Discover Artificial Intelligence

Research

A hybrid approach to Bangla handwritten OCR: combining YOLO and an advanced CNN

Aye T. Maung[1] · **Sumaiya Salekin**[1] · **Mohammad A. Haque**[1]

Received: 25 September 2024 / Accepted: 13 March 2025
Published online: 21 June 2025

Abstract
Optical Character Recognition (OCR) plays a vital role in automating data entry from handwritten forms into digital systems. However, a significant gap exists in the research on OCR techniques tailored for handwritten texts in complex languages such as Bangla. Challenges in Bangla script arise from the presence of modifiers, compound characters, and diacritic marks, making accurate recognition difficult. Our research introduces a scalable and effective OCR pipeline for Bangla handwritten documents that addresses these complexities. The proposed pipeline leverages the YOLO (You Only Look Once) model for character detection, accurately isolating base alphabets, consonant conjuncts, and characters with modifiers (matras). For character recognition, the pipeline utilizes the EfficientNet-B4 model, which demonstrated a recognition accuracy of 93.87% for grapheme roots, 98.22% for vowel diacritics, and 98.0% for consonant diacritics on publicly available datasets, combined and adapted for our use. Additionally, the system's resilience was enhanced using a Word2Vec-based spelling correction layer, reducing the Character Error Rate (CER) from 10.37% to 2.47%. Comparative evaluations on in-house data show that the proposed pipeline with spelling correction achieves the highest precision (0.9701) and lowest CER (0.0247), outperforming the Google Cloud Vision API's OCR. In contrast, the Vision API has the highest CER (0.1389) and lower precision (0.8220), highlighting the effectiveness of the proposed approach for Bangla OCR.

Keywords Bangla OCR · Character detection · Character recognition · Word recognition · YOLO · EfficientNet

1 Introduction

Bangla (or Bengali) ranks among the most widely spoken languages globally, with over 250 million speakers, and remarkably is the fifth most extensively utilized writing system worldwide [1, 2]. The Bengali Language Introduction Act of 1987 [3] mandates the use of Bangla in all government documentation in Bangladesh. Hence, the digitization of handwritten Bangla documents is crucial for maintaining cultural heritage and ensuring access to information for a large population. In this context, Optical Character Recognition (OCR) technology plays a vital role in providing an efficient way to convert images into text, reduce errors, and enable data extraction and manipulation for digital copies [4]. OCR enhances accessibility and allows for a smoother transition of Bangla and other languages into the digital age.

Aye T. Maung and Sumaiya Salekin contributed equally to this work.

✉ Mohammad A. Haque, arifulhoque@eee.buet.ac.bd; Aye T. Maung, 1806195@eee.buet.ac.bd; Sumaiya Salekin, sumaiyasalekin321@gmail.com | [1]Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

Table 1 Intricacies of Bangla Handwritten Characters

Reasons for Complexity	Character Image 1	Character Image 2
Same compound character written in different styles		
Same modifier written in different styles		
Variation in writing styles for different writers		
Same character extended in the upper zone due to writing style		
Compound characters, with 3 consonants, extended in the upper and lower zones		
Different compound characters looking similar due to writing style		

The images were from the character datasets mentioned in Sect. 3

Although considerable developments have been made in the field of OCR, challenges still remain in the recognition of handwritten text images for languages with complex characters and writing styles like Bangla. Our focus is to address these challenges in the context of Bangla handwritten words. The inherent intricacies of the Bangla script, which are highlighted by a wide variety of compound characters and modifiers, make proper recognition much more demanding. The complexities of the characters and words create a challenging dataset to train an OCR model, as shown in Table 1.

The primary complexity arises from the structure of the alphabets. The Bangla alphabets can be divided into two groups: 11 vowels and 39 consonants. The vowels can be written as independent letters or attached to consonants as dependent forms called modifiers, some of which may extend in the upper and lower zones of written words. The vowel "আ" (ā) can appear independently or as "া" when combined with a consonant (e.g., কা, ka).

Multiple ligatures exist in Bangla, where two or more consonants are combined into a single glyph. These conjuncts may have a completely different appearance from the base consonants. Like when, ক (ka) + ষ (Sha) becomes ক্ষ (kkha). There are approximately 171 [5] unique compound characters present in the Bengali script, which are composed of 2, 3 or 4 consonants. There also exists hasanta for suppressing the inherent vowel in a consonant, often leading to the formation of conjuncts.

The main difficulty when designing an OCR arises from the nonlinearity of Bangla scripts. Unlike linear scripts, Bengali writing involves complex positioning of diacritical marks and conjunct forms, which can appear above, below, before, or after the main consonant. The positioning of the nasalization marks, Anusvara (ং) and Chandrabindu (ঁ), also poses difficulties for the OCR model in distinguishing them from the base character. For example, চাঁদ (chānd) - the nasalization mark " ঁ " is placed above the first character "চ". Although the absence of uppercase and lowercase letters offers some simplifications, it makes it difficult to identify proper nouns. Furthermore, some Bangla characters have more than one form in handwritten text.

The enhancement of Bangla Handwritten OCR technologies has the potential to significantly impact the digitization process of South Asia. The digitization of historical and legal documents, linguistics research, handwriting analysis, cultural preservation and analysis, and healthcare record administration are some of the key fields that stand to gain from OCR applications [6]. Developments in these fields will also have an impact on the region's general standard of living.

2 Literature review

Existing approaches to handwritten word recognition can be grouped into two overarching methods, feature extraction-based approach and segmentation-based approach.

The feature extraction method focuses on extracting meaningful information from raw image data. Early approaches to OCR primarily used manually designed algorithms to analyze statistical data to distinguish specific characteristics of scripts. Wahid et al. (2021) [7] presented an important study on Bangla handwritten digit recognition where they compared the OCR performance of three handcrafted feature extraction techniques: Histogram of Oriented Gradients (HOG), Local Binary Pattern (LBP), and Gabor filters to that of four classification methods: K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Random Forest (RF), and Gradient-Boosted Decision Trees (GBDT). The study names HOG combined with SVM (HOG+SVM) as the most successful arrangement, and the recognition accuracies are 93.32%, 98.08%, 95.68%, and 89.68% for the NumtaDB, CMARTdb, Ekush, and BDRW datasets respectively.

The utilization of deep learning methods such as Convolutional Neural Network (CNN) was found to introduce an encouraging shift in the Optical Character Recognition (OCR) technological development. CNNs automatically learn hierarchical features without the need for manual feature engineering; hence, they reduce the degree of dependence on this. According to work of Alom et al. (2017) [8], Bangla handwriting has many complex patterns and structures that can be captured by deep CNNs, demonstrating its utility.The study "Arabic and Latin Scene Text Recognition by Combining Handcrafted and Deep-Learned Features" [9] mentions how integrating features that get removed using manual methods like HOG and SIFT with deep learning models such as Convolutional Neural Networks (CNNs) and Deep Sparse Auto-Encoders (SAEs) increases the precision of recognition.

Recently developed approaches for handwritten word recognition rely on sequence-to-sequence (seq2seq) models. Though commonly employed for Natural Language Processing (NLP) tasks, for OCR, the seq2seq model is adapted to convert text images to machine-readable text. Seq2seq models use an encoder to read the input images and feed it to a Recurrent Neural Network (RNN), like Long Short-Term Memory (LSTM) which acts as a decoder, to generate the output sequence. One such method is using Bi-directional LSTM (BLSTM) with Connectionist Temporal Classification (CTC) loss [10]. The architecture generates the text word by producing character posterior probabilities on a sequence. The primary problem Bangla words face in this process is when words contain modifiers, especially (ে) and (ৈ). Yet another drawback is the inability to effectively handle compound characters and modifiers, leading to sub-optimal recognition accuracy.

Another approach is the conventional segmentation methods used in handwritten word recognition [11]. The method is primarily used for typed Bangla text images. The segmentation procedure these models follow in separating characters from words is to first detect the 'matra' line. The challenge in detecting the line for handwritten words is that, owing to writing preferences, there is a disparity in the length and presence of matra for the same characters in the same words written by different persons. Then, the baseline, which is the line separating the middle zone and the lower zone, is detected. Next, for the character segmentation portion, the middle zone containing the most characters is vertically split, ignoring the matra line. The handwritten characters that originally did not have a matra in the handwritten text get vertically split. However, this problem could be solved by introducing a constraint that a character in the middle zone must touch the baseline to combine the split parts; still challenges may arise due to the difference in handwriting. The presence of modifiers and compound characters further complicates the segmentation process [12].

Instead of the segmentation method that divides the words into upper, lower and middle zones [13], segmentation can be done by focusing on dividing the word image into graphemic units, which are the smallest written units in alphasyllabary languages. These techniques hold potential for recognizing diverse scripts. In the paper by Bensefia et al. (2003) [14], graphemes are used as local features in a Grapheme-Based Writer Verification system. To assess handwriting similarity through statistical measures, a set of graphemes is extracted using connected component segmentation. Tested on a dataset of 88 writers, this method demonstrated reliable differentiation between the writings of the same and different authors, using mutual information as a robust criterion. While its primary application is in writer verification, such segmentation-based approaches underscore the importance of grapheme extraction, a challenge also relevant to OCR tasks for complex scripts like Bangla. A similar approach is proposed by Jung et al. [15] in the article *A Structural Method for Grapheme Segmentation of Hangul Characters for OCR*. They introduce a method for Hangul grapheme segmentation, where common vowel types and accompanying consonants are separated by boundary tracking to facilitate easier recognition.

