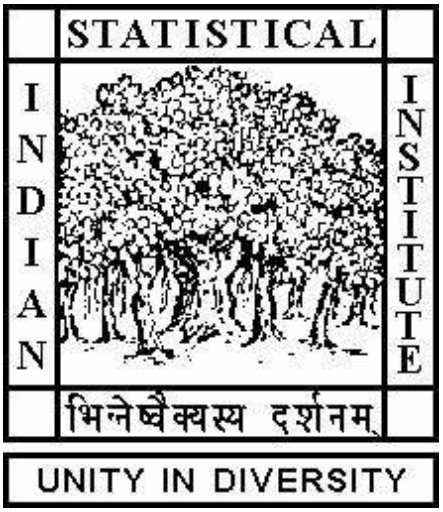
Hindi Script Refinement & Improved OCR

BS-2342 Satyaprakash Sahdev Jaiswar
BS-2344 Saurabh Bhikan Nankar
BS-2308 Aman Singh
BS-2353 Subhadip Roy
BS-2317 Aryan Sahu
BS-2335 Pallav Sahay

May 20, 2024



1 Abstract

Hindi, one of the most widely spoken languages globally, presents unique challenges for Optical Character Recognition (OCR) due to its complex script. This project addresses the issue by proposing a new set of 60 characters optimized for ease of writing and mechanical differentiation, enhancing OCR efficiency. Inspired by simple shapes, limited complexity, and consistent distinctive features, these characters aim to streamline both manual writing and automated recognition processes. Leveraging the K-nearest neighbors (KNN) classification algorithm, a model is trained on a dataset comprising handwritten characters contributed by the project team. Preprocessing and segmentation utilize established algorithms within the Python OpenCV module. The resulting OCR application demonstrates improved accuracy and usability for Hindi text recognition, marking a significant advancement in bridging the gap between manual and digital Hindi writing systems.

2 Introduction

The Hindi language, one of the most widely spoken languages globally, poses unique challenges for optical character recognition (OCR) due to its complex script. While Hindi is rich in cultural heritage and widely used in literature, administration, education, and everyday communication, its intricacies present obstacles for mechanical interpretation.

In this project, we address the challenges of OCR for Hindi by proposing a novel set of characters designed to enhance ease of writing and mechanical distinguishability. Recognizing the significance of simplicity in character design, we meticulously crafted a set of 60 symbols tailored to streamline both writing and OCR processes for Hindi.

Our approach aimed to bridge the gap between traditional script complexity and modern computational requirements. By introducing characters that are intuitive to write and easily discernible by machines, we sought to revolutionize the OCR landscape for Hindi language applications.

The culmination of our efforts resulted in the development of an OCR application specifically tailored for the newly designed Hindi characters. This application promises to significantly enhance the efficiency and accuracy of Hindi text recognition, opening doors to a myriad of applications across various domains.

Through this project, we aim to contribute to the advancement of Hindi language technology and facilitate greater accessibility and usability of Hindi content in the digital age.

PART 1: Character Design

The Devanagari script, used for writing Hindi among other languages, presents challenges in both writing and recognition due to its intricate characters and writing rules. Tradi-

tionally, writing Hindi involves sketching a guiding line followed by the characters, often resulting in curvy and complex shapes that require refinement.

To address these challenges, our design considerations for creating a new set of Hindi characters were as follows:

1. **Simple Shapes:** We aimed to utilize basic geometric shapes such as lines, circles, squares, and triangles, as well as their combinations. This simplification minimizes the complexity of characters, making them easier to distinguish in scans and write by hand.

2. **Distinctive Features:** Each symbol was designed with distinctive features to ensure easy differentiation from others. Variations in shape, size, orientation, or strokes were incorporated to enhance readability and recognition.

3. **Consistency:** Consistency in design across the alphabet was crucial. We maintained uniform stroke width and proportions to ensure that each symbol is unique and recognizable, aiding both writing and scanning processes.

4. **Limited Complexity:** Keeping the overall complexity of the alphabet low was essential for ease of learning and memorization. This approach also contributes to improved accuracy in both writing and scanning.

5. **Legibility:** We prioritized legibility by avoiding symbols that are too similar or easily confused. Clear differentiation between characters is vital for accurate writing and scanning.

6. **Test and Refine:** The alphabet underwent rigorous testing with various handwriting styles and scanning methods to validate its efficacy. Feedback and usability testing guided continuous refinement to meet desired criteria.

