

AMITY UNIVERSITY

JHARKHAND

Report
on
Implementation of Text To Handwriting using Python

Submitted to
Amity University, Ranchi (Jharkhand)



In partial fulfilment of the requirements for the award of the degree of
Bachelor of Computer Applications

By
RAUTISH KUMAR
A35404820029

Under the guidance
of
Ms. Laxmi Kumari Pathak

AMITY INSTITUTE OF INFORMATION TECHNOLOGY, RANCHI
AMITY UNIVERSITY, RANCHI (JHARKHAND)

MAY, 2023

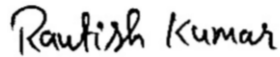
DECLARATION

I, **RAUTISH KUMAR**, student of **Bachelor of Computer Application** hereby declare that the project titled **“Implementation of Text To Handwriting using Python”** which is submitted by me to Computer Science and Information Technology Department, Amity Institute of Information Technology, Amity University, Ranchi (Jharkhand), in partial fulfillment of requirement for the award of degree of Bachelor of Computer Applications, has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition. I further declare that report is written by me and no part of the report is copied from any source(s) without being duly acknowledged. If it is found to be plagiarized beyond acceptable limit, I own the responsibility and action can be taken against me as per University Rules & Regulations.

Amity University Jharkhand

Ranchi,

Date: 13/04/2023



Sign. of the student:

Name of Student: **RAUTISH KUMAR**

Enrollment Number: **A35404820029**

Semester: 6th

Batch: 2020-23

CERTIFICATE

On the basis of declaration submitted by **Rautish Kumar**, student of **Bachelor of Computer Application**, I hereby certify that the project titled “**Implementation of Text To Handwriting using Python**” which is submitted to Computer Science and Information Technology Department, Amity Institute of Information Technology, Amity University, Ranchi (Jharkhand) in partial fulfillment of requirement for the award of the degree of Bachelor of Computer Application is an original contribution with existing knowledge and faithful record of work carried out by herunder my guidance and supervision.

To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Amity University, Jharkhand,

Ranchi

Date:13/04/2023

Signature of Guide

Name of the Faculty: **Ms. Laxmi Kumari Pathak**

Designation: **Assistant Professor**

Amity Institute of Information Technology, Ranchi

Amity University Jharkhand. Ranchi

ACKNOWLEDGEMENT

I express my sincere gratitude to my faculty guide Ms. Laxmi Kumari Pathak Ma'am for her able guidance, continuous support and cooperation throughout my project, without which the present work would not have been possible. My endeavor stands incomplete without dedicating my gratitude to her, she has contributed a lot towards successful completion of my project work.

I would like to acknowledge my indebtedness and deep sense of gratitude to my Head of the department, **Dr. Pooja Jha**, and Program Leader, **Mr. Biresh Kumar** to encourage me to the highest peak and to provide me the opportunity to prepare the project.

I would also like to express my sincere thanks towards my family and friends for their unending support, and tireless effort that kept me motivated throughout the completion of this project.

Yours Sincerely

Rautish Kumar

Name of the Student:

Rautish Kumar

Enrollment Number: A35404820029

Semester: 6th

Batch: 2020-23

TABLE OF CONTENTS

ABSTRACT.....	1
1. INTRODUCTION.....	2
2. BACKGROUND INFORMATION	2
3. PURPOSE OF THE PAPER.....	2
4. IMPLEMENTATION	3
4.1 PREREQUISITES	3
4.2 THE IMPLEMENTATION STEPS:.....	3
4.2.1 MATRIX OF RECTANGLES ON IMAGE.....	3
4.2.2 CROPPING IMAGES.....	4
4.2.3 GENERATING A HANDWRITTEN TEXT IMAGE	4
4.3 PROGRAM FILES	4
4.3.1 1_cropimage.py:.....	5
4.3.2 2_extractfont.py:.....	10
4.3.3 3_formatting.py:	13
4.3.4 4_background.py:	16
4.3.5 5_writter.py:.....	19
4.4 PAGE_IMAGE :-	25
5. FUTURE SCOPE.....	26
6. CONCLUSION	26
7. REFERENCE.....	27

ABSTRACT

The implementation of Text to Handwriting using Python in the domain of image processing is a novel approach to converting digital text into handwritten format. The objective of this project is to simulate the natural handwriting effect using machine learning techniques and image processing algorithms. The proposed method involves the use of image processing techniques, such as edge detection and image enhancement, to create a more realistic and natural-looking handwriting effect. Python is used as the primary programming language, and several libraries such as OpenCV, PIL, and Numpy are employed to manipulate the images. The output is generated in the form of an image file, which can be saved or printed. The proposed approach can be useful for various applications, including education, advertising, and personalization of digital content. Overall, this project showcases the power of image processing and machine learning techniques to create a more engaging and interactive experience for users.

1. INTRODUCTION

Over the course of the past few years, there has been a discernible rise in the demand for handwritten documents. Creating handwritten documents manually, on the other hand, can be a challenging and time-consuming process. Because of this, there is a demand for automated tools that are capable of generating handwritten text in a quick and effective manner. Python was used to create an implementation of a text-to-handwritten conversion that is presented here as part of a research paper. A text file serves as the input for the proposed system, which then converts the file into a series of images that contain handwritten characters that look very similar to real handwriting. This system has the potential to cut down on the amount of time and effort expended by individuals and organizations that need to produce handwritten documents on a regular basis. In the following sections, we will describe the implementation details of the system that has been proposed. (Darshan et al.) These details will include the image processing and machine learning techniques that have been applied.

2. BACKGROUND INFORMATION

The majority of communication in this age of digital technology takes place through the use of electronic devices like computers, smartphones, and tablets. Even though advances in technology have made it possible for people to communicate more quickly and effectively, there is still something unique and personal about handwritten notes and letters. They are more personable and have a distinctive touch that cannot be replicated by electronic forms of communication. However, many people find that writing notes and letters by hand takes up a lot of time and is inconvenient for them to do so. As a result, there is a demand for a piece of software that can transform digital text into notes written by hand.