We propose a method that isolates individual characters from the word image. This includes isolating simple base characters, ligatures, conjuncts, characters with diacritic marks, and characters with modifiers (matras). The You Only Look Once (YOLO) identifies the boundary box around each character using its single-forward pass technique. The EfficientNet architecture is then utilized to recognize the complex patterns in the character patches extracted using YOLO's prediction. The recognition model is trained to recognize both base and compound characters. Cascading the independently trained YOLO and EfficientNet models helps us address the detection and recognition tasks of the OCR separately.

Thus, our framework can address the subtleties of Bangla handwriting with increased effectiveness. Additionally, an optional spellchecker model can be employed at the output stage to boost accuracy. This improves dependability and successfully fixes errors across a variety of handwriting styles in real-world circumstances. Our framework aims to enhance accuracy while seeking to contribute to the ongoing progress of efficiency standards in Bangla handwritten word recognition systems through the thoughtful integration of cutting-edge technologies.

3 Datasets

Datasets from two different sources are used to train the models in the proposed pipeline, publicly available datasets and our collected image data. The publicly available datasets include two different types of data: image datasets and text datasets. A brief summary of the uses of datasets in training and testing is provided in Table 2.

3.1 Publicly available datasets

3.1.1 Handwritten grapheme dataset

The dataset contains 411,882 images of 1,295 common and approximately 900 rare Bengali graphemes. Developed for the Kaggle "Bengali Grapheme Classification" competition [16], it uses a unique grapheme-based labeling scheme for simple and compound characters [17]. The dataset separates roots, vowel diacritics, and consonant diacritics as distinct targets, making it useful for evaluating multi-target classification algorithms and supporting word-level OCR research.

Table 2 Dataset Usage

Datasets	Use in Pipeline
Publicly Available Datasets	
Image Datasets	Train and Evaluate
Handwritten Grapheme Dataset	Recognition model
BanglaLekha-Isolated	Recognition model
A Synthetically Generated Bangla Corpus	Detection model (Train)
CMATERdb2.1.2	Detection model (Train)
BanglaWriting: A multi-purpose offline Bangla handwriting dataset	Used to create a Test dataset
Text Datasets	Train
District Names	Spelling Correction model
Bengali names vs gender dataset	Spelling Correction model
Bengali dictionary	Spelling Correction model
Prothom Alo Articles	Spelling Correction model
Collected & Edited Datasets	
300 Handwritten Words	Train Character Detection model
243 Words from **BanglaWriting** Paragraph Images	Evaluate Overall Pipeline

3.1.2 BanglaLekha-Isolated

The BanglaLekha-Isolated dataset [18] comprises 166,105 preprocessed handwritten images of 84 characters, including 50 basic Bangla characters, 10 numerals, and 24 compound characters. Annotations include the age and gender of writers to ensure diversity.

3.1.3 A synthetically generated bangla corpus

A collection of approximately 2 million Bangla words typed in various fonts and domains [19] aims to enhance OCR performance, particularly in recognizing conjunct letters.

3.1.4 CMATERdb2.1.2

The dataset contains handwritten images of 120 popular West Bengal city names [20].

3.2 BanglaWriting

The dataset is a collection of single page handwritten paragraphs of 260 individuals [21]. It contains a total 21,234 words, of them includes 5,470 unique Bangla words.

3.2.1 Text datasets

A dataset created using district names [22], over 11,000 Bangla terms [23], and Bangla names from HuggingFace [24]. This dataset is a comprehensive tool for training and developing the OCR pipeline's spelling correction tasks.

3.3 Collected data

To compensate for the scarcity and lack of variations in publicly available handwritten Bangla word image datasets, we collected a dataset comprising 300 handwritten word images.

Newspaper articles containing a total of 10,278 words were collected from Prothom Alo [25] to find the word length distribution of commonly used Bangla words. From the articles, a list of unique words was filtered out, and their lengths were analyzed, resulting in the distribution of Table 3.

This distribution was used to generate the word list for data collection. Twenty sets, each containing 15 words, were created and distributed among 20 participants. The participants were both male and female, aged between 15 and 24 years. Their educational backgrounds ranged from high school to undergraduate engineering students at universities in Bangladesh. Each participant received a unique set, with word length distributions as shown in Table 3. This approach ensured that the dataset captured a variety of handwriting styles, modifiers, and character ligatures commonly found in handwritten Bangla text. As a result, the dataset contains authentic handwritten text noises such as ink blots, smudges, and irregularities in writing style, as illustrated in Fig. 1. This guarantees that feature learning takes place in a real-world context.

3.3.1 Test dataset for evaluation of the overall model

For the evaluation of the overall pipeline, another dataset of 243 words was created, available at[26]. These words were selected and extracted from the unprocessed paragraph images of the **BanglaWriting** dataset.

Table 3 Word Length Distribution

Word Length	Frequency
1	188
2	629
3	1327
4	1773
5	1825
6	1285
7	888
8	677
9	465
10	264
11	222
12	125
13	88
14	60
15	20
16	19
17	7
18	6
19	6
20	1

*The length includes single characters as well as modifiers

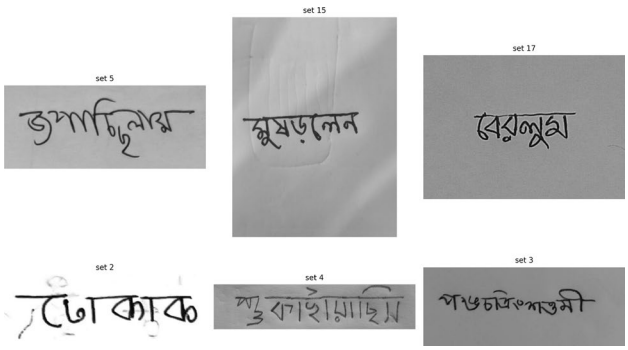


Fig. 1 Example of the Collected Dataset

3.4 Data preparation for training and testing

3.4.1 Word image data for detection model

The word image datasets underwent preprocessing to enhance quality and prepare for effective feature extraction. Noise reduction was implemented using median filtering to eliminate artifacts such as smudges, ink blots, and faint marks, ensuring cleaner inputs. Grayscale conversion removed unnecessary color information, while contrast enhancement, specifically Histogram Equalization, improved the distinction between text and background, making characters more prominent. This step was crucial for handwritten Bangla text, where overlapping, fading, or varying intensity is common due to diverse writing styles. Gaussian blur was applied to reduce residual background noise while preserving character edges, enabling accurate boundary identification. These preprocessing techniques were implemented using PyTorch and OpenCV for optimal performance.

The individual characters are then annotated in the word images using Roboflow [27], as shown in Fig. 2.

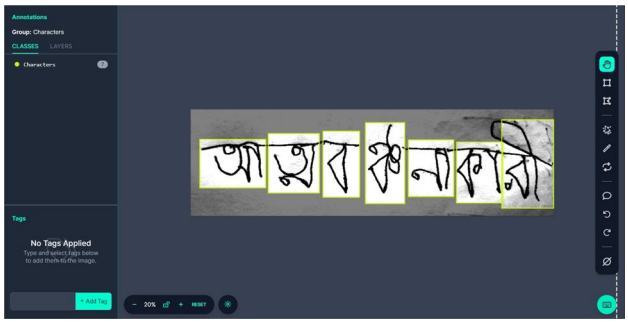
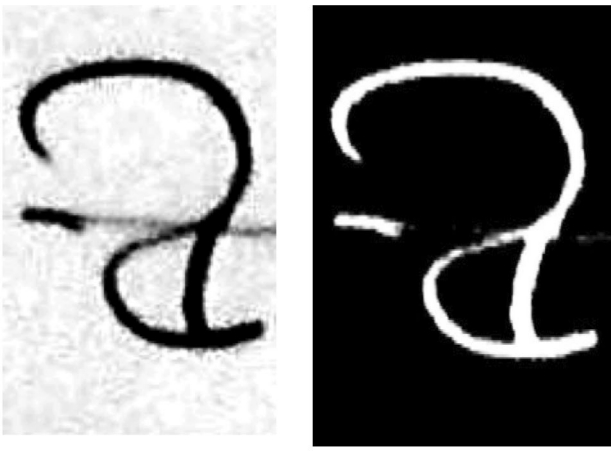


Fig. 2 Example of annotations using Roboflow



(a) Unprocessed character image (b) Processed and inverted character image

Fig. 3 Character Image data before and after processing

3.4.2 Character image data for recognition model

The dataset was created by combining the Handwritten Grapheme Dataset and the BanglaLekha-Isolated Dataset, both loaded from Parquet files. To ensure consistency, preprocessing steps were applied to enhance the data and standardize its appearance. The images were first inverted to maintain a uniform background and foreground, then resized to 128 × 256 pixels. Filtering was performed to retain only character images, excluding numerals.

Preprocessing using OpenCV included noise reduction to eliminate artifacts, histogram equalization for contrast enhancement, and standardization to ensure white characters on a black background. The images were then converted to RGB format to meet the requirements of deep learning models. Each image was paired with its corresponding grapheme root and diacritic labels, sourced from structured CSV files.

Further transformations were applied using PyTorch, including pixel normalization to a range of [-1, 1] and tensor conversion for deep learning compatibility, as shown in Figure 3. This systematic approach ensured a high-quality dataset that accurately captured the complexities of Bangla handwritten script, including grapheme roots, vowel diacritics, and consonant diacritics.

3.4.3 Text data for natural language processing

To train the spell-checker, the text file containing all Bengali words (from [22–24]) was preprocessed to clean all non-characters like, '\', ' " ', ',' and '-', to maintain consistency, and is saved as a pickle file.

