

OPTICAL CHARACTER RECOGNITION OCR

CREATING OPTICALLY RECOGNISABLE CHARACTERS

-presented by

B.STAT 1ST YEAR STUDENTS

AKANKSHA DAS(BS2307) SOHAN KARMAKAR(BS2349)
PRATYUSH PATRA(BS2336) RWITOBROTO DEY(BS2341)
SOUMYA CHANDA(BS2350)

A project work, presented for the Course:
STATISTICS METHODS-II , B.STAT 1st year, Semester-2



Department Name:BACHELOR OF STATISTICS
University Name:INDIAN STATISTICAL INSTITUTE
Country:INDIA
Date: May 12, 2024

Acknowledgement

We are grateful to our respected Professor Arnab Chakrabarty, who gave us the opportunity to work on such an interesting topic. He has also been instrumental in guiding us through this project successfully. With his wisdom and knowledge, we were able to complete this report with ease under his supervision which was a very enriching experience for all of us!

Contents

	Page
1 Problem Statement	4
2 Proposed Alphabets and New Alphabet Construction Methodology	5
2.1 Proposed Alphabets	5
2.2 New Alphabet Creation	11
3 Algorithm to Segregate the Letters	11
4 Recognition of Letters	12
4.1 Connected Component Labelling	12
4.2 Algorithm	13
5 Threshold Discussion	16
5.1 DBSCAN Thresholds	16
5.2 Letter Recognition Thresholds	16
6 Experimental Result	18
7 Conditions to obtain Optimal Results	20
8 Future Work	21
9 Conclusion	21

Abstract

The main purpose of this project is to make synthetic characters which are easily recognisable by machines. The characters created will have some special features that will make it easier for the machines to identify. Finally these characters will be mapped with the traditional English characters to satisfy the purpose of this project. Our method involves Connected-component labeling (CCL) (connected-component analysis (CCA), blob extraction, region labeling, blob discovery, or region extraction) which is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. The experiments carried out shows our algorithm manages to recognise these characters productively, although there are some errors under some conditions.

1 Problem Statement



















There are several algorithms involving neural networks and unsupervised learning to recognise the English alphabet which are rather complex as they were not created with the intention of being easily recognisable by a computer. Now in this digital era, the ability of a computer to recognise alphabets with ease and greater accuracy is a must. So, we have felt the need to design a new set of alphabets that are easy to write and can be recognised easily by a computer.


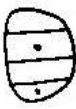

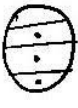



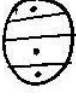





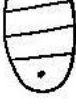

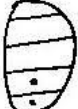
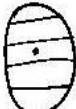

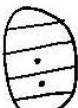


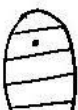
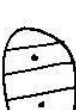

The problem consists of designing a class of alphabets with rules for extension that can be easily recognised by the computer and are easy to write. This new set of alphabets should be resilient to stray dots and lines and small pixel gaps. The time complexity to recognise the letters should be reasonable. Ideally, the algorithm should be able to do its job at minimum space constraints even when the text is produced in a moving vehicle.

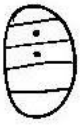


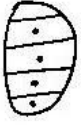
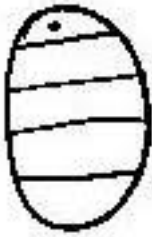
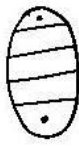
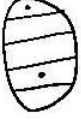


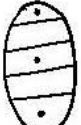

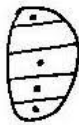
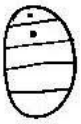


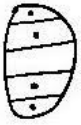




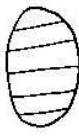
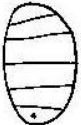


2 Proposed Alphabets and New Alphabet Construction Methodology

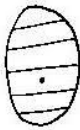
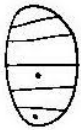
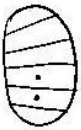


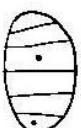
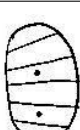

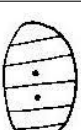
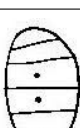
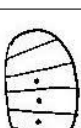
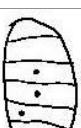
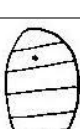
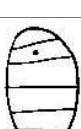
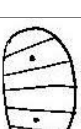
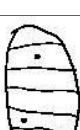
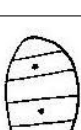
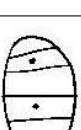
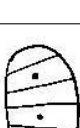
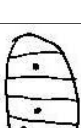
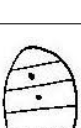