7. **Size Consistency:** Maintaining a consistent size for all symbols was crucial to prevent confusion during interpretation and ensure uniformity across the alphabet.

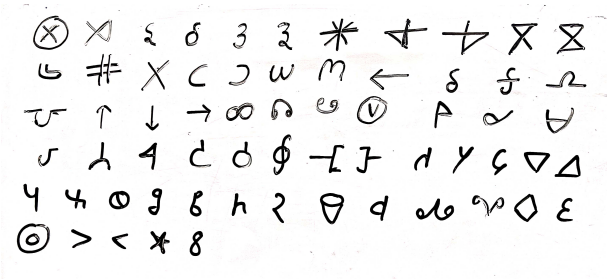


Figure 1: Prototype Characters for Hindi Language

After initial prototyping and refinement, our final set of characters was achieved. While some unworthy characters were discarded during the iterative design process, the finalized characters represent a significant improvement in both ease of writing and mechanical differentiation.

Next, we proceed to create a training dataset and develop an OCR application tailored to these newly designed Hindi characters.

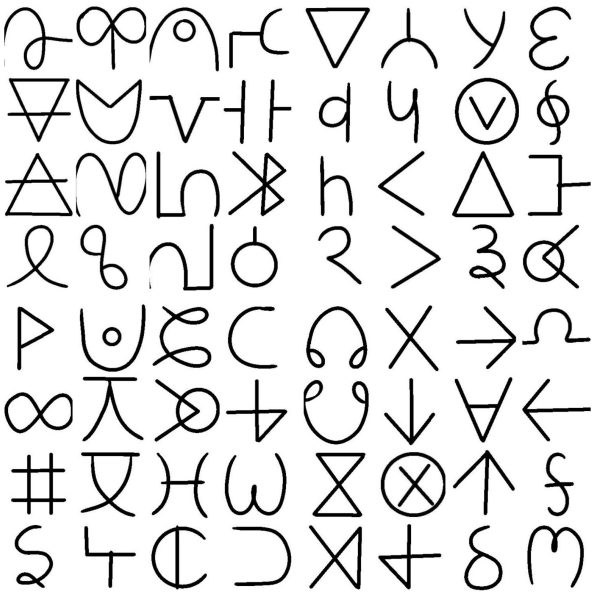


Figure 2: Final Characters

Part 2: Training OCR Model

After finalizing the characters, we created a diverse training dataset [1] by writing each character multiple times by a lot of people to capture various handwriting styles and variations. This dataset serves as the foundation for training our OCR model, ensuring robustness and accuracy. By exposing the model to a wide range of handwriting variations, we aim to enhance its capability to accurately recognize characters in real-world scenarios. In the next phase, we utilize this dataset to train our OCR model, focusing on preprocessing, feature selection, and algorithm optimization to achieve superior performance and accuracy in recognizing handwritten Hindi text.

Note: We utilized the Python OpenCV module to execute various preprocessing and segmentation tasks within the OCR model. This choice was made due to its robust functionality and efficiency in handling image processing operations.

I: Preprocessing[2] Function

After finalizing the design of the characters, we proceeded with the development of the OCR application. The first step in this process involved creating a preprocessing function. This function is crucial as it refines the input image to contain only the necessary features required for OCR. The preprocessing function consists of the following steps:

1. **Resize:** Resizing the image ensures uniformity in processing and reduces computational complexity.

2. **Adjust Brightness and Contrast:** Enhancing brightness and contrast improves image quality, aiding in text legibility.

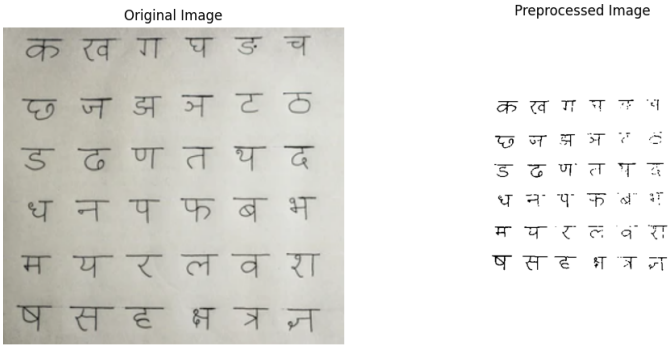
3. **De-skew:** Correcting the skew of an image aligns the text horizontally, thereby enhancing character recognition. This can be achieved by applying the Hough

Transform [4] on a Canny edge-detected image to determine the angle of rotation needed to deskew the image.