4 Proposed methodology

The pipeline proposed comprises two key steps for word-level OCR: character detection and character recognition, as seen from Fig. 4. There is also an additional layer of spelling corrector for added accuracy. The pipeline uses two isolated deep learning models to separately handle the detection and recognition tasks.

4.1 Data preprocessing

The procedure starts with a word image that needs to be processed. Before the datasets are fed to machine learning models, noise reduction and enhancements are performed. Grayscale conversion is applied to remove color information, and contrast enhancement is done to make the text more prominent against the background. Finally, Gaussian blur is applied to reduce background noise and improve character visibility.

4.2 Model architecture, tuning and post-processing

4.2.1 Detection model

The suitable model for Bangla handwriting needs to be able to process the complexities of the script, which may have various modifiers, conjunct characters, and even overlapping or touching characters.

The YOLO model is a convolutional neural network (CNN) architecture designed for real-time object detection. It formulates the object detection task as a single regression problem, directly predicting bounding boxes and class probabilities from full images in one evaluation, like shown in Fig. 5.

The loss functions, Box loss, DFL(DeFocus Loss) and CLS loss(Class loss), compute the difference between the ground truth annotations and the predicted bounding boxes. The model punishes the total loss of the YOLOv8 model, which is calculated from the loss functions, to train the model to correctly detect the character bounding boxes.

4.2.2 Post-processing of images after the detection model

After passing through the detection model, bounding boxes for characters exceeding the confidence threshold (0.5) are extracted. The bounding boxes are sorted based on the x-coordinate to process characters from left to right. The steps are shown in Algorithm 1. These sorted image patches are given to the recognition model to get isolated character outputs to form the complete result.

Algorithm 1 Sort and Process Isolated Character Patches for Recognition

Input: Character bounding boxes, original image

Predict characters with confidence threshold 0.5
Create a list of sorted bounding boxes based on ascending order of x-coordinate
$width, height \Leftarrow original\_image$
$previous\_x, previous\_y \Leftarrow None$
for each character_box from sorted_list **do**
    $x, y, w, h \leftarrow character\_bounding\_box$
    if $previous\_x \neq None$ **and** x, y too close to $previous\_x, previous\_y$ **then**
        Skip to next character box
    end if
    $x_1 \leftarrow x \times \text{width} - \frac{w \times \text{width}}{2}$
    $y_1 \leftarrow y \times \text{height} - \frac{h \times \text{height}}{2}$
    $x_3 \leftarrow x \times \text{width} + \frac{w \times \text{width}}{2}$
    $y_3 \leftarrow y \times \text{height} + \frac{h \times \text{height}}{2}$
    Crop patches from original image using x_1, y_1, x_3, y_3
    Update $patches\_list$
    $previous\_x, previous\_y \leftarrow x, y$
end for

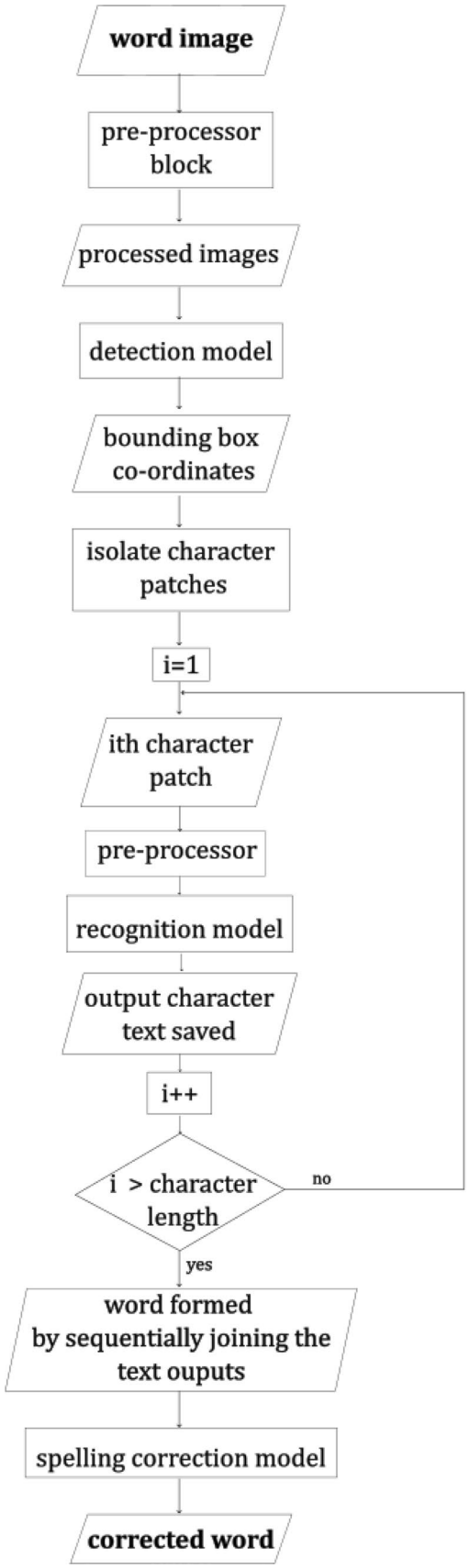


Fig. 4 Flowchart of the proposed pipeline

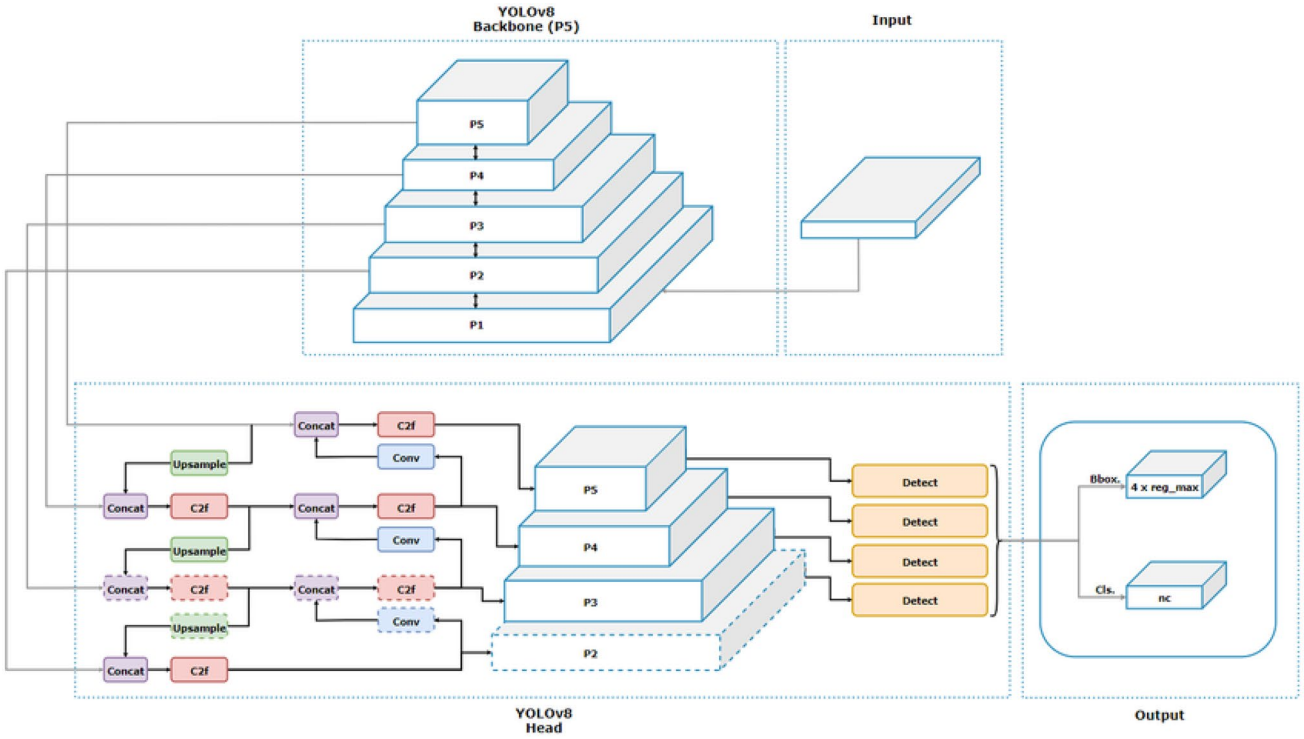


Fig. 5 The improved YOLOv8 network architecture includes an additional module for the head represented in the rectangle with a dashed outline. Source: [28]

Each images from the list of image patches is then fed into the recognition model to identify each character. The algorithm helps the repetition of a character appearing at the final result if it is erroneously detected multiple times.

4.2.3 Character Recognition Loop

The character recognition loop starts by setting an index $i = 1$, where i is the index of the current character patch. The present character patch is processed to have white written character on black background and resized (128 x 256) to be fed to a recognition model, which accepts the patch and outputs the recognized character text. After this, the system saves the identified character text and increases the index i by one. This process continues until it has gone through all the character patches.

4.2.4 Recognition model

For the character recognition task, we have chosen Convolutional Neural Networks due to its performance in recognizing character features[29]. EfficientNet makes use of a balanced scaling approach that helps improve performance while keeping computations fast. It is devised according to a constant proportion, across all dimensions of depth, width and resolution.

For our unique recognition task, we modified the EfficientNet-b4 architecture available in torchvision, shown in Fig. 6. The classifier layer is removed, a dropout layer is added before adding three fully connected layers, grapheme root, vowel diacritic and consonant diacritic. A custom loss function, as shown in Algorithm 2, that computes the average CrossEntropy loss across three outputs, is used to train the recognition model.