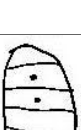
2.1 Proposed Alphabets

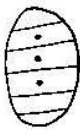
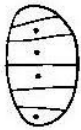
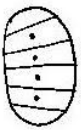

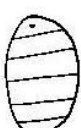
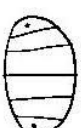


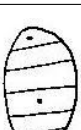
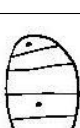

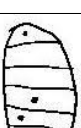
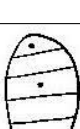
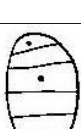
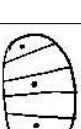
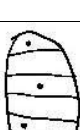
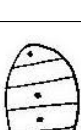
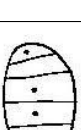

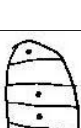
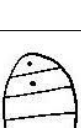



We found out that if we can make out characters with identifiers such as bounded regions and the presence or absence of blobs in a particular bounded region, then that can be easily recognized by our machine. So we can easily distinguish the characters and can map them to our desired alphabets, digits and special characters as per our own needs. For the sake of simplicity of writing, it is beneficial to assign frequently occurring characters to simpler symbols.

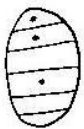
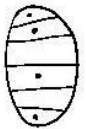
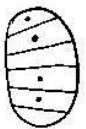
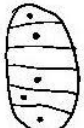
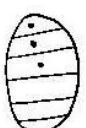
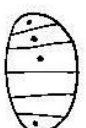
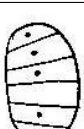
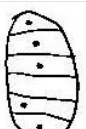
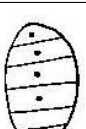
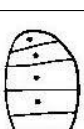
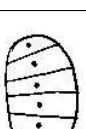
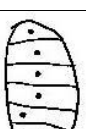
CHAR	ASCII	SYMB	CHAR	ASCII	SYMB	CHAR	ASCII	SYMB
	32	space		1			2	
	3			4			5	
	6			7			8	backspace
	9	tab		10	newline		11	
	12			13			14	
	15			16			17	

CHAR	ASCII	SYMB	CHAR	ASCII	SYMB	CHAR	ASCII	SYMB
	18			19			20	
	21			22			23	
	24			25			26	
	27			28			29	
	30			31			33	!
	34	"		35			36	
	37			38			39	
	40	(	41)		42	*

CHAR	ASCII	SYMB	CHAR	ASCII	SYMB	CHAR	ASCII	SYMB
	43			44	,		45	
	46	.		47			48	0
	49	1		50	2		51	3
	52	4		53	5		54	6
	55	7		56	8		57	9
	58	:		59	;		60	
 <u> </u>	61			62			63	?
	64	@		65	A		66	B

CHAR	ASCII	LETTERS	CHAR	ASCII	LETTERS	CHAR	ASCII	LETTERS
	67	C		68	D		69	E
	70	F		71	G		72	H
	73	I		74	J		75	K
	76	L		77	M		78	N
	79	O		80	P		81	Q
	82	R		83	S		84	T
	85	U		86	V		87	W
	88	X		89	Y		90	Z

CHAR	ASCII	LETTERS	CHAR	ASCII	LETTERS	CHAR	ASCII	LETTERS
	91	[	92			93]
	94			95			96	
	97	a		98	b		99	c
	100	d		101	e		102	f
	103	g		104	h		105	i
	106	j		107	k		108	l
	109	m		110	n		111	o
	112	p		113	q		114	r

CHAR	ASCII	LETTERS	CHAR	ASCII	LETTERS	CHAR	ASCII	LETTERS
	115	s		116	t		117	u
	118	v		119	w		120	x
	121	y		122	z		123	
	124			125			126	

2.2 New Alphabet Creation

We can keep on increasing the number of chambers to assign further new symbols if required. Each chamber gives rise to 2 different patterns. Hence, n chambers will give rise to 2^n different patterns. So, if we take maximum 6 chambers, we can get $\sum_{k=1}^6 2^k = 2^7 - 2 = 126$ different symbols.

Suppose, now we want to get 1000 different symbols, it is quite clear from our calculations that taking $n = 9$ will suffice as then, we will have

$$\sum_{k=1}^9 2^k = 2^{10} - 2 = 1022 \text{ many different symbols.}$$

3 Algorithm to Segregate the Letters

DBSCAN algorithm with epsilon = 7.5 and minimum no of points = 1 has been used to segregate the letters. First, the the image is converted into a binary one and the coordinates of the black points are stored and DBSCAN is applied on them to get each letter as a cluster.