4. **Thresholding:** Converting the image to greyscale removes color information while preserving essential text features. Then converting the image into a binary format to black and white enhances contrast, making text stand out from the background simplifies further processing by separating text from the background. This involves the use of established image thresholding techniques like Otsu Thresholding [3].

5. **Noise Removal:** Removing noise improves OCR accuracy by eliminating irrelevant details. This can be achieved by applying morphological operations like erosion and dilation, after which applying median filter or gaussian filter.

6. **Erosion and Dilation [5]:** Morphological operations shrink or expand text regions to improve text connectivity and remove small artifacts.



These preprocessing steps collectively prepare the image for OCR, enhancing text visibility and reducing background noise to improve character recognition accuracy.

Next, we proceed to implement the OCR algorithm and integrate it into the application workflow.

II: Segmentation Function

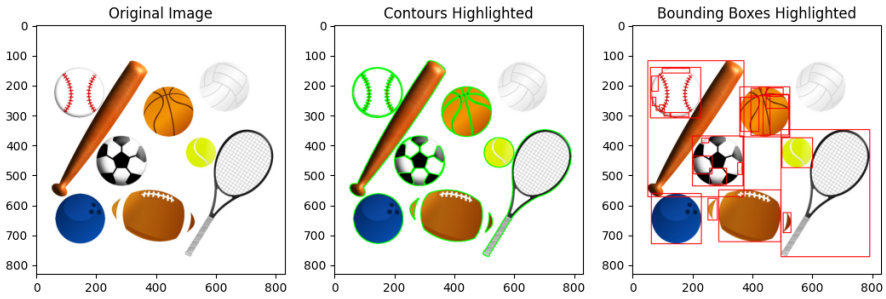
Following preprocessing, the next step in the OCR pipeline is segmentation, which involves identifying individual characters in the image. This is achieved using a segmentation function that utilizes contour detection.

Contours are essentially outlines or boundaries of objects in an image. In the context of OCR, contours represent the boundaries of individual characters. They are calculated using algorithms such as the Suzuki algorithm [6] or the Moore-Neighbor Tracing algorithm.

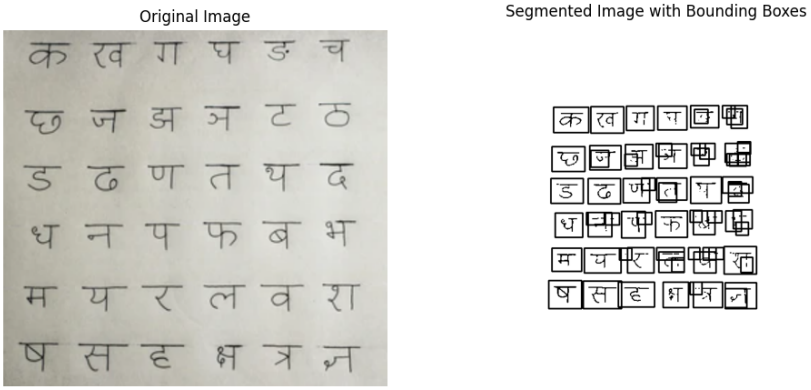
Here's how contours are calculated and how they help in finding characters in the binarized image:

1. **Contour Detection:** Contours, represented as continuous lines or curves, define the complete boundary of an object. Contour detection algorithms analyze the binarized image to identify connected regions of similar intensity. These regions correspond to the boundaries of characters in the image.

2. **Bounding Boxes:** Once contours are detected, bounding boxes are drawn around them to encapsulate individual characters. These bounding boxes serve as regions of interest for character extraction.



3. **Character Extraction:** The segmented characters are extracted from the original image based on the bounding boxes. Each bounding box represents a separate character that can be isolated for further processing.



By leveraging contour detection and bounding box techniques, the segmentation function effectively identifies and isolates individual characters in the binarized image. This step is crucial for creating the training dataset required for training the OCR model.

Next, we proceed to extract characters from the segmented regions and prepare the training data for the OCR model.

III: Training using KNN

In the training phase of the OCR pipeline, the K-nearest neighbors (KNN) algorithm is employed for classification. KNN is a simple and intuitive machine learning algorithm used for classification tasks. It operates on the principle of proximity, where an unknown sample is classified based on the majority class of its K nearest neighbors in the feature space.