Algorithm 2 Custom Loss Function: loss_fc

```
function LOSS_FC(outputs, targets)
    out1, out2, out3 ← outputs
    t1, t2, t3 ← targets

    loss1 ← CrossEntropyLoss(out1, t1)
    loss2 ← CrossEntropyLoss(out2, t2)
    loss3 ← CrossEntropyLoss(out3, t3)

    return (loss1 + loss2 + loss3)/3
end function
```

This algorithm calculates the mean of 3 CrossEntropy losses that are computed for model outputs and corresponding targets (labels); it is employed in optimization of models having several outputs(in this case, three outputs for grapheme root, vowel diacritic, and consonant diacritic predictions).

4.2.5 Final word formation

Once the pipeline has gone through all the character patches, the word is formed by sequentially joining the individual recognized characters. Then the newly formed word is transformed using a spell-correcting model to elinminate recognition errors and to give the grossly spelled word the form it should be. The correct word is the final outcome, which gives the precise guide to the original word image through the recognized word and correction process.

4.2.6 Spelling correction model

An optional spelling correction model is added at the end of the pipeline which can be plugged in to increase the accuracy. Gensim's Word2Vec model can learn word embeddings by using either skip-gram or Continuous Bag of Words (CBOW), and it can also use hierarchical softmax or negative sampling. The model allows efficient saving and loading of large datasets. It also supports the "word2vec C format" in Gensim. This model displays the ability of Word2Vec to capture semantic relationships and enhance various natural language processing activities.

The Skip-gram model in Word2Vec maximizes the probability of context words given a target word, while CBOW minimizes the negative-log likelihood of predicting the target word from context words. Both models aim to capture word relationships, but they differ in their approach: Skip-gram predicts context from the target word, while CBOW does the opposite. Additionally, Skip-gram often uses negative sampling or hierarchical softmax to efficiently handle large vocabularies, whereas CBOW typically relies on context words from the sentence.

4.3 Demonstration of the inference pipeline

This section describes inference pipeline of the OCR system, which is divided into two primary stages: detection and recognition. The steps of preprocessing, detection, and character isolation are detailed in the detection part, while individual character recognition and sequential word formation are demonstrated in the recognition part.

The accompanying Fig. 7a and b, respectively illustrate the detection pipeline and the recognition pipeline in action, from the initial detection of characters within the image to the final recognition and reconstruction of the word.

The system correctly recognizes the word in this example, so it skips the spell checker.

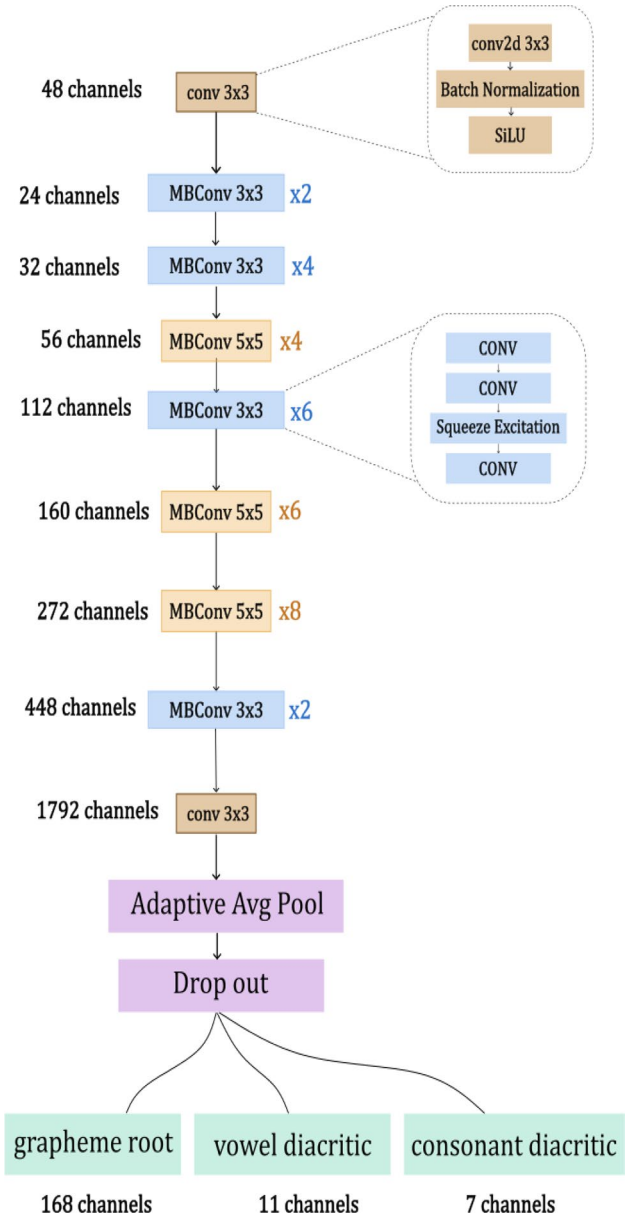


Fig. 6 The Modified Efficient-Net-b4 architecture used for Recognition Task

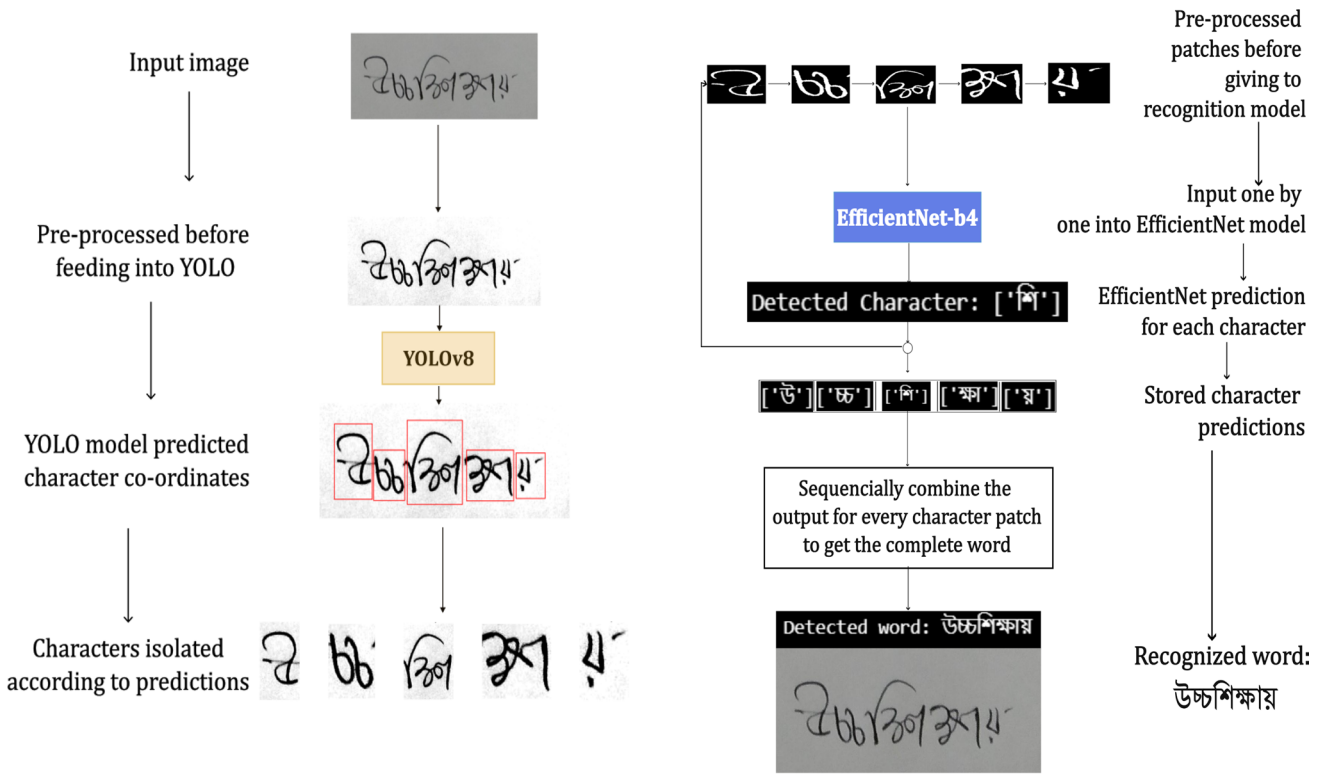
5 Experiment

5.1 Models under evaluation

5.1.1 Detection task

Considering the goal of the detection model in the pipeline, we have contemplated three different deep learning models, namely, Mask R-CNN, Character Region Awareness for Text Detection (CRAFT) and YOLO.

CRAFT is a deep learning framework designed for precise text detection by identifying individual character regions and their affinities, enabling effective segmentation in both scene text and handwritten scripts. However, Mask R-CNN is a two-stage convolutional neural network that extends Faster R-CNN to include pixel-level instance segmentation, enabling detailed object and text region delineation with high accuracy.



(a) Detection part (b) Recognition part

Fig. 7 Pipeline Inference: Detection and Recognition parts

A comparative analysis of the best three models based on computation time and scalability for large-scale datasets is shown in Table 5.

YOLOv8 offers significant advantages in terms of speed and efficiency, making it ideal for real-time applications and processing large volumes of data. In contrast, Mask R-CNN, while highly accurate due to its pixel-level segmentation capabilities, has slower processing, which limits its scalability for larger datasets. Similarly, CRAFT, although balanced in speed and accuracy, relies heavily on high-resolution inputs, increasing computational and memory requirements, which poses challenges for processing continuous scripts like Bangla. The comparison demonstrates that YOLOv8 effectively balances speed, scalability, and accuracy, making it the most suitable choice for Bangla OCR tasks requiring high-throughput processing. We tested YOLOv8 small and medium variants, selecting the one with the highest mean IoU (Intersection over Union), precision, and recall.