Note : Each letter is a cluster along with various error clusters that are removed later.

DBSCAN Algorithm: DBSCAN categorizes points into three main categories: core points, border points, and noise points. Core points are those with a sufficient number of neighboring points within a specified radius, while border points lie within the neighborhood of a core point but lack enough neighbors to be considered core themselves. Points that do not fall into either category are labeled as noise points or outliers. The algorithm proceeds by iteratively exploring the dataset, forming clusters around core points and including border points within these clusters. Through this process, DBSCAN effectively delineates clusters while filtering out noise. DBSCAN offers a robust and efficient approach to cluster data where number of clusters is not known beforehand.

4 Recognition of Letters

We make extensive use of Connected Component Labelling (CCL) in our algorithm to identify letters.

4.1 Connected Component Labelling

Connected component labeling (CCL) is a fundamental technique in image processing and computer vision used to identify and label distinct objects or regions in a binary image. The goal is to partition the image into connected components, where pixels within the same component are connected through adjacency. The algorithm generally comprises of the following major steps:

Input : The input image is typically a binary image, where each pixel is either foreground (1 or white) or background (0 or black). After grayscaleing the image, any pixel with grayscale value less than 145 was made 0 and the remaining was made 1.

Initialization: Each pixel in the input binary image is assigned a unique label. This labeling process starts from 1 and proceeds sequentially. A data structure is initialized to store the labels assigned to each pixel.

Scanning the Image: The image is scanned pixel by pixel to identify connected components. The scanning process follows a predefined left-to-right and top-to-bottom order, to ensure consistency.

Pixel Processing: For each foreground pixel encountered during the scan, we do the following:

- If the pixel has no labeled neighbours (i.e., it's not adjacent to any other labeled pixel), a new label is assigned to it.
- If the pixel has labeled neighbours, it inherits the label of the neighbor with the minimum label value. This step ensures that all pixels within the same connected component share the same label.

Label Equivalence: As the algorithm progresses, it may encounter situations where different labels are assigned to pixels belonging to the

same connected component. To ensure consistency, label equivalence is enforced by maintaining a data structure (e.g., disjoint-set data structure or union-find algorithm) that tracks equivalent labels. Whenever two labels are found to be equivalent, they are merged into a single label.

Finalization: Once the entire image has been scanned and labeled, a final pass may be made to re-label the connected components using canonical labels. This step ensures that each connected component is uniquely labeled, starting from 1 and incrementing sequentially.

4.2 Algorithm

Now, we describe our algorithm that efficiently uses Connected Component Labelling so as to perform letter recognition.

The `Count Filled Circles Per Region` algorithm is designed to identify and quantify filled circles within different regions of an image. Initially, we process an image where various regions have been distinguished and assigned unique labels. These labels denote different objects or areas within the image. Subsequently, we examine each labeled region individually. We determine the size of each region by calculating its area, representing the extent of space it occupies in the image. If a region surpasses a certain threshold size (like 100 pixels), we proceed to further analysis. Within each region, we isolate connected components to search for the presence of blobs. For every blob that we find, we assess its relative proportion of the total region area. Any component which overcomes a certain threshold (say, 0.001), is considered to represent a filled circle. We aggregate the counts of filled circles across all regions and return this information.

The `Create Tuples from Images` algorithm is a process to extract relevant information from images. Initially, we prepare an empty list (*inf*) to store this information. Then, using connected component labelling, we identify distinct regions or objects within the image. For each of these regions, we determine the count of filled circles present. If a region contains at least one filled circle, we add this count to our list of in-