Data Augmentation:
Data augmentation is a crucial technique in machine learning to enhance the diversity of a dataset without collecting additional data. We implemented augmentation by applying simple distortions and shifts [7] to existing images. Random shifts were performed by adjusting the x and y coordinates of pixels within a small percentage of the image dimensions, ensuring the changes remain subtle. Curved distortions were achieved by shifting image columns in a sinusoidal pattern, creating wave-like effects. Similarly, sinusoidal distortions introduced waveforms across the image, and elliptical distortions scaled the central portions differently than the edges, simulating natural variations in handwritten text. These methods collectively increase the variability in our dataset, improving the robustness and generalization capability of machine learning models.

SMOTE (Synthetic Minority Over-sampling Technique):
SMOTE [8] is a technique used to address class imbalance in the training dataset. It synthesizes new minority class samples by interpolating between existing samples, thereby balancing the class distribution. This is essential for improving the performance of the classifier, especially in scenarios where one class is significantly underrepresented.

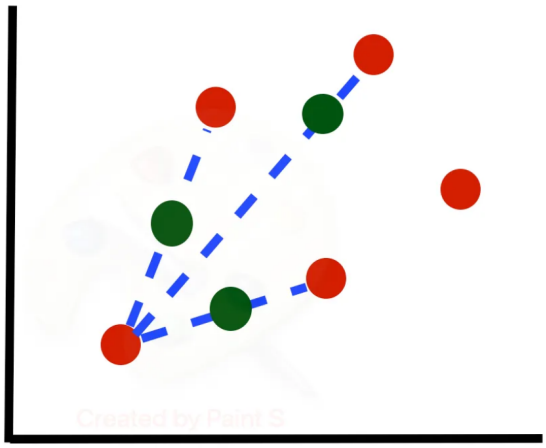


Figure 3: SMOTE working

Histogram of Oriented Gradients (HOG):
The Histogram of Oriented Gradients (HOG) [9] is a feature descriptor technique used in computer vision to extract essential information from images. It operates by calculating the distribution of gradient orientations in localized regions of an image. Gradients represent intensity changes across neighboring pixels, computed using gradient filters along the x and y directions. The image is divided into small, overlapping cells, each covering a specific region. Within each cell, histograms of gradient orientations are computed, quantizing orientations into discrete bins. These histograms accumulate gradient magnitudes based on their orientations. Finally, the histograms from all cells are concatenated to form a feature vector representing the image. [10] This vector encodes the distribution of gradient orientations across the image, providing a compact representation of shape and texture characteristics.

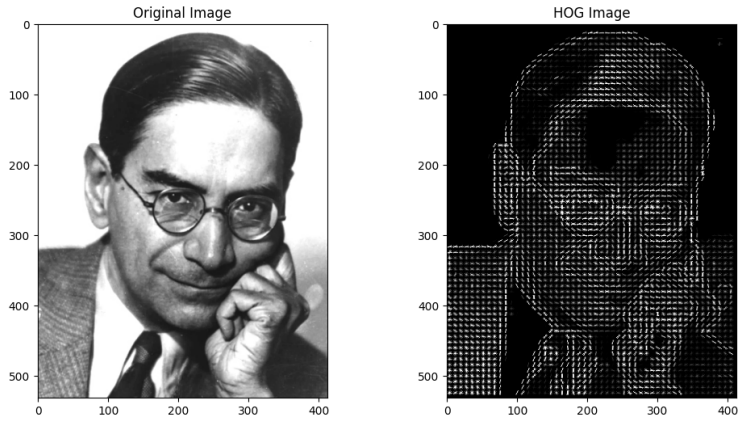


Image Space:
In the context of OCR, the image space refers to the feature space where images are represented as vectors of features. Each feature represents a characteristic of the image that aids in classification.

Using KNN to Classify Images:
K-Nearest Neighbors (KNN) [11] is a simple yet effective algorithm used for both classification and regression tasks.

In the context of image classification, KNN operates by first extracting relevant features from images, such as the Histogram of Oriented Gradients (HOG), which captures the distribution of gradient orientations in an image.

Once the HOG features are extracted, each image is represented as a vector in a high-dimensional feature space. During the classification phase, KNN calculates the distance between the test image and all training images in this feature space. It then assigns the test image to the majority class of its K nearest neighbors, where K is a predefined parameter.

