images”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16-2, 1994,pp.214-224.

[9] T. Nakamura, A. Zhu, K. Yanai, and S. Uchida, ”Scene text eraser” in Proc. 14th IAPR Int. Conf. Document Anal. Recognit. (ICDAR), Nov. 2017, pp. 832–837.

[10] Alan C. Bovik. 2009. The Essential Guide to Image Processing. Academic Press, Inc., USA.