Algorithm 1 Count Filled Circles Per Region

```
1: procedure COUNT_FILLED_CIRCLES(labeled_image, num_labels)
2:   Initialize counts_per_region as an empty dictionary
3:   for label in range(2, num_labels) do
4:     filled_circle_count  $\leftarrow$  -2
5:     mask  $\leftarrow$  Generate binary mask for region corresponding to label in labeled_image
6:     area1  $\leftarrow$  Calculate area of the region
7:     if area1 > 100 then
8:       mask  $\leftarrow$  cv2.threshold(mask, 145, 255, cv2.THRESH_BINARY_INV)[1]
9:       numlbl, lbl  $\leftarrow$  cv2.connectedComponents(mask)
10:      for lbls in range(0, numlbl) do
11:        mask1  $\leftarrow$  Generate binary mask for connected component lbls
12:        area  $\leftarrow$  Calculate area of connected component lbls
13:        per_area  $\leftarrow$  Calculate percentage area occupied by connected component lbls
14:        if per_area  $\geq$  0.001 then
15:          filled_circle_count  $\leftarrow$  filled_circle_count + 1
16:        end if
17:      end for
18:      counts_per_region[label]  $\leftarrow$  filled_circle_count
19:    end if
20:  end for
21:  return counts_per_region
22: end procedure
```

formation. Finally, we return this list, providing a summary of the filled circle counts across different regions of the image.

Algorithm 2 Create Tuples from Images

```
1: procedure TUPLE_CREATE(img)
2:   Initialize inf as an empty list
3:   num_labels, labels  $\leftarrow$  cv2.connectedComponents(img)
4:   filled_circle_counts  $\leftarrow$  Count filled circles for each region in labels
5:   for each region and count in filled_circle_counts do
6:     if count  $\geq$  0 then
7:       Append count to inf
8:     end if
9:   end for
10:  return inf
11: end procedure
```

The `to_1letter` function takes a list of numbers as input and converts it into a corresponding ASCII character. If the input list is [0], it returns a space character. For other lists, it interprets them as binary numbers, calculates their decimal equivalent, and converts it to the corresponding ASCII character. If the resulting ASCII value is less than or equal to 32 (non-printable range), it adjusts it accordingly before returning the character.

Algorithm 3 Convert to Letter

```
1: function TO_LETTER(ls)
2:   if ls = [0] then
3:     return chr(32)
4:   end if
5:    $n \leftarrow \text{length of } ls$ 
6:    $no \leftarrow 2^n - 1$ 
7:    $temp \leftarrow n - 1$ 
8:   for  $i$  in range(length of ls) do
9:      $no \leftarrow no + ls[i] \times 2^{temp}$ 
10:     $temp \leftarrow temp - 1$ 
11:   end for
12:   if  $no \leq 32$  then
13:      $no \leftarrow no - 1$ 
14:   end if
15:   return chr(no)
16: end function
```

The `process_images` function takes an image path as input, opens the image, and converts it into a binary format using a threshold value of 145. It then saves the binarized image and opens it again. Next, it performs clustering on the binarized image, extracting distinct regions. For each region, it converts it to grayscale, applies thresholding again, and generates a list of tuples representing the filled circles within the region. These tuples are converted into letters using the `to_letter` algorithm. The resulting letters are concatenated into a string and returned.

Algorithm 4 Process Images

```
1: procedure PROCESS_IMAGES(st)
2:    $temp \leftarrow$  Open image at path st
3:    $temp \leftarrow$  Binarize temp using threshold value 145
4:   Save temp as "Binarised.jpg"
5:    $frame1 \leftarrow$  Open "Binarised.jpg"
6:    $climg \leftarrow$  Perform clustering on frame1
7:   Initialize empty dictionary infdict
8:    $str \leftarrow ""$ 
9:   for  $i$  in range(length of climg) do
10:    Convert climg[ $i$ ] to grayscale
11:    Threshold climg[ $i$ ] with threshold value 145
12:     $lst \leftarrow \text{TUPLECREATE}(climg[i])$ 
13:     $infdict[i] \leftarrow lst$ 
14:    if length of lst is not 0 then
15:       $s \leftarrow \text{TO\_LETTER}(infdict[i])$ 
16:       $str \leftarrow str + s$ 
17:    end if
18:  end for
19:  return str
20: end procedure
```

5 Threshold Discussion

As this is a real life scenario, the thresholds vary a lot. So, we have cleverly substituted values in some cases to the best of our knowledge and in other cases, we have used experimental results and tried to minimise the error. Both the above scenarios have resulted in some limitations and ways to avoid them that we will discuss later.

Our algorithm for OCR requires us to convert the original image into a binarised image of only 0 (or black) and 1 (or white). To do so, we need a threshold value such that anything below the threshold is made 0 and anything above it is changed to 1. Now, we have experimented with 8 data sets and have found that 145 serves as a good value for this threshold and the binarised image so produced is quite favourable to us.

5.1 DBSCAN Thresholds

DBSCAN has two parameters - minimum number of points and epsilon.