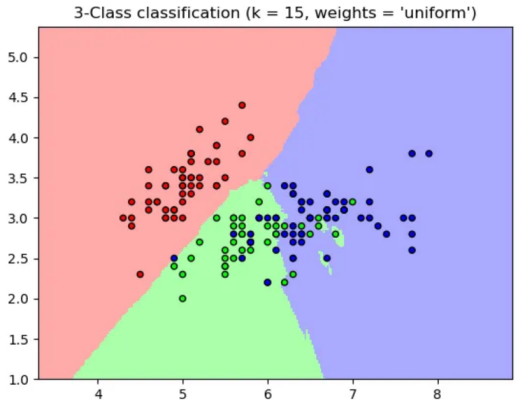


Figure 4: A k-NN example (From scikit-learn documentation)

In simpler terms, KNN classifies an unknown image by comparing it to the labeled images in its training set. The algorithm finds the K images in the training set that are closest to the unknown image based on some distance metric (e.g., Euclidean distance), and then assigns the unknown image to the class that is most common among its K nearest neighbors.

This approach is intuitive and often effective, especially when dealing with relatively small datasets or when the decision boundaries between classes are well-defined. However, it can be computationally expensive, especially as the size of the training set grows, since it requires calculating distances between the test image and all training images. Additionally, the performance of KNN can be sensitive to the choice of distance metric and the value of K.

Joblib File:
Joblib is a library in Python used for serialization and deserialization of Python objects. It is particularly useful for saving machine learning models to disk, allowing for easy retrieval and reuse. The trained KNN model is dumped to a joblib file for later use in predicting characters from new images.

Next, we proceed to implement the trained KNN model in the OCR application for character recognition.

PART 3: OCR Application

After training the OCR model and saving it to a joblib file, we developed a simple python program for character recognition using the 'model.joblib' file. The application takes an input image as its input and generates output image with predicted characters that is, it replaces the characters in the original input image with the predicted characters obtained from the OCR model.

The OCR application utilizes the previously developed functions and codes for preprocessing, segmentation, and character recognition.
Below are the results of evaluating the OCR model using the confusion matrix and classification report:

Confusion Matrix: A confusion matrix is a powerful tool for evaluating the performance of a classification algorithm. It provides a detailed breakdown of the algorithm's performance by comparing the actual target values with the predictions made by the model.

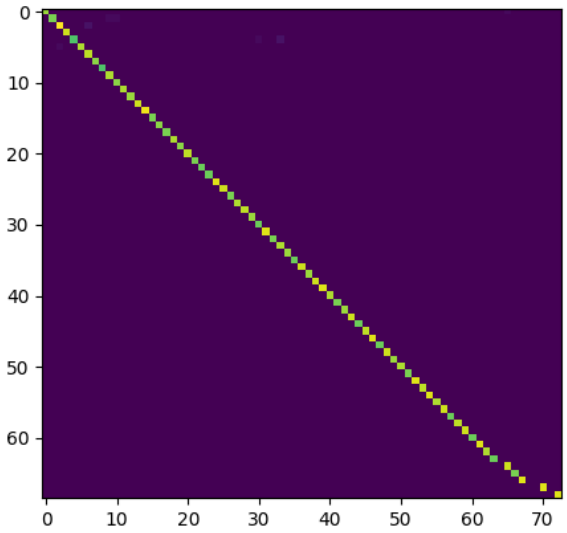


Figure 5: Confusion Matrix

Classification Report: A classification report is a performance evaluation metric in machine learning that summarizes the performance of a classification algorithm. It provides several key metrics to assess the quality of predictions made by the model. Here's a breakdown of the main components of a classification report:

Precision: The ratio of true positive predictions to the total predicted positives (true positives + false positives).
Interpretation: High precision indicates that the model has a low false positive rate. It answers the question, "Of all the instances predicted as positive, how many were actually positive?"

Recall: (Sensitivity or True Positive Rate):The ratio of true positive predictions to the total actual positives (true positives + false negatives).
Interpretation: High recall indicates that the model has a low false negative rate. It answers the question, "Of all the actual positive instances, how many were correctly predicted as positive?"

F1-Score: The harmonic mean of precision and recall.
Interpretation: The F1-Score provides a single metric that balances both precision and recall. It's especially useful when you need a balance between precision and recall and there is an uneven class distribution.