5.1.2 Recognition task

The deep learning models considered for the character recognition tasks are shown in Table 4. All the models were tested on the same dataset, and from analyzing their results, the model that has a good balance between loss and accuracy was chosen for the pipeline.

5.2 Experimental setup

5.2.1 Detection models

Training setup:

- The Roboflow annotated data were partitioned into training and validation datasets according to their intended use, as shown in Table 2.
- Batch size was set to 16, and training was set for 300 epochs.
- Early stopping was applied with a patience of 50 epochs to prevent over-fitting and ensure optimal performance.
- YOLOv8 models were optimized via AdamW optimizer with a learning rate of 0.002 and momentum of 0.9.

The small and medium variants of the YOLOv8 model were first trained on a synthetic word dataset to learn basic features and patterns relevant to single character detection. After initial training, the best models from each YOLOv8 variant were fine-tuned on a handwritten word dataset. Synthetic-noise augmentation techniques were employed in parallel with nonaugmented techniques during the final fine-tuning stage. Testing of the YOLOv8 models were done using the handwritten dataset used to evaluate the performance of the overall pipeline.

Evaluation Metrics:

- *Precision* Gives the probability of the detected regions being actual characters from the ratio of true positives to total predicted positives.
- *Recall* Calculates the probability of actual characters regions being detected from the ratio of true positives to total actual positives.
- *Intersection over Union (IoU)* Measures the spatial accuracy of the detected region by checking the alignment of predictions with ground truths.

Table 4 Comparison of Various Neural Network Architectures

Architecture	Conv Layers	FC Layers	Parameters	Additional Info
ResNet-18	20	1	11.7M	8 residual blocks
ResNet-34	36	1	21.8M	16 residual blocks
ResNet-50	49	1	25.6M	16 bottleneck blocks
ResNet-101	100	1	44.5M	33 bottleneck blocks
ResNet-152	151	1	60.2M	50 bottleneck blocks
GoogLeNet (Inception v1)	57	1	6.8M	9 inception modules
DenseNet-121	120	1	8.0M	4 dense blocks
DenseNet-161	160	1	28.7M	4 dense blocks
DenseNet-169	166	1	14.3M	4 dense blocks
DenseNet-201	200	1	20.0M	4 dense blocks
EfficientNet-B0	16	1	5.3M	8 MBConv blocks
EfficientNet-B1	16	1	7.8M	8 MBConv blocks
EfficientNet-B2	16	1	9.2M	8 MBConv blocks
EfficientNet-B3	16	1	12.0M	8 MBConv blocks
EfficientNet-B4	16	1	19.0M	8 MBConv blocks
EfficientNet-B5	16	1	30.0M	8 MBConv blocks
EfficientNet-B6	16	1	43.0M	8 MBConv blocks
EfficientNet-B7	16	1	66.0M	8 MBConv blocks
EfficientNetV2-S	19	1	24.2M	17 MBConv/Fused-MBConv blocks
EfficientNetV2-M	35	1	54.2M	31 MBConv/Fused-MBConv blocks

* The number of convolutional layers for DenseNet and EfficientNet models is approximate, as these models use complex block structures (e.g., dense blocks and MBConv blocks) that integrate multiple convolutional operations

Table 5 Comparison of YOLOv8, Mask R-CNN, and CRAFT for Bangla Handwritten Character Isolation

Model	Accuracy vs. Speed	Application Suitability	Suitability for Bangla Handwritten Character Isolation
YOLOv8 [30]	Fast (0.3225s of average processing time per image), moderate accuracy.	Real-time, multiple object detection, cluttered environments.	Handles varying complexity and noise effectively. Simultaneous prediction of bounding boxes and character classes. YOLOv8 is optimized for real-time detection, making it suitable for applications requiring rapid processing of large volumes of data [31].
Mask R-CNN [30]	High accuracy, slower compared to YOLOv8 (0.4867s of average processing time).	Detailed segmentation, handles complex scripts, versatile with shapes and sizes, densely packed structures.	Ideal for complex scripts and closely spaced characters.
CRAFT [32]	Balanced speed and accuracy.	Precise character detection in diverse scripts. Reliance on high-resolution input images increases memory and computational requirements.	Isolates characters in noisy or variable text. But has difficulty in segmenting characters from continuous scripts like Bangla and Arabic.

5.2.2 Recognition models

Training setup:

- The character dataset was partitioned into training, testing and validation datasets.
- Batch size varied between 64 and 128 for different models.
- Learning rate was set to 0.01 for most models, with a ReduceLROnPlateau scheduler to dynamically adjust learning rates based on validation loss.
- Adam optimizer was consistently used for efficient gradient-based optimization.
- Training epochs typically ranged from 20 to 50, depending on the model complexity and dataset size.

The recognition models are trained using the character training dataset to accurately recognize and classify handwritten characters. To ensure the effectiveness of the model, its performance is validated against the validation dataset.

Model Saving and Early Stopping

- *Best Model Saving* To mitigate the risk of overfitting and ensure that the model generalizes well to unseen data, the model state is saved if the validation loss improves.
- *Early Stopping* The training process stops early if the validation loss does not reduce for six consecutive epochs, preserving the best-performing iteration and saving computational resources during training.

The best trained model's performance is then measured using the test dataset.

Evaluation metrics

- *Precision* Measures the probability of the correct prediction of characters.
- *Recall* Measures the percentage of instances at which a character is correctly identified.
- *F1 Score* Balances precision and recall by measuring the harmonic mean.
- *Accuracy* Reflects the percentage of the correct predictions among all character prediction.

5.2.3 Spelling correction model

Training configurations

- *Skip-gram* Determines the training algorithm (1 for skip-gram, 0 for CBOW).
- *Window Size* 4 (for word vectorization).
- *Vector Size* 300 (dimensionality of the word vectors).
- *Minimum Word Count* 1 (ignores words with fewer occurrences than 1).
- *Workers* 8
- *Epochs* 100
- *Sampling* 0.01 (controls the downsampling of frequent words).

Check-pointing and Training

The training process utilizes the configured parameters.

- The model is trained on the preprocessed Bengali text data loaded from the pickle file.
- Check-pointing is controlled, allowing for the model's state to be saved after each epoch.
- After completing training, the trained Word2Vec model is saved.

The pretrained weights are exported and serialized in a pickle file containing the learned embeddings, which are ready for use in various natural language processing tasks.

5.2.4 Overall pipeline evaluation metrics

The overall correctness of the OCR system in converting handwritten text images into digital text is measured using the following metrics.

- **Precision**
- **Recall**
- **Character Error Rate (CER)**: Determines the proportion of incorrectly recognized characters to the total number of characters in a word.

5.2.5 Bottleneck Identification

To ensure that the OCR system operates efficiently, we conducted an in-depth performance analysis focusing on various time and throughput metrics.

- *Average Processing Time* Calculates the average time it takes for each part of the OCR system, detection, recognition and spelling correction, to identify a word.
- *Average I/O Waiting Time* Assess the average wait time of each part for input/output operations.
- *Average Throughput* Throughput measures the number of images processed by the part per unit time.

$$\text{Average Throughput} = \frac{\text{Number of Images Processed}}{\text{Total Processing Time}}$$

- *Average Latency* The delay from the time an image is input into the model until the final recognition result is out. It encompasses both processing and I/O delays.

5.2.6 Control Measures

To ensure consistency and reliability throughout the evaluation process, several control measures were implemented.

- *Cross-validation* Ensuring the robustness of model evaluation by splitting the dataset into multiple folds and training/testing on different combinations of these folds.
- *Uniform dataset usage* Utilizing the same datasets for training, validation, and testing across all the considered models for our pipeline to maintain fairness and comparability.

6 Results and discussion

6.1 Selection of the best performing models

6.1.1 Detection model

The performances of the small and medium YOLOv8 models trained only on handwritten word images are shown in Table 6.

The results can be further enhanced with fine-tuning. We first train the models on typed word images and then fine-tune them with handwritten word images. The improved efficacy of this method can be seen in Table 8.

Although the small model has a higher precision score, the medium model demonstrates overall better performance in terms of Recall and Mean IoU. Therefore, the YOLOv8 medium model was selected for the pipeline. Furthermore, the

Table 6 Performance of YOLOv8 Models without Fine-tuning

Model	Mean IoU	Precision	Recall
Small	0.8109	0.9085	0.9748
Medium	0.8166	0.9388	0.9607

Table 7 Impact of Synthetic Noise Augmentation

Model	Mean IoU	Precision	Recall
Medium	0.8311	0.9502	0.9626
Medium (Augmented)	0.8335	0.9528	0.9617

Table 8 Performance of YOLOv8 Models Trained on Typed Words and Fine-tuned on Handwritten Words

	Trained on Typed Word Images			Fine-Tuned on Handwritten Word Images		
Model	Mean IoU	Precision	Recall	Mean IoU	Precision	Recall
Small	0.7821	0.8690	0.8804	0.8205	0.9513	0.9673
Medium	0.7752	0.8436	0.9579	0.8220	0.9359	0.9682

impact of synthetic noise augmentation on model performance is highlighted in Table 7, showing enhancement of the model's robustness to variations in handwritten input.

6.1.2 Recognition model

Testing the deep learning models on the test dataset created from character images reflects the strengths and weaknesses of the various models.

Table 9 shows that EfficientNetb4 has the lowest average loss (= 0.1309). Among all the models tested, the EfficientNetb4 has been selected based on this criterion. In terms of accurately identifying all three labels, namely the grapheme root, vowel diacritic, and consonant diacritic, it performs satisfactorily close to the highest scoring models.