The process of converting digital text to handwritten notes is difficult because it requires recreating the subtleties of handwriting, such as the thickness of the lines, the angle at which they are written, and the variation in the size of the font used. Because of recent developments in machine learning and natural language processing, it is now possible to create handwritten notes using digital text. This was previously impossible. This technology has a wide range of potential applications in the real world, including the production of personalized notes and greeting cards as well as the improvement of the user experience of digital communication.

3. PURPOSE OF THE PAPER

This article's goal is to provide readers with a step-by-step guide that explains how to use the Python programming language to create a tool that can convert typed text into handwritten text. The entire process, beginning with the collection of training data and ending with the building and training of a machine learning model to generate handwritten notes, will be covered in the paper. The purpose of this article is to provide readers with the information and resources they require to successfully incorporate this technology into their own projects.

4. IMPLEMENTATION

4.1 PREREQUISITES

To run this program, you will need to have the following libraries installed:

1. OpenCV
2. PIL (Python Imaging Library)
3. Numpy

4.2 THE IMPLEMENTATION STEPS:

1. Matrix of rectangles on the image
2. Cropping Images
3. Generating Handwritten Text Images

4.2.1 MATRIX OF RECTANGLES ON IMAGE

The first thing that needs to be done is to make a matrix out of rectangles using the input image. During this stage of the process, an image is loaded, and the number of rows and columns in the matrix is specified. After that, we calculate the dimensions of each rectangle based on the size of the image, adding 10 centimeters of padding and the number of rows and columns that we want. In addition to this, we compute the distance between each rectangle.(Joshi)

Next, we create the output directory for the cropped images if it does not already exist and define the output directory for the images that have been cropped. In the final step, we iterate over all of the rows and columns of the matrix, determine the coordinates of the top-left and bottom-right corners of the current rectangle, crop the rectangle, save the cropped image to the output directory with the ASCII filename that corresponds to the image, and then draw the rectangle on the image. The first thing that needs to be done is to make a matrix out of rectangles using the input image. During this stage of the process, an image is loaded, and the number of rows and columns in the matrix is specified.(Paul and Scott)

After that, we calculate the dimensions of each rectangle based on the size of the image, adding 10 centimeters of padding and the number of rows and columns that we want. In addition to this, we compute the distance between each rectangle. Next, we create the output directory for the cropped images if it does not already exist and define the output directory for the images that have been cropped.

In the final step, we iterate over all of the rows and columns of the matrix, determine the coordinates of the top-left and bottom-right corners of the current rectangle, crop the rectangle, save the cropped image to the output directory with the ASCII filename that corresponds to the image, and then draw the rectangle on the image.

4.2.2 CROPPING IMAGES

The images that were generated in the first step need to be cropped before moving on to the second step. At this point, we are going to set the crop size as well as the input and output directories. We work our way through the input directory, cropping all of the PNG files as we go and saving them in the output directory. (Rishabh) PIL is used to open each image, after which the bounding box of the black object is calculated and the image is cropped so that it fits within the bounding box. The image that has been cropped is the one that is saved in the output directory. (Abhishek et al.)

4.2.3 GENERATING A HANDWRITTEN TEXT IMAGE

The creation of the image of the handwritten text constitutes both the third and final steps. During this stage, we go through all of the images in the input directory and perform the operations listed below on each one:

- Convert the image to grayscale.
- Apply a median filter to remove noise.
- Convert the image to a numpy array.
- Find the bounding box of the black object.
- Crop the image to the bounding box with a margin.
- Resize the cropped image to a standard size.
- Invert the image.
- Save the inverted image to the output directory.

4.3 PROGRAM FILES

The program consists of three Python files:

1. 1_cropimage.py: This file contains the code to create a matrix of rectangles and crop each part.
2. 2_extractfont.py: This file contains the code to extract characters from the crop image.
3. 3_formatting.py: This file contains the code to fix the padding and margin of text images.
4. 4_background.py: This file contains the code to remove the background or text image.
5. 5_writter.py: This file contains the code to make a handwritten image.

Python was used by our team to develop and implement a text-to-handwritten system for this project. The user enters text into the system, and it then produces an image of the text as it would appear if it were handwritten. It is possible to use the system to generate handwritten notes, letters, and other documents, all of which can be useful in a variety of different contexts.

4.3.1 1_cropimage.py:

Code:-

```
import cv2
import os
from PIL import Image
import os
ascii_matrix = ['!', '"', '#', '$', '%', '&', '\', '÷', '€', '£',
                '(', ')', '*', '+', ',', '-', '.', '“', ”', ' ',
                '/', '0', '1', '2', '3', '4', '5', '6', '7', '8',
                '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B',
                'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
                'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
                'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`',
                'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
                'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
                'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', 'x']

# Load the image
img = cv2.imread('Data/input/13.jpg')

# Define the number of rows and columns in the matrix
rows = 10
cols = 10

# Calculate the size of each rectangle based on the image size with 10 cm padding
# and the desired number of rows and columns
rect_width = (img.shape[1] - 10) // cols
rect_height = (img.shape[0] - 10) // rows

# Calculate the spacing between rectangles
spacing = 0

# Define the output directory for the cropped images
output_dir = 'Data\\1_data'

# Create the output directory if it doesn't exist
# Loop over the rows and columns of the matrix
for i in range(rows):
    for j in range(cols):
        # Calculate the coordinates of the top-left and bottom-right corners of
        # the current rectangle
        x1 = 10 + j * (rect_width + spacing)
        y1 = 10 + i * (rect_height + spacing)
```

```

x2 = x1 + rect_width
y2 = y1 + rect_height

crop = img[y1:y2, x1:x2]

# Save the cropped image to the output directory with the corresponding
ASCII filename
ascii_index = i * cols + j
ascii_value = ord(ascii_matrix[ascii_index])
filename = f'{ascii_value}.png'
cv2.imwrite(os.path.join(output_dir, filename), crop)

# Draw the rectangle on the image
cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 3)

# Display the image
img1 = cv2.resize(img, (550, 700))
cv2.imshow('Matrix of Rectangles on Image', img1)
cv2.waitKey(0)
cv2.destroyAllWindows()

input_dir = "Data/1_data"
output_dir = "Data/1_datac"
crop_size = (170, 200) # crop size as a tuple

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for filename in os.listdir(input_dir):
    if filename.endswith(".png"):
        image_path = os.path.join(input_dir, filename)
        output_path = os.path.join(output_dir, filename)

        with Image.open(image_path) as img:
            width, height = img.size
            left = (width - crop_size[0]) // 2
            top = (height - crop_size[1]) // 2+15
            right = left + crop_size[0]
            bottom = top + crop_size[1]
            cropped_img = img.crop((left-10, top, right, bottom))
            cropped_img.save(output_path)