Support: The number of actual occurrences of the class in the dataset.
Interpretation: Support helps to understand how many instances of each class are present in the actual dataset. It provides context to the precision, recall, and F1-score.

	precision	recall	f1-score	support
0	0.89	0.97	0.93	35
1	0.97	0.94	0.96	34
2	1.00	0.89	0.94	45
3	0.93	1.00	0.96	37
4	0.94	0.85	0.89	34
5	0.83	1.00	0.91	15
6	0.93	0.96	0.94	26
7	0.94	0.96	0.95	46
8	0.83	0.90	0.86	21
9	0.85	0.85	0.85	27
10	0.73	1.00	0.84	24
11	0.85	1.00	0.92	11
12	0.92	0.92	0.92	24
13	0.84	1.00	0.91	21
14	0.96	1.00	0.98	25
15	0.97	0.93	0.95	30
16	1.00	0.96	0.98	28
17	1.00	0.97	0.99	39
18	0.97	0.94	0.96	34
19	0.97	1.00	0.99	37
20	0.92	1.00	0.96	47
21	0.92	0.92	0.92	25
22	0.92	0.79	0.85	14
23	0.91	1.00	0.96	32
24	0.88	0.82	0.85	28
25	0.95	1.00	0.97	38
26	0.97	0.90	0.94	41
27	0.95	0.87	0.91	45
28	0.97	0.91	0.94	33
29	1.00	0.94	0.97	32
30	0.80	0.93	0.86	30
31	0.93	0.98	0.95	42
32	1.00	1.00	1.00	41
33	0.88	0.93	0.90	30
34	0.92	1.00	0.96	36
35	0.94	0.94	0.94	31
36	0.95	0.95	0.95	41
37	0.93	1.00	0.96	27
38	0.97	0.97	0.97	30
39	0.96	0.94	0.95	48
40	1.00	0.93	0.96	29
41	0.91	0.94	0.92	32
42	1.00	1.00	1.00	32
43	0.94	0.97	0.96	33
44	0.96	1.00	0.98	26
45	0.97	0.91	0.94	35
46	0.97	0.92	0.94	36
47	0.93	1.00	0.96	25
48	0.97	0.97	0.97	30
49	0.97	0.92	0.95	39
50	0.97	0.93	0.95	30
51	0.98	0.89	0.93	45
52	1.00	0.94	0.97	36
53	1.00	0.91	0.95	33
54	0.96	0.89	0.92	27
55	1.00	0.88	0.93	32
56	0.92	0.94	0.93	35
57	0.87	0.81	0.84	16
58	1.00	1.00	1.00	22
59	0.97	0.97	0.97	38
60	0.97	0.94	0.95	31
61	0.97	0.97	0.97	29
62	0.95	0.95	0.95	20
63	0.98	0.91	0.95	57
accuracy			0.94	2052
macro avg	0.94	0.94	0.94	2052
weighted avg	0.95	0.94	0.94	2052

Figure 6: Classification report

The following is the mapping between hindi characters and our designed characters:

[0: 'ँ', 1: 'ं', 2: 'ः', 3: 'अ', 4: 'आ', 5: 'इ', 6: 'ई', 7: 'उ', 8: 'ऊ', 9: 'ऋ', 10: 'ए', 11: 'ऐ', 12: 'ओ', 13: 'औ', 14: 'क्', 15: 'क्ष्', 16: 'ख्', 17: 'ग्', 18: 'घ्', 19: 'ङ्', 20: 'च्', 21: 'छ्',

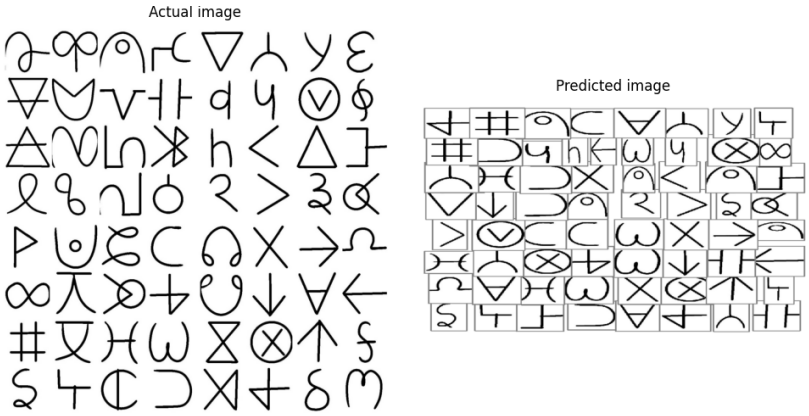
22: 'ज्', 23: 'ज्ञ्', 24: 'झ्', 25: 'ञ्', 26: 'ट्', 27: 'ठ्', 28: 'ड्', 29: 'ढ्', 30: 'ण्', 31: 'त्', 32: 'त्र्', 33: 'थ्', 34: 'द्', 35: 'ध्', 36: 'न्', 37: 'प्', 38: 'फ्', 39: 'ब्', 40: 'भ्', 41: 'म्', 42: 'य्', 43: 'र्', 44: 'ल्', 45: 'ळ्', 46: 'व्', 47: 'श्', 48: 'ष्', 49: 'स्', 50: 'ह्', 51: 'ः', 52: 'ा', 53: 'ि', 54: 'ी', 55: 'ु', 56: 'ू', 57: 'ृ', 58: 'े', 59: 'ै', 60: 'ो', 61: 'ौ', 62: '्', 63: '।']