Table 10 presents the separate average accuracies for the three different labels. EfficientNetB4 demonstrates consistently high accuracy across all three recognition tasks but does not achieve the highest score in any individual category. EfficientNetV2S attains the best grapheme root accuracy at 0.9405, slightly surpassing EfficientNetB4's 0.9387. Similarly, EfficientNetV2M achieves the highest vowel diacritic accuracy at 0.9832, marginally exceeding EfficientNetB4's 0.9822. For consonant diacritic recognition, DenseNet201 performs the best with an accuracy of 0.9809, slightly higher than EfficientNetB4's 0.9804. Although EfficientNetB4 is not the top-performing model in any single category, it remains one of the most well-balanced architectures, offering a good balance between accuracy and loss, making it a robust choice for Bangla OCR tasks.

The high accuracy in detecting vowel and consonant diacritics demonstrates the OCR model's ability to handle the complexities associated with diacritic recognition. In the recognition part, compound characters are incorporated within grapheme roots, and the model exhibits a reasonable level of accuracy in processing these complex compound characters, reflected in the grapheme root recognition accuracy of 0.9387.

Comparison of our Character Recognition Model with a Similar Model

In Table 11, we compared the accuracy of the EfficientNetB4 model with that of the Borno model [33], which utilizes a similar approach to our recognition model, on a different dataset. This demonstrates that EfficientNetB4 performs competitively, achieving strong results both in individual accuracy metrics and in the overall average accuracy. It should be clear that the discrepancy in datasets could impact the comparability of the results.

6.2 Performance of the overall pipeline

The performance metrics of the OCR system shown in Table 12, both before and after correction, highlight its efficiency and accuracy.

To improve the recognition of noisy or misaligned images, we further fine-tuned the EfficientNetB4 model using augmented data. The results can be seen in Table 13.

The overall performance metrics of the recognition model, after training with augmented data, are shown in Table 14.

The results from Table 12 and Table 14 indicate that while the initial CER increased slightly due to the introduction of more challenging augmented data, the overall performance metrics postcorrection significantly improvemented.

Table 9 Loss and Accuracy Metrics

Model	avg_acc1	avg_acc2	avg_acc3	avg_loss
Resnet18	0.0175	0.1063	0.8748	0.1753
Resnet34	0.0175	0.0970	0.8841	0.1684
Resnet50	0.0145	0.0865	0.8981	0.1724
Resnet101	0.0132	0.0774	0.9083	0.1652
Resnet152	0.0130	0.0742	0.9121	0.1515
Googlenet	0.0132	0.0764	0.9093	0.1557
Densenet121	0.0121	0.0774	0.9097	0.1549
Densenet161	0.0115	0.0736	0.9141	0.1675
Densenet169	0.0118	0.0719	0.9152	0.1444
Densenet201	0.0116	0.0762	0.9115	0.1747
Efficientnetb0	0.0113	0.0795	0.9084	0.1393
Efficientnetb1	0.0129	0.0741	0.9122	0.1359
Efficientnetb2	0.0128	0.0776	0.9088	0.1518
Efficientnetb3	0.0121	0.0751	0.9120	0.1370
Efficientnetb4	0.0118	0.0732	0.9144	**0.1309**
Efficientnetb5	0.0125	0.0761	0.9105	0.1489
Efficientnetb6	0.0126	0.0766	0.9100	0.1509
Efficientnetb7	0.0129	0.0798	0.9065	0.1554
EfficientnetV2S	0.0122	0.0707	0.9165	0.1391
EfficientnetV2M	0.0117	0.0724	0.9152	0.1380

** avg_acc3, avg_acc2 and avg_acc1 are average accuracies in terms of correctly identifying all three labels, only two out of three labels and only one out of three label respectively*

Table 10 Accuracy Metrics

Model	avg_g_acc	avg_v_acc	avg_c_acc
Resnet18	0.9136	0.9755	0.9654
Resnet34	0.9163	0.9742	0.9732
Resnet50	0.9257	0.9793	0.9768
Resnet101	0.9333	0.9800	0.9796
Resnet152	0.9370	0.9808	0.9799
Googlenet	0.9364	0.9796	0.9780
Densenet121	0.9368	0.9802	0.9788
Densenet161	0.9394	0.9807	0.9809
Densenet169	0.9400	0.9824	0.9790
Densenet201	0.9376	0.9808	0.9802
Efficientnetb0	0.9341	0.9819	0.9793
Efficientnetb1	0.9361	0.9825	0.9792
Efficientnetb2	0.9336	0.9819	0.9790
Efficientnetb3	0.9370	0.9813	0.9800
Efficientnetb4	0.9387	0.9822	0.9804
Efficientnetb5	0.9344	0.9820	0.9800
Efficientnetb6	0.9347	0.9815	0.9798
Efficientnetb7	0.9338	0.9815	0.9767
EfficientnetV2S	0.9405	0.9823	0.9801
EfficientnetV2M	0.9388	0.9832	0.9801

Table 11 Comparison of Recognition Accuracy Between Our Proposed Model and the Borno Model from [Rabby, 2021] [33]

Work	Dataset Description	Accuracy
Borno Model	1,069,132 images in 207 root characters, 10 modifiers, and 6 consonant diacritic classes. This model used a different dataset from ours and shows promising results in handling large-scale datasets and complex character structures.	Root: 86.09% Modifier: 95.56% Diacritic: 93.99% Average: 91.88%
Our Proposed Method	Dataset from BanglaGrapheme and Isolated_ lekha excluding the numerical data in 168 grapheme root class, 11 vowel diacritic class, and 7 consonant diacritic class. This method integrates a more comprehensive approach to diacritic handling, leading to improved recognition accuracy.	Grapheme Root: 93.87% Vowel Diacritic: 98.22% Diacritic: 98.04% Average: 91.44%

Table 12 Overall Pipeline Metrics on In-house Data

With Spelling Correction	Average CER	Precision	Recall
No	0.0834	0.8207	0.8315
Yes	0.0274	0.9620	0.9475

Table 13 Metrics of recognition model trained on augmented data

Class	Average Accuracy	Precision	Recall	F1 Score
Grapheme Root	0.9534	0.9497	0.9469	0.9475
Vowel Diacritic	0.9863	0.9848	0.9780	0.9766
Consonant Diacritic	0.9855	0.9781	0.9752	0.9766

Table 14 Overall Pipeline Metrics with the recognition model trained on augmented data

With Spelling Correction	Average CER	Precision	Recall
No	0.1037	0.8016	0.9842
Yes	0.0247	0.9701	0.9857

This suggests that the model trained on augmented data is more robust and better at handling noisy or misaligned images, resulting in higher precision and recall rates after correction.

6.3 Sample output of handwritten text recognition pipeline

The Handwritten Text Recognition Pipeline begins with a detection stage as shown in Figure 8. In this stage, blue bounding boxes identify detected character regions, representing the system's ability to isolate individual characters for recognition. However, challenges may arise, as seen with the red bounding boxes that indicate extra detected regions, signifying unintended segmentation. These errors highlight the complexities involved in accurately identifying and isolating handwritten characters, particularly for intricate scripts, or preprocessing to have the intended number of bounding boxes.

Following detection, the pipeline progresses to the recognition and correction stage, where identified characters are processed to generate recognized words.

The output shown in Figure 9 demonstrates varying levels of accuracy, with mild recognition errors (orange boxes) displaying minor character discrepancies, while more significant errors (red boxes) arise from incorrectly detected grapheme roots. These errors reflect the challenges in recognizing complex scripts. However, a correction module

Fig. 8 Character Detection from Word Images

Image Index	Recognized Text	Corrected Word
1	জলাবায়ু	জলবায়ু
2	মাধ্যেমিক্ত	মাধ্যমিক
3	অপরিবর্তনীয়	অপরিবর্তনীয়
4	পুরোহিত	পুরোহিত
5	মুদ্রা	মুদ্রা
6	কয়েকটি	কয়েকটি
7	অজানা	অজানা
8	রাষ্ট্রীয়	রাষ্ট্রীয়
9	জীবন	জীবন
10	ক্ষমা	ক্ষমা

Fig. 9 Text Recognition and Spelling correction

then refines these outputs, as evidenced by the green-highlighted corrected words. This module effectively mitigating errors and enhancing overall recognition accuracy.

6.4 Comparison between the proposed pipeline and a state-of-the-art model

As a state-of-the-art model, we have chosen Google's Cloud Vision API [34], a robust machine learning model optimized for various text recognition tasks. This API offers a range of features for text detection and extraction, making it suitable for processing complex scripts like Bangla.

For a better performance, we added the appropriate language hint for Bengali handwriting, available from the OCR Language Support. Both Google's Vision API and our proposed hybrid model were employed to evaluate their performance in recognizing Bangla handwritten text on the same test dataset mentioned in that contain words with complex

Table 15 Performance Comparison of Text Recognition Models

Model	Total Images	Exact Matches	CER (Average)	Precision (Average)	Recall (Average)
Proposed Pipeline (without spelling correction layer)	243	162	**0.1037**	0.8016	0.9842
Proposed Pipeline (with spelling correction layer)	243	228	0.0247	0.9701	0.9857
Vision API	232	153	**0.1389**	0.8220	0.9653

compound characters and diacritics. Table 15 summarizes the results, including Character Error Rate (CER), Precision, Recall, and Exact Matches.