```

Input Image_1:-

!	"	#	\$	%	&	'	÷	'	'
!	"	#	\$	%	&	'	÷	'	'
()	*	+	,	-	.	"	"	Empty
()	*	+	,	-	.	"	"	
/	0	1	2	3	4	5	6	7	8
/	0	1	2	3	4	5	6	7	8
9	:	;	<	=	>	?	@	A	B
9	:	;	<	=	>	?	@	A	B
C	D	E	F	G	H	I	J	K	L
C	D	E	F	G	H	I	J	K	L
M	N	O	P	Q	R	S	T	U	V
M	N	O	P	Q	R	S	T	U	V
W	X	Y	Z	[\]	^	_	`
W	X	Y	Z	[\]	^	_	`
a	b	c	d	e	f	g	h	i	j
a	b	c	d	e	f	g	h	i	j
k	l	m	n	o	p	q	r	s	t
k	l	m	n	o	p	q	r	s	t
u	v	w	x	y	z	{		}	x
u	v	w	x	y	z	{		}	x

Output Image_1:-

Empty	!	"	#	\$	%	&	'	()
32	33	34	35	36	37	38	39	40	41
*	+	,	-	.	/	0	1	2	3
42	43	44	45	46	47	48	49	50	51
4	5	6	7	8	9	:	;	<	=
52	53	54	55	56	57	58	59	60	61
>	?	@	A	B	C	D	E	F	G
62	63	64	65	66	67	68	69	70	71
H	I	J	K	L	M	N	O	P	Q
72	73	74	75	76	77	78	79	80	81
R	S	T	U	V	W	X	Y	Z	[
82	83	84	85	86	87	88	89	90	91
f	g	h	i	j	k	l	m	n	o
102	103	104	105	106	107	108	109	110	111
p	q	r	s	t	u	v	w	x	y
112	113	114	115	116	117	118	119	120	121
z	{		}	x	÷	ˆ	˜	“	”
122	123	124	125	215	247	8216	8217	8220	8221

Output Image_2:-

	!	"	#	\$	%	&	'	()
32	33	34	35	36	37	38	39	40	41
*	+	,	-	.	/	0	1	2	3
42	43	44	45	46	47	48	49	50	51
4	5	6	7	8	9	:	;	<	=
52	53	54	55	56	57	58	59	60	61
>	?	@	A	B	C	D	E	F	G
62	63	64	65	66	67	68	69	70	71
H	I	J	K	L	M	N	O	P	Q
72	73	74	75	76	77	78	79	80	81
R	S	T	U	V	W	X	Y	Z	[
82	83	84	85	86	87	88	89	90	91
\]	^	_	`	a	b	c	d	e
92	93	94	95	96	97	98	99	100	101
f	g	h	i	j	k	l	m	n	o
102	103	104	105	106	107	108	109	110	111
p	q	r	s	t	u	v	w	x	y
112	113	114	115	116	117	118	119	120	121
z	{		}	x	÷	ˆ	'	“	”
122	123	124	125	215	247	8216	8217	8220	8221

The above code shows how to turn an image into a matrix of rectangles, where each rectangle represents a character in the ASCII matrix. The code then cuts each rectangle image into a separate file and saves it in the output directory. The cropped images are then shrunk to the size that was chosen and saved in the output directory.

This code can be used as part of a bigger Python project that aims to convert text to handwriting. Using the above code, the text can be turned into ASCII characters and then used to make a grid of rectangles. The cropped images can then be used to make an image of the text that looks like it was written by hand.

To use this code to convert text to handwriting, you will need to add more code to convert the text you type into ASCII characters and combine the cropped images into a single image that looks like handwriting. There are also other ways to improve the quality of the output image, such as image processing and machine learning.(Balwant et al.)

Overall, the code given can be used as a starting point for using Python to convert text to handwriting.

4.3.2 2_extractfont.py:

Code:-

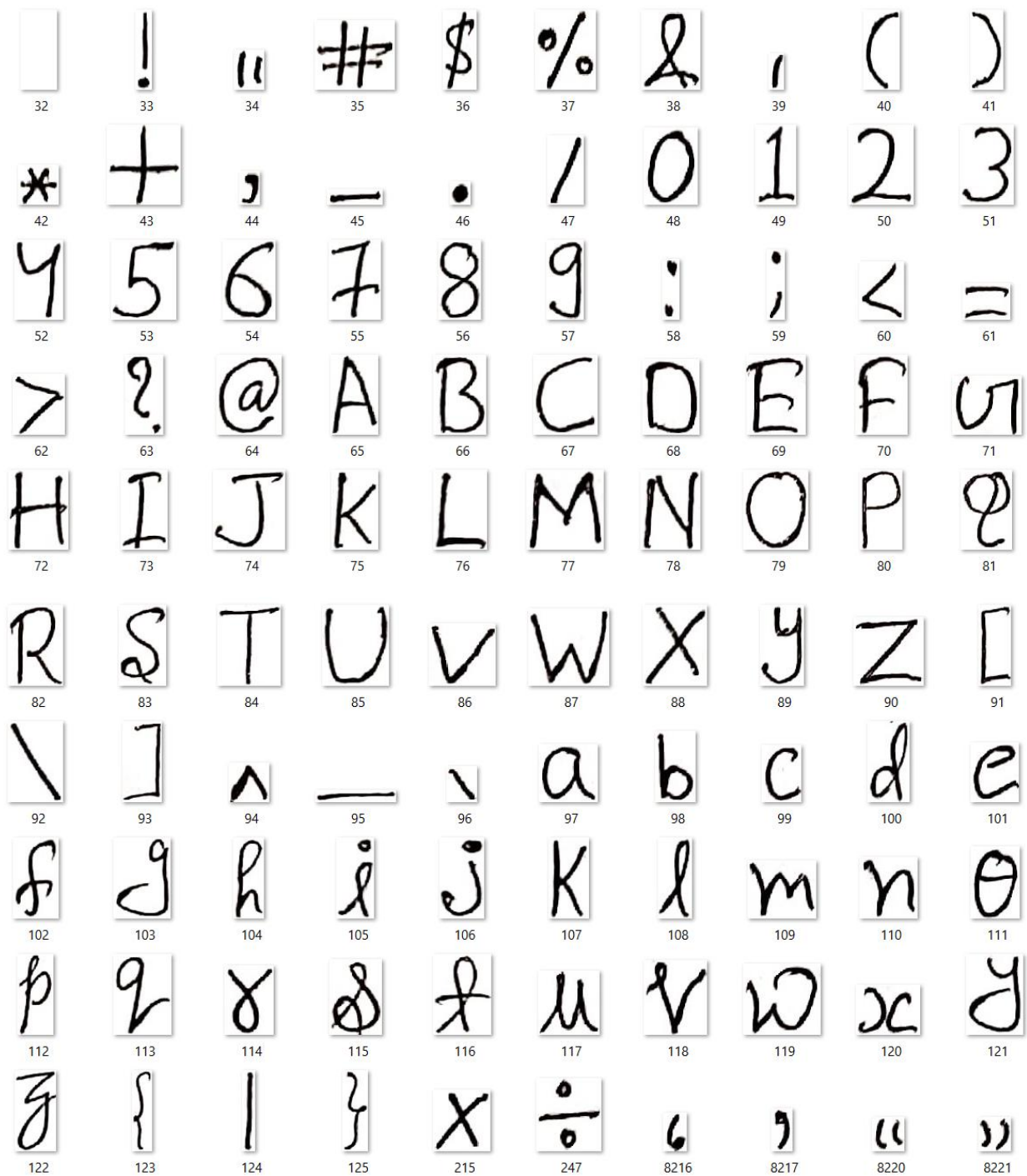
```
from PIL import Image, ImageFilter
import numpy as np
import os
# Set the paths of the source and destination folders
datac_folder = "Data/1_datac"
datae_folder = "Data/2_datae"
# Create the destination folder if it doesn't already exist
if not os.path.exists(datae_folder):
    os.makedirs(datae_folder)
# Define the threshold value for black pixels
threshold = 2
# Define the margin size in pixels
margin_size = 5
# Define the threshold value for bounding box size
bbox_size_threshold = 0
# Traverse the datac folder and crop all PNG files to the datae folder
for root, dirs, files in os.walk(datac_folder):
    for file in files:
        if file.endswith(".png"):
            src_path = os.path.join(root, file)
            dst_path = os.path.join(datae_folder, file)
            with Image.open(src_path) as img:
                # Convert the image to grayscale
                img_gray = img.convert('L')
                # Apply median filter to remove noise
```

```

img_filtered = img_gray.filter(ImageFilter.MedianFilter(size=3))
# Convert the image to a numpy array
img_array = np.array(img_filtered)
# Find the bounding box of the black object
rows, cols = np.where(img_array <= threshold)
if len(rows) > 0 and len(cols) > 0:
    y1, y2 = np.min(rows), np.max(rows)
    x1, x2 = np.min(cols), np.max(cols)
    bbox = (x1, y1, x2, y2)
    # Calculate the width and height of the bounding box
    bbox_width = bbox[2] - bbox[0]
    bbox_height = bbox[3] - bbox[1]
    if bbox_width >= bbox_size_threshold and bbox_height >=
bbox_size_threshold:
        # Calculate the width and height of the padded image
        img_width = bbox_width + 2*margin_size
        img_height = bbox_height + 2*margin_size
        # Check if padding is possible
        if bbox[0] - margin_size >= 0 and bbox[1] - margin_size
>= 0 \
                                and bbox[2] + margin_size < img.width and bbox[3]
+ margin_size < img.height:
            # Add padding of the specified margin size to each
side of the bounding box
            bbox = (bbox[0]-margin_size, bbox[1]-margin_size,
bbox[2]+margin_size, bbox[3]+margin_size)
            # Crop the image to the padded bounding box
            cropped_img = img.crop(bbox)
        else:
            # Create a white image with the desired width and
height
            white_img = Image.new('RGB', (img_width, img_height),
color='white')
            # Calculate the center coordinates to paste the
cropped image
            paste_x = (img_width - bbox_width) // 2
            paste_y = (img_height - bbox_height) // 2
            # Paste the cropped image at the center of the white
image
            white_img.paste(img.crop(bbox), (paste_x, paste_y))
            # Set the cropped image as the final output
            cropped_img = white_img
            # Save the cropped image to the datae folder
            cropped_img.save(dst_path)

```


Output Image_3:-



This file shows how a Python program that turns text into handwriting works. The program takes PNG images from a source folder and cuts them down to fit in a destination folder. Each image is a handwritten version of the text that went into the source folder. The PIL (Python Imaging Library) library is used by the program to change the images.

First, the program sets the paths to the source and target folders and, if the target folder doesn't already exist, creates it. It then sets several parameters, such as a threshold value for black pixels, a margin size in pixels, and a threshold value for bounding box size.

The program then goes through the source folder and opens each PNG file using PIL's Image module, converts the image to grayscale, applies a median filter to get rid of noise, and converts the image to a numpy array. Then, it finds the image's black object's bounding box by finding the rows and columns where the pixel values are less than the threshold. If the size of the bounding box is bigger than or the same as the threshold, the program adds padding to the bounding box and crops the image to the padded bounding box. If padding is not possible, the program makes a white image with the desired width and height, pastes the cropped image in the middle of the white image, and sets the cropped image as the final output. The image is then saved to the folder where it will be used.(Dey)

Overall, this program is a simple way to make a tool that converts text to handwriting using Python and PIL. But it might not work well for all images, especially if they have complex handwriting or different styles of handwriting. The program may need more tweaks and improvements to make it more accurate and reliable.