Preprocessing Note

Depending on the input image, it may be necessary to make adjustments to various parameters and the sequence or selection of sub-functions in the preprocessing phase to obtain an optimized preprocessed image for character recognition. Additionally, since we have designed characters corresponding to Hindi alphabets, we encounter the challenge of dealing with Hindi matras. To address this, we have employed the concept of "vard vichhed" (वर्ण-विच्छेद) in Hindi words. For instance:

1. Combining अ + च् + छ् + आ = अच्छा.
2. Combining ग् + उ + ल् + आ + ब् + अ = गुलाब.
3. Combining च् + आ + ँ + द् + न् + ई = चाँदनी.

Result



As you can see Outputs are not that good. Reasons mentioned in further improvement section.

3 Conclusions

In this project, we embarked on a journey to enhance the Optical Character Recognition (OCR) capabilities for the Hindi language. Recognizing the challenges posed by the intricate Devanagari script, we began by redesigning a set of characters optimized for ease of writing and mechanical differentiation. Through careful consideration of design principles such as simplicity, distinctiveness, and consistency, we developed a new set of characters that streamline both manual writing and automated recognition processes.

With our characters finalized, we proceeded to create a comprehensive training dataset by capturing diverse handwriting styles and variations for each character. This dataset served as the foundation for training our OCR model, aiming to achieve robustness and accuracy in recognizing handwritten Hindi text. Leveraging preprocessing techniques, feature selection, and algorithm optimization, we trained our model to accurately classify and recognize characters across various contexts and scenarios.

Throughout the project, our focus remained on bridging the gap between manual and digital Hindi writing systems. By developing an OCR model tailored to our newly designed characters, we aimed to enable seamless and accurate recognition of handwritten Hindi text, thereby facilitating digital transformation and accessibility in Hindi language processing.

In conclusion, our project represents a step forward in enhancing OCR capabilities for the Hindi language, contributing to advancements in digital literacy and accessibility for Hindi speakers worldwide. As we continue to refine and optimize our OCR model, we envision a future where handwritten Hindi text can be effortlessly and accurately digitized, opening up new opportunities for communication, education, and information dissemination.

Further Improvements

While our current OCR model shows promising results, there are several avenues for further improvement and enhancement:

1. **Feature Extraction:** Even after significant efforts and achieving reasonably good accuracy, our OCR application struggled to predict new input characters accurately, often yielding poor results. To address this issue, We can go to further step and incorporated additional feature extraction methods such as zoning and projection histograms, thus adopting a combined feature approach [13]. Each feature extraction method contributes to forming a feature vector, and by using a combination of features, we derive feature vectors with more elements. This comprehensive feature set enhances the efficiency of recognition.

2. **Improving Accuracy:** While training models for character recognition, it's essential to consider the limitations of creating a dataset with limited variation. When the training dataset contains only a small number of images for each character, the model may struggle to generalize effectively to unseen data, leading to lower accuracy and suboptimal predictions.

   To address this limitation, it's beneficial to increase the variation in the training dataset. This can be done using techniques like 'Structural Crossing-Over Technique' [12] and Generative Adversarial Networks or GANs. By including more than (say) 1000 images for each character and ensuring a diverse range of samples, we can provide the model with a richer set of features to learn from. This increased variation enables the model, particularly K-Nearest Neighbors (KNN), to better

capture the underlying patterns in the data, resulting in higher accuracy and more reliable predictions.

Therefore, to achieve higher accuracy and improve the quality of predictions in character recognition tasks, it's advisable to augment the training dataset with a significant number of images for each character, ensuring sufficient variation and diversity.