As it is difficult to handle two parameters, we fix one of them (*minimum number of points* = 1). Now, we have to set epsilon such that in a particular symbol, every pixel has minimum 1 pixel as its epsilon neighbour and no two pixels belonging to two different characters have less than epsilon distance between them.

We have observed that an epsilon value of 7.5 works optimally for the data sets that we have obtained.

5.2 Letter Recognition Thresholds

In order to recognise the characters, we use two area thresholds. the first area threshold ensures that any connected component (or mask) has a minimum area before being recognised as a letter. This ensures that any careless gaps in the blobs made by the user does not affect the outcome. After experimentation with over 100 characters, we decided

to use 100 square pixels as a threshold in this case.

We use our second area threshold to give a lower bound on the area of the blobs drawn by the user. To negate the effects of the dimensions of the picture, we use the proportional area between those of the blobs and the entire label. We have observed that the value 0.001 serves as a good threshold in most of the cases and hence agreed to move on with it.

6 Experimental Result

DATASET	Letters Present	Letters Recovered	Letters Correctly Recognized
1	30	30	30
2	32	32	32
3	32	32	32
4	32	32	31
5	5	5	5
6	40	40	39
7	1	1	1
8	9	9	9
9	20	20	20

Error

Type-1 error:

We define **Type-1 Error** as the erroneous results produced during execution as a result of the non-recovery of characters from the image due to the failure of the clustering algorithm.

We give a mathematical formula for the **Type-1 Error** produced by our code.

$$\text{Type-1 Error} = \frac{\text{number of letters not recovered}}{\text{total number of letters}} \times 100$$

As observed from the above table where we experimented on 8 datasets, we get our **Type-1 Error** as $\frac{0}{201} \times 100 = 0\%$

Type-2 error:

We define **Type-2 Error** as the erroneous results produced during execution as a result of incorrect recognition of characters from the image due to the failure of the letter recognition algorithm.

We give a mathematical formula for the **Type-2 Error** produced by our code.

$$\text{Type-2 Error} = \frac{\text{number of letters incorrectly recognised}}{\text{total number of letters correctly recovered}} \times 100$$

As observed from the above table,

$$\text{Type-2 Error} = \frac{2}{201} \times 100 = 0.995\%(\text{approx.})$$

Reason behind Type-1 Errors

Here are some causes which may lead to **Type-1 Errors**.

1. If the user is used to writing the letters quite close to each other and in the process does not abide by the 7.5 pixels gap requirement, the clustering algorithm may fail and cause the two closely spaced characters to be recognised as one single character which leads to errors.

Reason behind Type-2 Errors

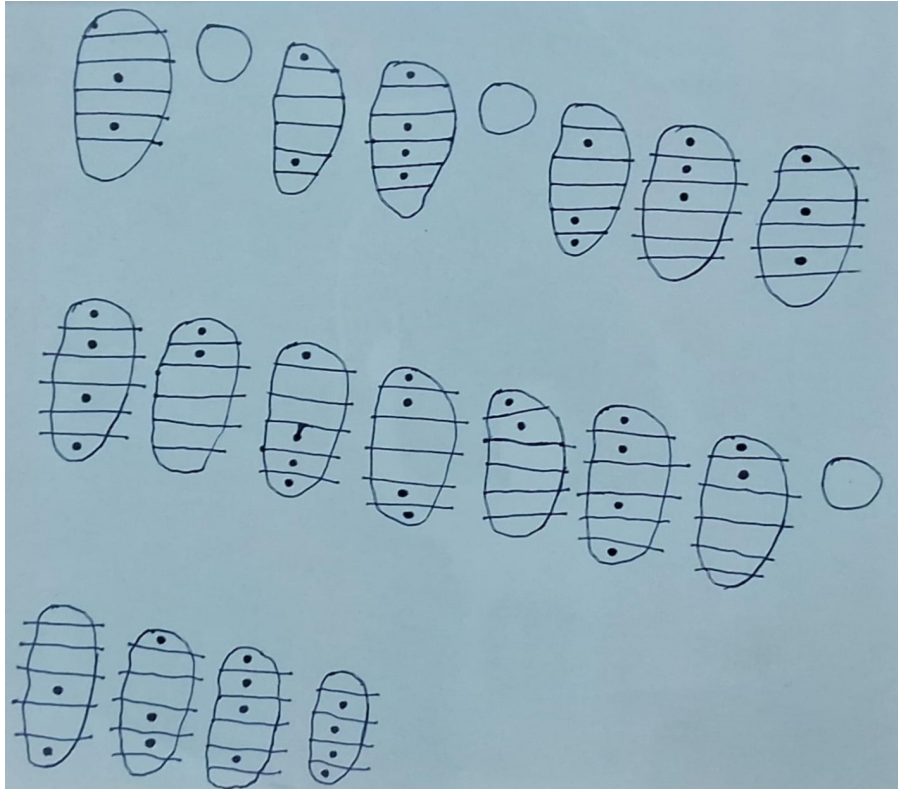
Here are some causes which may lead to **Type-2 Errors**.