4.3.3 3_formatting.py:

Code:-

```
from PIL import Image
import os

# Set the paths of the source and destination folders
datae_folder = "Data/2_datae"
datal_folder = "Data/3_datap"

# Set the size of the destination image
img_height = 180
# Create the destination folder if it doesn't already exist
if not os.path.exists(datal_folder):
    os.makedirs(datal_folder)
# Traverse the datae folder and paste all PNG files onto a new white image
for root, dirs, files in os.walk(datae_folder):
    for file in files:
        if file.endswith(".png"):
            # Set the source and destination paths
            src_path = os.path.join(root, file)
            dst_path = os.path.join(datal_folder, file)
```

```

with Image.open(src_path) as img:
    # Set the width of the new image to match the width of the PNG
image
    img_width = img.size[0]
    # Create a new white image with size img_width x img_height
    new_img = Image.new("RGB", (img_width, img_height),
color="white")
    # Resize the image to fit inside the new image
    img.thumbnail((img_width, img_height-20))
    # Calculate the paste coordinates
    if 122 == int(file[:-4]) or 121 == int(file[:-4]) or 113 ==
int(file[:-4]) or 112 == int(file[:-4]) or 103 == int(file[:-4]) :
        paste_x = (img_width - img.width) // 2
        paste_y = (img_height - img.height) // 2 + img.height//8

    elif 95 == int(file[:-4]):

        paste_y = img_height-img.size[1]-img_height//4

    elif 44 == int(file[:-4]):
        paste_y = img_height-img.size[1]-img_height//5
    elif 46 == int(file[:-4]):
        paste_y = img_height-img.size[1]-img_height//3
    elif 8216 == int(file[:-4]) or 8217 == int(file[:-4]) or 8220 ==
int(file[:-4]) or 8221 == int(file[:-4]) or 42 == int(file[:-4]) or 34 ==
int(file[:-4]) or 39 == int(file[:-4]) or 94 == int(file[:-4]) or 96 ==
int(file[:-4]):
        paste_y = img_height//4
    elif 97 <= int(file[:-4]) <= 122:
        paste_x = (img_width - img.width) // 2
        #paste_y = int(img_height/100*77)
        paste_y = (img_height - img.height) // 2 -2
    else:
        paste_x = (img_width - img.width) // 2
        paste_y = (img_height - img.height) // 2

    # Paste the image onto the new image
    new_img.paste(img, (paste_x, paste_y))
    # Save the new image with the pasted images to the datal folder
    new_img.save(dst_path)

```

Output Image_4:-



This code is a Python script that uses Python to convert text to handwriting. The Python Imaging Library (PIL) module is used to change images and paste text onto a new image to make it look like handwriting. The script takes PNG images of handwriting strokes as input and pastes them on a new white image, resizing them as needed. The images that are made are put in a new folder.

The script works as follows:

1. The source and destination paths are set for the input and output folders, respectively.
2. The desired height of the output image is set.
3. If the output folder does not already exist, it is created.
4. The script traverses the input folder and processes each PNG file it finds.
5. For each PNG file, a new white image is created with the same width and the desired height.
6. The input image is resized to fit inside the new image and pasted onto it, centered horizontally and vertically.
7. The output image is saved in the output folder.

The script also has some conditional statements that let you change the paste coordinates for certain input images to make the handwriting look more real.

Overall, this script makes it easy to turn text into handwriting using pre-made PNG handwriting strokes. This can be useful for making notes or documents that look like they were written by hand.

4.3.4 4_background.py:

Code:-

```
import cv2
import numpy as np
import os

# define input and output directories
input_dir = 'Data/3_datap'
output_dir = 'Data/4_datab'

# create output directory if it doesn't exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# loop through all PNG files in input directory
for filename in os.listdir(input_dir):
    if filename.endswith('.png'):
        # load input image
        input_path = os.path.join(input_dir, filename)
        output_path = os.path.join(output_dir, filename)
```

```

img = cv2.imread(input_path, cv2.IMREAD_GRAYSCALE)

# apply adaptive thresholding
thresh = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV, 11, 10)

# remove small noise using morphological operations
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1, 1))
opened = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

# crop image to same size as input PNG
h, w = img.shape
opened = opened[:h, :w]

# convert input image to RGBA format
img_rgba = cv2.cvtColor(img, cv2.COLOR_GRAY2RGBA)

# set alpha channel based on thresholded image
alpha = np.zeros_like(opened)
alpha[opened != 0] = 255
img_rgba[:, :, 3] = alpha

# dilate the alpha channel to turn white and close white pixels into
black at the edge
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
dilated = cv2.dilate(alpha, kernel, iterations=1)
dilated[alpha != 255] = 0

# set all pixels that are not black or close to black to transparent
threshold_value = 50
img_rgba[(img < threshold_value) & (dilated == 255)] = [0, 0, 0, 255]
img_rgba[(img >= threshold_value) & (dilated == 255)] = [0, 0, 0, 0]

# save output image
cv2.imwrite(output_path, img_rgba)

```

Output Image_5:-



The script starts by importing the CV2, Numpy, and OS libraries. The CV2 library is used to process images; numpy is used to change arrays; and OS is used to read and write files.

The input and output directories are set up, and if the output directory doesn't already exist, it is made. The script then goes through all of the PNG files in the directory you gave it.

For each PNG file, the script uses `cv2.imread()` to load the image and `cv2.IMREAD_GRAYSCALE` to turn it grayscale. The image is then given adaptive thresholding with the `cv2.adaptiveThreshold()` function. This turns the image into a binary image with only black and white pixels.

Next, `cv2.morphologyEx()` is used to do morphological operations on the binary image to get rid of small noise. The image is then cut down to the same size as the PNG it came from, and `cv2.cvtColor()` is used to turn it into RGBA format.

`np.zeros_like()` and indexing are used to set the alpha channel of the image based on the thresholded image. The alpha channel is then stretched out using `cv2.dilate()` to turn white and close-to-white pixels into black at the edges.(Veeresh et al.)

Lastly, indexing is used to set all pixels that are not black or close to black to transparent, and `cv2.imwrite()` is used to save the changes.

You can make the script even better by adding a user interface that lets you type in text and choose handwriting styles. Also, the program could work better if it used a neural network or an algorithm for machine learning to make the handwriting.