3. **Exploration of Different Classification Algorithms:** While the K-nearest neighbors (KNN) algorithm was chosen for its simplicity and effectiveness, exploring other classification algorithms such as Support Vector Machines (SVM), Random Forests, or Deep Learning models like Convolutional Neural Networks (CNNs) could lead to improved accuracy and performance. CNNs, in particular, have shown remarkable success in image classification tasks due to their ability to learn hierarchical features directly from pixel data.

4. **Incorporation of Line and Word Detection:** By implementing techniques [14] such as vertical and horizontal histogram projections, we can enhance our OCR system to detect lines and words within the text. This would allow for more accurate segmentation and recognition of individual words, improving overall accuracy and readability of the extracted text.

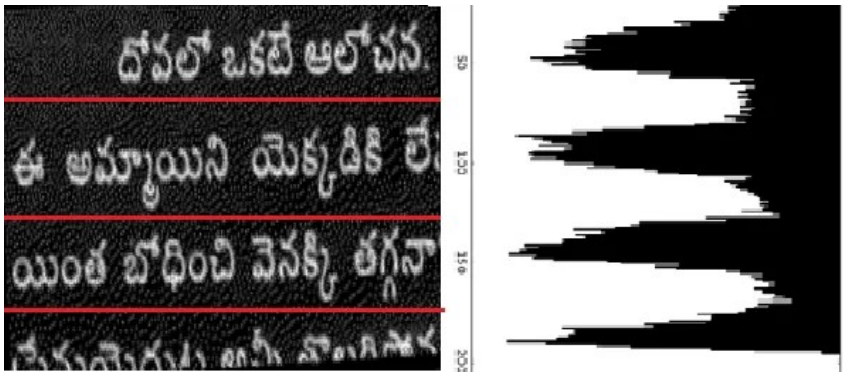


Figure 7: Line Segmentation

5. **Language Formatting and Spell Checking:** Integration of language formatting and spell checking functionalities would enable the OCR system to output words and sentences in the correct language format. Additionally, incorporating a dictionary to identify and correct misspelled words or words incorrectly identified by the OCR would further enhance the accuracy and usability of the system.

6. **Utilization of Designed Characters for Faster and Easier Writing:** The newly designed characters aim to make Hindi writing faster, easier, and more distinguishable. By incorporating these characters into our OCR system, we can streamline the process of Hindi text input and make it more accessible to a wider audience. Additionally, we could explore developing a simpler cursive style for writing characters to facilitate quicker handwriting. We can also apply the 'Huffman Encoding'

principle to map the easiest-to-write characters to the most frequently used characters in the Hindi alphabet. This would streamline the writing process, making it more efficient and accessible.

7. **Extension to Different Languages and Fonts:** Finally, the OCR system can be extended to support different languages and fonts, allowing for broader applicability and usability. By adapting the system to recognize characters from various languages and fonts, we can cater to diverse user needs and facilitate text recognition in multilingual environments.

Incorporating these improvements would not only enhance the performance and accuracy of our OCR system but also extend its usability and applicability to a wider range of scenarios and languages. By continuously refining and evolving the system, we can pave the way for advancements in text recognition technology and contribute to the accessibility and digital transformation of linguistic diversity.

4 References

References

[1] A Universal Way to Collect and Process Handwritten Data for Any Language

[2] GitHub : OCR Python Textbook (pre-processing).

[3] Wikipedia : Otsu's Method for thresholding.

[4] Machine Vision Recipes: Deskewing document images

[5] Towards Data Science : Introduction to Image Processing with Python (dilation and erosion).

[6] Medium : Suzuki's Contour Tracing Algorithm.

[7] Synthetic data generation for Indic handwritten text recognition

[8] Medium : Synthetic Minority Over-sampling Technique (SMOTE).

[9] Medium : Introduction to Histogram of Oriented Gradients (HOG).

[10] Medium : OCR with Machine Learning.

[11] Towards Data Science : Image Classification with K-Nearest Neighbors.

[12] AUTOMATIC TRAINING DATA SYNTHESIS FOR HANDWRITING RECOGNITION USING THE STRUCTURAL CROSSING-OVER TECHNIQUE

[13] Feature Extraction and Classification Techniques in O.C.R. Systems for Handwritten Gurmukhi Script – A Survey

[14] Towards Data Science : Segmentation in OCR.