1. Even if there is a single pixel that produces a gap on a line which is supposed to be separating a region into two, the connected component labelling algorithm will recognise the two different regions as a single label and produce undesirable results.
2. If the blobs so drawn by the user happen to be small enough that it does not satisfy the area requirement, it will be recognised as some different letter.
3. If the blobs get connected to the adjacent boundaries, then our algorithm will consider it as a boundary point and will no longer recognise it as a blob, causing erroneous results.

Though the second and third points can be overcome by the carefulness of the user. The first reason is a bit tricky to handle. In fact, both the **Type-2 Errors** in our test datasets were caused due to this reason.

Output of Some Datasets

Result 1 (Dataset 9):



Output : I am Rwitobroto Dey.

In this case, both **Type-1** and **Type-2** errors are 0.

7 Conditions to obtain Optimal Results

In order to obtain the best results from this code, it is better if the following conditions are satisfied.

1. The lighting of the image is uniform all over. Specific dark places may lead to disruptions as it may cause problems during binary conversion if the darkness causes its grayscale value to go below 145.
2. The DBSCAN algorithm operates left to right and hence in order to get the alphabets sequentially, the user is encouraged to write a bit

downhill ensuring the highest point of any letter to the left is above the highest point of any letter to the right.

3. It is also advised that the user gives space in writing and leaves at least two lines worth of space after each row. The user is also recommended to fill out any large unfilled region if it is unwanted.
4. To erase mistakes, the user can use the backspace character duly assigned or he can also distort the letter such that it coincides with no other letter. If a blob has been wrongly placed in any unwanted region, the user can delete the blob by connecting it to the boundary regions.

8 Future Work

Our algorithm definitely has some limitations, and here are some areas which can be significantly improved.

1. The synthetic characters which we developed are very time consuming to draw by hand. Moreover these characters look a lot like each other so any slight error while drawing these characters, like misplacing even a single dot, can make the results differ. So a simple but more distinct and easy-to-draw set of new characters would be appreciated.
2. The Connected component labeling method although efficient is very sensitive to quality of the input of image. If there is a single pixel that produces a gap on a line which is supposed to be separating a region into two, it will recognise the 2 different regions as a single label and produce undesirable results.
3. The clustering algorithm consumes a lot of time and memory. So it will be a significant upgrade if the algorithm can be made more time and memory efficient.

9 Conclusion

In conclusion, our project has successfully tackled the challenge of creating alphabets that can be reliably recognized through Optical

Character Recognition (OCR) systems. Through constant efforts, iterative development, and rigorous testing, we have achieved significant milestones in the field of character recognition.

Our OCR system demonstrates commendable accuracy and performance in recognizing the custom-designed alphabets, surpassing expectations and laying a strong foundation for future advancements. By overcoming various challenges encountered during the development process, such as data acquisition, algorithm optimization, and model training, we have proven the feasibility and effectiveness of our approach.

Moreover, our project introduces innovative techniques and methodologies that contribute to the ongoing evolution of OCR technology. The insights gained from this endeavor pave the way for further exploration and refinement in the realm of character recognition, promising enhanced solutions for diverse applications and industries.

Looking ahead, there are promising avenues for future research and development. We envision continued improvements in scalability, robustness, and adaptability of our OCR system, as well as exploration into new applications and integration possibilities. Our work opens doors to a multitude of opportunities, from document digitization to assistive technologies, empowering individuals and organizations with efficient and reliable text recognition capabilities.

In summary, our project underscores the importance of accurate character recognition in the digital age and demonstrates our commitment to advancing the state-of-the-art in OCR technology. We are proud of our accomplishments and look forward to the continued impact and innovation that will stem from this endeavor.