4.3.5 5_writer.py:

Code:-

```
from PIL import Image
import os
import random
import re

# Define font color, threshold for intensity, character and line spacing, font
size and ratio
font_color_range = ((37, 21, 113, 170), (47, 31, 173, 220))

threshold = 30
char_space_range = (-10, -13)
line_space_range = (0, 10)
word_space = (30, 40)
font_size_range = (20, 22)
angle_range = (-7, 10)
font_ratio = 3
padding_left = 350
```



```

padding_right = 490
padding_top = 330
padding_bottom = 150

bagrd = "Data/pagebg/new1.png"

def format_text(text, font_size, font_ratio, word_space, char_space, line_space,
padding_left, padding_top, bg_img_path):
    # Load gap image
    gap = Image.new('RGBA', (word_space, word_space), color=(0, 0, 0, 0))
    # Load character images
    char_imgs = {}
    for i in range(32, 127):
        try:
            char_imgs[chr(i)] =
Image.open("Data/4_datab/{i}.png".format(i)).convert('RGBA')
        except FileNotFoundError:
            char_imgs[chr(i)] = gap.copy()

    # Load background image and get page dimensions
    bg_img = Image.open(bg_img_path).convert('RGBA')
    page_width = bg_img.size[0] - padding_right
    page_height = bg_img.size[1] - padding_top

    # Define output text
    formatted_text = ""
    # Split input text by lines first
    lines = text.split("\n")
    # Define initial line and gap values
    line = ""
    gap = padding_left
    ht = padding_top
    # Loop through each line
    for i, line_text in enumerate(lines):
        # Split line into words using a regular expression that matches one or
more whitespace characters
        words = re.findall(r'\S+|\s+', line_text)
        # Loop through each word
        for j, word in enumerate(words):
            # Add current word to current line
            new_line = line + word if len(line) > 0 else word
            # Calculate width of current line
            line_width = padding_left
            for c in re.findall(r'\S|\s', new_line):

```

```

        if c == " ":
            line_width += char_imgs[" "].width * font_ratio * font_size /
100 + word_space + char_space
        else:
            line_width += char_imgs[c].width * font_ratio * font_size /
100 + char_space
        # If current line fits on page, update line and gap values
        if line_width <= page_width:
            line = new_line
            gap += char_imgs[word[0]].width * font_ratio * font_size / 100 +
word_space + char_space
            # If current line doesn't fit on page, add current line to output
text and start a new line
        else:
            formatted_text += line + "\n"
            line = word
            gap = padding_left
            ht += font_size + line_space
        # If current page is full, add current line to output text and start
a new page
        if ht + font_size > page_height:
            formatted_text += line + "\n"
            line = ""
            gap = padding_left
            ht = padding_top
            formatted_text += "\n"
        # Add last line to output text
        if len(line) > 0 and j == len(words) - 1:
            formatted_text += line
            if i != len(lines) - 1:
                formatted_text += "\n"
            line = ""
            gap = padding_left
            ht += font_size + line_space

    return formatted_text

# Load background image and get page dimensions
imgsize = Image.open(bagrd).convert('RGBA')
page_width = imgsize.size[0]
page_height = imgsize.size[1] - padding_bottom

input_files = ['Data/dummy.txt']

```

```

for input_file in input_files:
    with open(input_file, "r") as txt:
        text1 = txt.read()

    text = format_text(text1, (font_size_range[1]+font_size_range[1])//2,
font_ratio, (word_space[0]+word_space[1])//2,
(char_space_range[0]+char_space_range[0])//2, line_space_range[0], 0,
padding_top,bagrd)

    output_file = os.path.join("output/",
os.path.splitext(os.path.basename(input_file))[0] + "_output_1.png")

    BG = Image.open(bagrd).convert('RGBA')

    gap, ht = 0, padding_top

    file_num = 1

    for line_num, line in enumerate(text.split("\n")):
        gap = random.randint(padding_left, padding_left+15)

        line_height = 0
        for i in range(len(line)):
            if line[i] == " ":
                gap += random.randint(word_space[0], word_space[1])
                continue
            elif i < len(line) - 1 and line[i+1] == " ":
                char_space = 0
            else:
                char_space = random.uniform(char_space_range[0],
char_space_range[1])

            try:
                cases =
Image.open("Data/4_datab/{i}.png".format(str(ord(line[i])))).convert('RGBA')
            except FileNotFoundError:
                cases = gap.copy()

            data = cases.getdata()
            newData = []
            # choose a random font color for each character
            font_color = tuple([random.randint(font_color_range[0][j],
font_color_range[1][j]) for j in range(4)])

```

```

for pixel in data:
    intensity = pixel[0] + pixel[1] + pixel[2]
    if intensity < threshold:
        newData.append(font_color)
    elif pixel[3] == 0:
        newData.append((0, 0, 0, 0))
    else:
        newData.append((255, 255, 255, 255))
cases.putdata(newData)

font_size = random.randint(font_size_range[0], font_size_range[1])
new_width = int(cases.width * font_ratio * font_size / 100)
new_height = int(cases.height * font_ratio * font_size / 100)
cases = cases.resize((new_width, new_height), resample=Image.LANCZOS)
angle = random.uniform(angle_range[0], angle_range[1])
cases = cases.rotate(angle, resample=Image.BICUBIC, expand=True)

if ht + cases.height > page_height:
    BG.save(output_file)
    BG = Image.open(bagrd).convert('RGBA')
    ht = padding_top

    file_num += 1

if gap + cases.width + char_space > page_width:
    ht += line_height + random.randint(line_space_range[0],
line_space_range[1])
    gap = padding_left
    line_height = cases.height
else:
    line_height = max(line_height, cases.height)
    BG.paste(cases, (int(gap), int(ht)), mask=cases)
    size = cases.width
    gap += size + char_space
    print("printing ... character: - ", i)
    ht += line_height + random.randint(line_space_range[0],
line_space_range[1])
    if ht > page_height:
        BG.save(output_file)
        BG = Image.open(bagrd).convert('RGBA')
        ht = padding_top
        file_num += 1
    output_file = os.path.join("output/{}.png".format(file_num))
    BG.save(output_file)

```

Output Image_6:-

Narendra Damodardas Modi is an Indian politician and the current Prime Minister of India. He was born on September 17, 1950, in Vadnagar, a small town in Gujarat, India. Modi completed his schooling in Vadnagar and obtained a Bachelor's degree in Political Science from the University of Delhi.