It can be noted that, out of the 243 images processed, the Google Cloud Vision API successfully annotated 232 images. For the remaining images, the API encountered an error: "Object of type AnnotateImageResponse is not JSON serializable." Additionally, in some instances, the API was unable to detect any content within the images, as seen from Figure 10. In contrast, the proposed pipeline effectively addresses the complexities of Bangla characters and successfully annotates all 243 word images, demonstrating a significant improvement over the Vision API with Bangla handwritten language support.

The Figure 11 compares the Character Error Rates (CER) of the proposed Hybrid Model (without spelling correction) with Google Cloud Vision's OCR model, evaluated on 243 Bangla handwritten words. Each word's CER for both models is plotted as individual points, with blue representing Google Cloud Vision's OCR model and orange indicating the Hybrid Model.

The Google Cloud Vision API demonstrates an average Character Error Rate (CER) of 0.1389, while the Hybrid Model achieves a lower average CER of 0.1037, consistently maintaining an overall CER below 0.7. In contrast, for several words, the Vision API produces a CER of 1 or higher. This reduction in CER highlights the superior accuracy of the Hybrid Model. The improved CER of the Hybrid Model suggests enhanced real-world applicability, particularly in contexts where precise character recognition is crucial.

6.5 Bottleneck in the overall pipeline

In the OCR pipeline, several bottlenecks have been identified that impact the system's overall efficiency when assessing processing time, I/O wait time, throughput and latency. As shown in Table 16, the Correction step requires

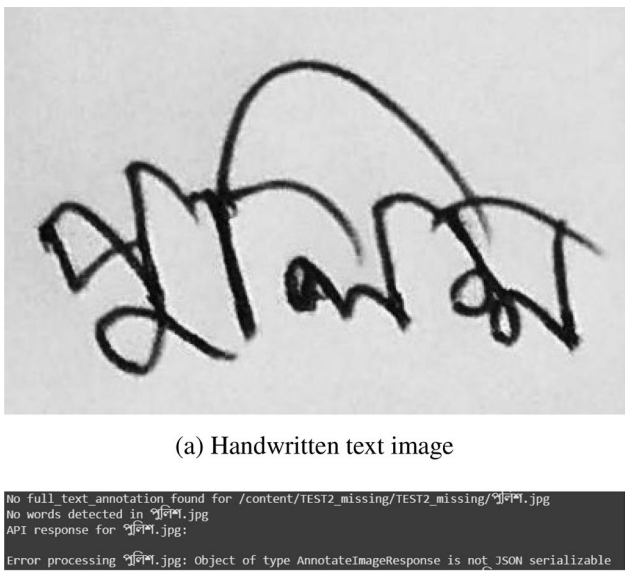


Fig. 10 Error Encountered by Vision API in Processing Some Images

(a) Handwritten text image

(b) Error message screenshot

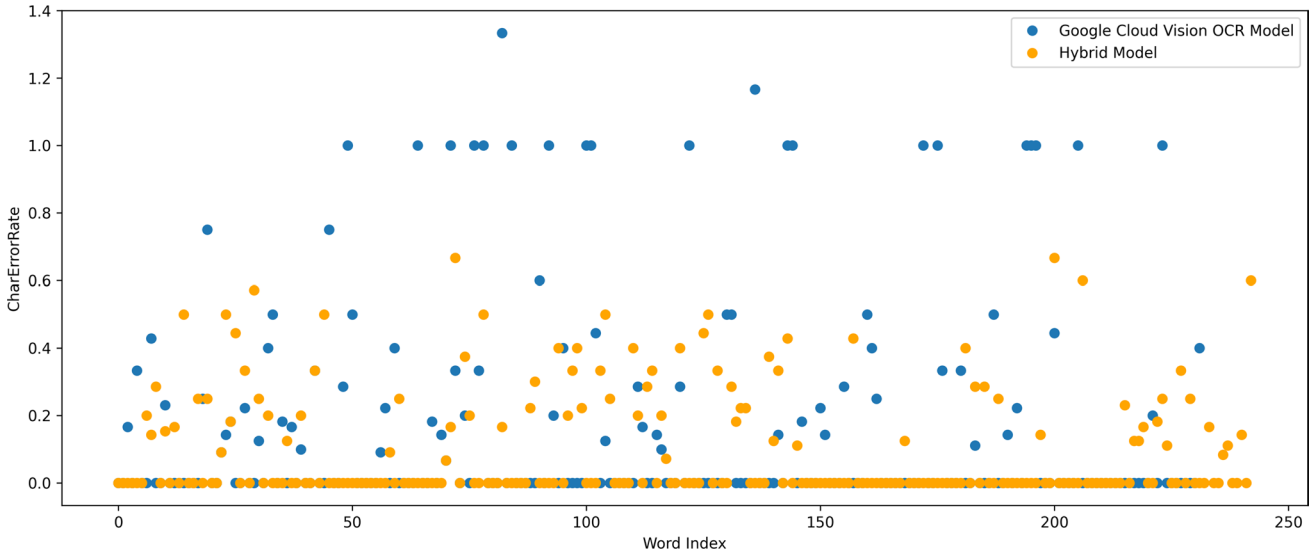


Fig. 11 Comparison of Character Error Rate (CER) for Vision API and Hybrid OCR Models Across 243 Words

Table 16 OCR Pipeline Processing Times and Bottlenecks

Metric	Average Time (seconds)	Bottleneck
Detection Time	0.0573	-
Recognition Time	0.0900	-
Correction Time	0.3663	Correction

Table 17 Detailed I/O Wait Times, Throughputs, and Latencies

Metric	Detection	Recognition	Correction	Bottleneck
Average I/O Wait Time (seconds)	0.0033	0.0167	0.0033	Recognition
Average Throughput (items/second)	64.537	41.639	695.228	Recognition
Average Latency (seconds)	0.0562	0.0919	0.3755	Correction

a substantially large amount of processing time, averaging 0.3663 s per image, compared to the Detection and Recognition models. This stage handles spelling correction and final text adjustments, which cause it to be slower due to the large vocabulary size and the high dimensionality of the word vectors.

Additionally, the recognition model proves to be a bottleneck in the overall pipeline when average I/O wait times and throughput are measured. From Table 17, it is clear that the recognition model lags significantly behind the other parts, primarily due to the large number of character images input into the model.

For a single word image input to the system, the recognition model may have to process up to 6 or 7 character images, depending on the word length. This, combined with the large model size and computational complexity of the EfficientNetB4 model, can account for its limitations.

The correction model also presents a bottleneck with a high average latency of 0.3755 s, as shown in Table 17. This is because it not only processes recognized text but also applies context-aware corrections, adding to the delay from input to final output.

However, some of the constraints can be attributed to the hardware and resources on which the system is tested. To reduce bottlenecks, optimization strategies, such as enhancing the correction model parameters, data caching, and parallel processing, particularly for the recognition phase, and the use of hardware accelerators, need to be implemented.

7 Limitations of the proposed pipeline

While the proposed pipeline demonstrates promising results in Bangla handwritten text recognition, several limitations must be acknowledged, providing avenues for future research and improvement.

- *Limited Data Augmentation* The pipeline employs standard data augmentation techniques, which may not fully capture the diverse variability in Bangla handwriting. The absence of synthetically generated handwriting data with varying noise levels restricts the model's ability to generalize to unseen and complex input data, potentially impacting performance in extreme scenarios.
- *Basic Preprocessing Techniques* The preprocessing stage does not effectively address challenges such as overlapping characters, irregular diacritics, crossed out letters, or poorly written compound characters. These inadequacies can lead to reduced recognition accuracy, especially with low-quality or highly variable handwriting samples.
- *Model Architectural Constraints* Although the pipeline incorporates advanced architectures like YOLO and EfficientNet, it lacks customizations tailored to the unique challenges of Bangla script. Small, intricate characters remain challenging, limiting the system's capacity to handle such complexities.
- *Evaluation on Limited Datasets* The pipeline has been tested on a constrained set of datasets, limiting its evaluation in diverse real-world scenarios. Broader testing on independent datasets is necessary to assess the model's generalizability and robustness.
- *Simplistic Spelling Correction Mechanism* The current spelling correction mechanism is relatively static and lacks the ability to adapt dynamically to evolving language use and context. This can result in uncorrected errors, particularly those dependent on nuanced contextual interpretations.
- *Scalability and Real-World Adaptability* The scalability of the pipeline for real-time applications remains uncertain. Factors such as computational resource demands, latency, and adaptability to diverse environments require further investigation to enable practical and widespread deployment.
- *Deployment Efficiency and Processing Speed* The pipeline has not been thoroughly evaluated for deployment efficiency and processing speed. Real-world applications often require low-latency, high-speed processing to ensure usability in practical scenarios. Further studies are needed to optimize the system for efficient deployment without compromising accuracy.

By addressing these limitations, future research can significantly enhance the robustness, adaptability, and scalability of Bangla handwritten OCR systems, advancing the field toward more practical and accurate solutions.

8 Conclusion

Our research focused on developing a deployable and precision-oriented OCR system for Bangla handwritten words. By addressing the detection and recognition components separately, we were able to independently test and select models that best address task specific challenges. The experimental results demonstrate that the proposed system effectively handles the intricacies of the writing style, correctly recognizing words that contain diacritics and ligatures. Furthermore, the performance of the end-to-end pipeline, evaluated on our collected and curated dataset, shows promising results. However, to advance Bangla Handwritten OCR further and achieve contextually sustainable text recognition, additional trials and studies are needed to address issues such as information degradation, the shift from character-based to paragraph-based OCR, and the extension to multi-script language interfaces. The limitations of character-based OCR can be addressed by integrating techniques like attention mechanisms and sequence modeling. Attention mechanisms allow the system to focus on key parts of the text, making it more robust to noise and better at handling complex scripts with modifiers and diacritics. Sequence modeling, such as transformers or RNNs, can capture the flow of text in paragraphs, enabling a smooth transition to word- or paragraph-level recognition. Specifically, the transformer encoder-decoder architecture offers significant potential for sequence learning tasks by effectively modeling long-range dependencies in text. The encoder processes the input sequence to generate a context-rich representation, while the decoder utilizes this representation to produce the output sequence, allowing the system to handle complex text structures.