Modi began his political career in 1985 as an organizer for the Bharatiya Janata Party (BJP), a right-wing political party in India. In 2001, he became the Chief Minister of Gujarat, a position he held for three consecutive terms until 2014. During his tenure as Chief Minister, Modi implemented several economic and social reforms that earned him both praise and criticism.

In 2014, Modi was elected as the Prime Minister of India, defeating the Indian National Congress, which had been in power for ten years. He won a historic landslide victory with the BJP securing a majority in the Lok Sabha, the lower house of the Indian Parliament.

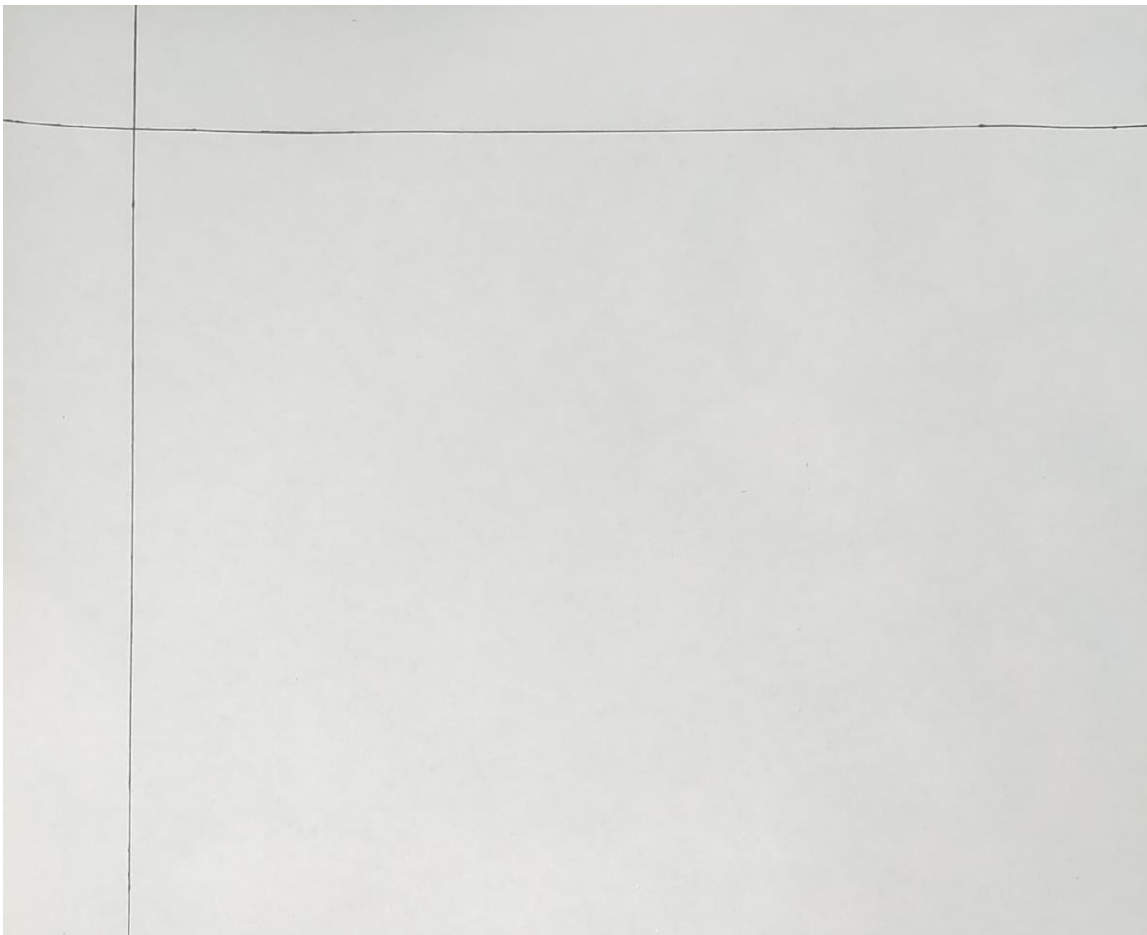
As Prime Minister, Modi has implemented several initiatives aimed at promoting economic growth and development, including the Make in India, Digital India, and Skill India programs. He has also launched several

The code is an implementation of a function that takes a text string, font size, font ratio, word space, character space, line space, padding_left, padding_top, and the path to a background image. With the given font size, font ratio, word space, character space, and line space, the function formats the text so that it fits into the size of the background image. The formatted text is then sent back.

The code loads the image for each character, the image for the background, and the gap images that are needed to format the text. The text is then split into lines and words. After adding each word, the function figures out how much wider the current line is. If the current line width is less than or the same as the page width, the function adds the current word to the current line, changes the gap value, and moves on to the next word. If the current line width is bigger than the page width, the function adds the current line to the output text, starts a new line with the current word, and changes the gap value. If the current page is full, the function adds the current line to the output text, starts a new page, and moves on to the next word.(Howse)

Lastly, the code reads the text to be formatted from an input file and uses the formatting function on it. The formatted text is then added to the background image, and the whole thing is saved as an output file.

4.4 PAGE_IMAGE :-



5. FUTURE SCOPE

Using computer vision to convert text to handwriting is a field that is growing quickly and has many possible uses. As technology keeps getting better, we can expect these techniques to be used in more advanced and complicated ways.

1. Improving the quality of handwriting: At the moment, handwritten text is simple and consistent. Future work could focus on making handwriting styles that are more realistic and varied, or even on simulating the handwriting of certain people.
2. Better recognition of different languages: The current system works well for English text, but more work could be done to support other languages or even different styles of handwriting in different parts of the world.
3. Integration with other applications Text-to-handwritten conversion could be added to a number of other programs, such as note-taking or document management software. By adding this feature, users could quickly turn typed text into notes or annotations that look like they were written by hand.
4. Further development of deep learning models: Deep learning has a lot of potential for computer vision applications, and future research could focus on making more complex models that can improve the accuracy of text recognition and handwriting simulation.
5. Exploration of real-time applications: The current implementation needs a static image of typed text, but future work could focus on making real-time applications that can recognize and convert typed text into handwritten form as the user types.

6. CONCLUSION

The conversion of typed text into handwritten form is an intriguing application of computer vision that has a wide variety of useful applications. Python was used to discuss the implementation of the text-to-handwritten conversion that was discussed in this paper. The implementation can be broken down into three distinct steps: cropping the image, extracting the text, and converting it to a format that looks like it was handwritten. The computer program files that are provided in this article can serve as a jumping off point for additional research and development in this area of study.

7. REFERENCE

1. Abhishek, Drode, et al. *Open CV Based Information Extraction From Cheques*. In, 2020.
2. Darshan, Trivedi, et al. *Handwritten Character Recognition Using Deeplearning*. In, 2018.
3. Rishabh, Jain. *Getting Started With OpenCV*. Advanced Home Automation Using Raspberry Pi Building Custom Hardware Voice Assistants and Wireless Nodes, 2021.
4. Joshi, Prateek. *OpenCV With Python by Example*. Packt Publishing Ltd, 2015.
5. Paul, D., and T. Rickard Scott. *Automatic ASCII Art Conversion of Binary Images Using Nonnegative Constraints*. 2008.
6. Balwant, Ram, et al. *Novel Image Processing Technique for Feature Detection of Wheat Crops Using Python OpenCV*. In, 2019.
7. Howse, Joseph. *OpenCV computer vision with python*. Vol. 27. Birmingham Packt Publishing, 2013.
8. Veeresh, Ambe, et al. *An Intelligent Text Reader Based on Python*. In, 2020.
9. Dey, Sandipan. *HandsOn Image Processing With Python Expert Techniques for Advanced Image Analysis and Effective Interpretation of Image Data*. Packt Publishing Ltd, 2018.

PAPER NAME

Rautish Kumar_2.pdf

WORD COUNT

3234 Words

CHARACTER COUNT

15711 Characters

PAGE COUNT

8 Pages

FILE SIZE

175.1KB

SUBMISSION DATE

May 9, 2023 12:09 PM GMT+5:30

REPORT DATE

May 9, 2023 12:09 PM GMT+5:30**● 6% Overall Similarity**

The combined total of all matches, including overlapping sources, for each database.

- 2% Internet database
- 2% Publications database
- Crossref database
- Crossref Posted Content database
- 5% Submitted Works database



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 23/02/2023.

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 01 of 08

Week Duration: 16/02/2023 to 22/02/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 16/02/2023	Explored Various sites to get the Ideas.
FRI / 17/02/2023	Studied Python Basic.
MON/20/02/2023	Studied Working Of Open-CV
TUE/ 21/02/2023	Continued previous day work
WED/ 22/02/2023	Continued previous day work

Note: Student must include all the original signed WPRs in the NTCC Final Report.

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 02/03/2023.

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 02 of 08

Week Duration: 23/02/2023 to 01/03/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 23/02/2023	Planned for the process of execution to study on my Seminar Topic.
FRI / 24/02/2023	Explored various sites containing the information
MON/27/02/2023	Explored various sites containing the information
TUE/28/02/2023	Researched for papers
WED/01/03/2023	Researched for papers

Note: Student must include all the original signed WPRs in the NTCC Final Report.

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 09/03/2023.

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 03 of 08

Week Duration: 02/03/2023 to 08/03/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 02/03/2023	Planned for the process of execution to study on my Seminar Topic.
FRI/ 03/03/2023	Explored various sites containing the information
MON/06/03/2023	Did research
TUE/07/03/2023	Continued the work
WED/08/03/2023	Continued the work

Note: Student must include all the original signed WPRs in the NTCC Final Report.

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 16/03/2023.

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 04 of 08

Week Duration: 09/03/2023 to 15/03/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 09/03/2023	Explaining Handwritten Text
FRI/ 10/03/2023	Explored various sites containing the information.
MON/13/03/2023	Read the papers
TUE/14/03/2023	Downloaded more papers
WED/15/03/2023	Continued the work

Note: Students must include all the original signed WPRs in the NTCC Final Report.

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 23/03/2023.

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 05 of 08

Week Duration: 16/03/2023 to 22/03/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 16/03/2023	Wrote future scope of Implementation of water level indicator using Arduino.
FRI / 17/03/2023	Wrote future scope of Implementation of water level indicator using Arduino.
MON/20/03/2023	Searched for more papers and had a view on it.
TUE/21/03/2023	Searched for more papers and had a view on it.
WED/22/03/2023	Debugging Error from the implementing code.

Note: Student must include all the original signed WPRs in the NTCC Final Report.

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 30/03/2023.

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 06 of 08

Week Duration: 23/03/2023 to 29/03/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 23/03/2023	Error solving
FRI/ 24/03/2023	Writing case Study.
MON/27/03/2023	Writing Problem Statement
TUE/28/03/2023	Writing methodology.
WED/29/03/2023	Writing methodology.

Note: **Student must include all the original signed WPRs in the NTCC Final Report.**

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 06/04/2023

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 07 of 08

Week Duration: 30/03/2023 to 05/04/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 30/03/2023	Code implementation.
FRI / 31/03/2023	Code implementation.
MON/03/04/2023	Code implementation
TUE/04/04/2023	understanding methodology.
WED/05/04/2023	understanding methodology.

Note: Student must include all the original signed WPRs in the NTCC Final Report.

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)



NTCC WEEKLY PROGRESS REPORT (WPR) (Even Semester 2022 - 2023 Session)

Student Name: Rautish Kumar **Enrol No.:** A35404820029

Program: BCA

Batch: 2020-23 **Sem-** 06

FG Name: Ms. Laxmi Kumari Pathak **NTCC Commencement Date:** 16/02/2023 **NTCC Completion Date:** 13/04/2023

WPR Submitted On: 13/04/2023.

NTCC Work Title: Implementation of Text To Handwriting using Python

NTCC – Major Project – [ETMJ100] **WPR Wk. No:** 08 of 08

Week Duration: 06/04/2023 to 12/04/2023

DAY / Date	Summary (precise & quantified information should be given here and student must show their weekly progress to FG)
THU/ 06/04/2023	Understanding algorithms
FRI / 07/04/2023	Writing conclusion.
MON/10/04/2023	Writing limitation.
TUE/11/04/2023	Arranging the report.
WED/12/04/2023	Pasting code in the report.

Note: **Student must include all the original signed WPRs in the NTCC Final Report.**

Rautish Kumar

Student Signature with Date

(Industry Guide Signature with date, if any)

(Faculty Guide Signature with Date)