Our proposed model can be expanded to handle paragraph-level OCR by detecting and recognizing individual characters within paragraphs. These characters can then be combined with appropriate spacing and alignment during postprocessing to reconstruct words and sentences. By leveraging our current framework's ability to handle character detection and recognition with high accuracy, and incorporating a spatial analysis module to maintain text structure, the model can be extended to seamlessly process entire paragraphs. This approach ensures that the contextual flow and alignment of text are preserved, making it suitable for more complex OCR tasks.

Author contributions All the authors contributed to the study's conception, pipeline design, data collection, and labeling. Data pre-processing, recognition model training and analysis, and overall pipeline evaluation were conducted by Aye Thein Maung. The detection model, spelling correction model training and analysis, and bottleneck testing were performed by Sumaiya Salekin. The first draft of the manuscript was written by Aye Thein Maung and Sumaiya Salekin. Mohammad A. Haque supervised the project, reviewed the manuscript, and provided feedback. All authors read and approved the final manuscript.

Funding No funds, grants, or other support was received.

Data availability All links to public datasets used to train and evaluate the models are available in the reference, cited in the Sect. . All the codes used are available in the git repository: https://github.com/Salekin-13/banglaWrittenWordOCR.git. The collected dataset to train the YOLO models is available at: Annotated Bangla Handwritten Word Images for Character Detection (HandwrittenWordsB), 1, ID:HandwrittenWordsB_1,URL:https://tc11.cvc.uab.es/datasets/HandwrittenWordsB_1. The dataset to test the overall performance of the pipeline is available at [26].

Declarations

Competing interests The authors have no relevant financial or non-financial interests to disclose.

References

1. University of Washington: Bengali (Bangla) | Asian Languages & Literature | University of Washington. https://asian.washington.edu/. Accessed 29 July 2024
2. Encyclopaedia Britannica: The World's 5 Most Commonly Used Writing Systems. https://www.britannica.com/list/the-worlds-5-most-commonly-used-writing-systems. Accessed 29 July 2024
3. Laws of Bangladesh: বাংলা ভাষা প্রচলন আইন, ১৯৮৭ Accessed 28 August 2024. http://bdlaws.minlaw.gov.bd/act-details-705.html
4. Raj R, Kos A. A comprehensive study of optical character recognition. In: 2022 29th International Conference on Mixed Design of Integrated Circuits and System (MIXDES), 2022;pp. 151–154 . https://doi.org/10.23919/MIXDES55591.2022.9837974
5. Das N, Acharya K, Sarkar R, Basu S, Kundu M, Nasipuri M. A benchmark image database of isolated Bangla handwritten compound characters. IJDAR. 2014;17:413–31. https://doi.org/10.1007/s10032-014-0222-y.
6. Singh A, Bacchuwar K, Bhasin A. A survey of ocr applications. Int J Machine Learn Comput (IJMLC). 2012. https://doi.org/10.7763/IJMLC.2012.V2.137.
7. Wahid MF, Shahriar MF, Sobuj MSI. A classical approach to handcrafted feature extraction techniques for bangla handwritten digit recognition. In: 2021 International Conference on Electronics, Communications and Information Technology (ICECIT), 2021;pp. 1–4 . https://doi.org/10.1109/ICECIT54077.2021.9641406
8. Alom MZ, Sidike P, Hasan M, Taha TM, Asari VK. Handwritten Bangla Character Recognition Using The State-of-Art Deep Convolutional Neural Networks 2018
9. Tounsi M, Moalla I, Pal U, Alimi AM. Arabic and latin scene text recognition by combining handcrafted and deep-learned features. Arab J Sci Eng. 2022;47:9727–40. https://doi.org/10.1007/s13369-021-06311-1.
10. Garain U, Mioulet L, Chaudhuri BB, Chatelain C, Paquet T. Unconstrained bengali handwriting recognition with recurrent models. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR), 2015;pp. 1056–1060 . https://doi.org/10.1109/ICDAR.2015.7333923
11. Farjana Yeasmin Omee, MANB. Shiam Shabbir Himel: A complete workflow for development of bangla ocr. International Journal of Computer Applications **21** (2011)

12. Akter N, Hossain S, Islam MT, Sarwar H. An algorithm for segmenting modifiers from bangla text. 2008 11th International Conference on Computer and Information Technology, 2008;177–182
13. Ahmed S, Enhancing the character segmentation accuracy of bangla ocr using bpnn. Int J Sci Res (IJSR). 2013;2:157–61.
14. Bensefia A, Paquet T, Heutte L. Grapheme based writer verification. In: Proceedings of the 11th Conference of the International Graphonomics Society (IGS'2003), Scottsdale, Arizona, 2003;pp. 274–277
15. Jung H, Ha J-Y. A structural method on grapheme segmentation of hangul characters for ocr. Future Gen Commun Network Symposia Int Conf on. 2008;3:71–4. https://doi.org/10.1109/FGCNS.2008.21.
16. Alam S, Reasat T, Sushmit AS, Siddique SM, Rahman F, Hasan M, Humayun AI. A large multi-target dataset of common bengali handwritten graphemes. Cham: Springer; 2021.
17. Roy K, Hossain MS, Saha PK, Rohan S, Ashrafi I, Rezwan IM, Rahman F, Hossain BM, Kabir A, Mohammed N. A multifaceted evaluation of representation of graphemes for practically effective bangla ocr. Int J Document Anal Recogn (IJDAR). 2023;1:23.
18. Biswas M, Islam R, Shom GK, Shopon M, Mohammed N, Momen S, Abedin A. Banglalekha-isolated: A multi-purpose comprehensive dataset of handwritten bangla isolated characters. Data in Brief. 2017;12:103–7. https://doi.org/10.1016/j.dib.2017.03.035.
19. Roy K, Hossain MS, Saha P, Rohan S, Rahman F, Ashrafi I, Rezwan IM, Hossain BMM, Kabir A, Mohammed N. Synthetic Printed Words and Test Protocols Data for Bangla OCR. Int J Document Anal Recog. 2022. https://doi.org/10.6084/m9.figshare.20186825.v1.
20. Group CR. CMATERdb: Pattern Recognition Database Repository. https://code.google.com/archive/p/cmaterdb/. Accessed: 2023-07-25.2015.
21. Mridha Dr MF. BanglaWriting: A multi-purpose offline Bangla handwriting dataset. Mendeley. 2020. https://doi.org/10.17632/R43WKVDK4W.1
22. Chowdhury A, HOSSEN MA, Baru A. BD_DB_64. Mendeley Data. 2023. https://doi.org/10.17632/zb5g5td4ns.1 . https://data.mendeley.com/datasets/zb5g5td4ns/1
23. Kamal M. Bengali Dictionary. https://github.com/MinhasKamal/BengaliDictionary. GitHub repository. 2024. https://github.com/MinhasKamal/BengaliDictionary
24. Faruk M. Bengali Names vs Gender Dataset. Hugging Face Accessed. 2024;22:2024.
25. Prothomalo: প্রথম আলো | বাংলা নউিজ পপোর Accessed 29 July 2024. https://www.prothomalo.com
26. Salekin S, Maung AT. BanglaWriting Words Dataset: A Collection of Isolated Word Images from the BanglaWriting Multi- Purpose Bangla Offline-handwriting Dataset (WoBW). https://doi.org/10.5281/zenodo.14163687 .
27. Roboflow: Sign in to Roboflow. https://app.roboflow.com/. Accessed 7 July 2024
28. Karna N, Putra MAP, Rachmawati S, Abisado M, Sampedro G. Toward Accurate Fused Deposition Modeling 3D Printer Fault Detection Using Improved YOLOv8 With Hyperparameter Optimization - Scientific Figure on ResearchGate. https://www.researchgate.net/figure/The-improved-YOLOv8-network-architecture product penalty- @M-includes-an-additional-module-for-the-head_fig2_372207753. Accessed 21 June 2024
29. Rakshit P, Chatterjee S, Halder C, Sen S, Obaidullah SM, Roy K. Comparative study on the performance of the state-of-the-art CNN models for handwritten Bangla character recognition. Multi Tools Applic. 2023;82(11):16929–50. https://doi.org/10.1007/s11042-022-13909-6..
30. Choi Y, Bae B, Hee Han T, Ahn J. Application of mask R-CNN and yolov8 algorithms for concrete crack detection. IEEE Access. 2024;12:165314–21. https://doi.org/10.1109/ACCESS.2024.3469951.
31. Turnbull R, Mannix E. Detecting and recognizing characters in Greek papyri with YOLOv8, DeiT and SimCLR. 2024. https://arxiv.org/abs/2401.12513
32. Baek Y, Lee B, Han D, Yun S, Lee H. Character Region Awareness for Text Detection (2019). https://arxiv.org/abs/1904.01941
33. Rabby ASA, Islam MM, Hasan N, Nahar J, Rahman F. Borno Bangla handwritten character recognition using a multiclass convolutional neural network. Cham: Springer; 2021.
34. Cloud G. Handwriting Recognition with the Vision API. https://cloud.google.com/vision/docs/handwriting. Accessed: 15-11-2024