

**Bachelor of Science in Electrical and Electronic Engineering
EEE 400 (July 2023): Thesis**

**Bangla Handwritten OCR: A Hybrid Deep Learning
Approach**

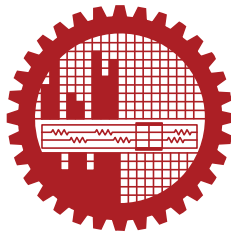
Submitted by

Sumaiya Salekin
201806191

Aye Thein Maung
201806195

Supervised by

Dr. Mohammad Ariful Haque
Professor



**Department of Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh**

June 2024

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, “Bangla Handwritten OCR: A Hybrid Deep Learning Approach”, is the outcome of the investigation and research carried out by us under the supervision of Dr. Mohammad Ariful Haque.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Sumaiya Salekin
201806191

Aye Thein Maung
201806195

CERTIFICATION

This thesis titled, “**Bangla Handwritten OCR: A Hybrid Deep Learning Approach**”, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for EEE 400: Project/Thesis course, and as the requirements for the degree B.Sc. in Electrical and Electronic Engineering in June 2024.

Group Members:

Sumaiya Salekin

Aye Thein Maung

Supervisor:

Dr. Mohammad Ariful Haque

Professor

Department of Electrical and Electronic Engineering

Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

We are deeply grateful to our supervisor, Dr. Mohammad Ariful Haque sir, for giving us the opportunity to work on this project and guiding us through the exploration of different techniques related to the project. We are thankful to our fellow supervise-es who we learned a lot of things from. Last but not the least, We are eternally indebted to our parents for encouraging us to take on newer challenges and providing us with never-ending support along the way.

Dhaka

June 2024

Sumaiya Salekin

Aye Thein Maung

Contents

<i>CANDIDATES' DECLARATION</i>	i
<i>CERTIFICATION</i>	ii
<i>ACKNOWLEDGEMENT</i>	iii
List of Figures	vii
List of Tables	viii
List of Algorithms	ix
<i>ABSTRACT</i>	x
1 Introduction	1
1.1 Motivation	1
1.2 Challenges	2
1.2.1 Complex Character Set	2
1.2.2 Variability in Handwriting	3
1.2.3 Lack of Annotated Datasets	4
1.2.4 Presence of Noise and Degradation	4
1.2.5 Computational Resource Requirements	5
1.2.6 Multilingual Considerations	6
1.3 Objectives	6
1.4 Contribution	8
1.5 Thesis Outline	10
2 Literature Review	12
2.1 Feature Extraction Methods	12
2.1.1 Handcrafted Features	12
2.1.2 Deep Learning Features	14
2.2 Segmentation Techniques and Recognition Systems	15
2.2.1 Traditional Segmentation Techniques	15
2.2.2 Advanced Segmentation Methods	16

2.3	Classifier Approaches	17
2.3.1	Traditional Classifiers	17
2.3.2	Deep Learning Classifiers	19
2.4	Integration of Deep Learning in OCR	19
2.4.1	Transfer Learning	20
2.4.2	Data Augmentation	20
2.4.3	Sequence-to-Sequence Models	20
2.5	Research Gaps and Opportunities	20
2.5.1	Data Scarcity and Quality	21
2.5.2	Model Robustness and Scalability	21
2.5.3	Computational Resources	21
3	Proposed Methodology	22
3.1	Theoretical Framework	22
3.1.1	Pipeline Example	24
3.2	System Design	26
3.2.1	Choosing Detection Model for Character Isolation	26
3.2.2	YOLO Model Architecture	28
3.2.3	Selecting Recognition Model for Identification	31
3.2.4	ResNet Model Architecture	33
3.2.5	GoogLeNet Model Architecture	34
3.2.6	DenseNet Model Architecture	36
3.2.7	EfficientNet Model Architecture	38
3.2.8	Spelling Correction Model and Architecture	40
3.3	Reasoning behind the Hybrid Model Approach	42
4	Data Pre-processing and Training	43
4.1	Datasets	43
4.1.1	Detection Model Training Data	43
4.1.2	Recognition Model Training Data	45
4.1.3	Spelling Correction Model Training Data	45
4.2	Data Preprocessing	45
4.2.1	Image Datasets	46
4.2.2	Text Dataset	47
4.3	Training Process	48
4.3.1	Detection Models	48
4.3.2	Recognition Models	49
4.4	Word2Vec Model Training Process	51
4.4.1	Data Preprocessing	52

4.4.2	Training Configuration	52
4.4.3	Checkpointing and Training	52
4.4.4	Exporting Pretrained Weights	53
4.5	Evaluation Metrics	53
4.5.1	Detection Task (YOLO Model)	53
4.5.2	Metrics for Recognition	54
4.5.3	Overall Evaluation	55
4.6	Data Collection	57
4.7	Evaluation Procedures	58
4.7.1	Detection Models	58
4.7.2	Recognition Models	58
4.7.3	Control Measures	58
5	Results and Observation	60
5.1	Demonstrating OCR System in Action	60
5.2	Performance Analysis of Detection Model	62
5.2.1	Overview	62
5.2.2	Results	63
5.2.3	Discussion	63
5.3	Performance Analysis of Recognition Model	64
5.3.1	Overview	64
5.3.2	Results	64
5.3.3	Discussion	67
5.4	Overall System Evaluation	67
5.4.1	Pipeline Integration	67
5.4.2	Results	68
5.4.3	Discussion	69
5.5	Future Works	69
	References	73
A	Codes	77
A.1	Sample Code	77
A.2	Another Sample Code	80

List of Figures

3.1	Framework for the proposed pipeline	23
3.2	Utilisation of the proposed pipeline(Detection part)	24
3.3	Utilisation of the proposed pipeline(Recognition part)	25
3.4	YOLO Architecture. Source: [1].	29
3.5	The improved YOLOv8 network architecture includes an additional module for the head represented in the rectangle with a dashed outline. Source: [2].	31
3.6	Neural network architecture of ResNet152. The dotted shortcuts increase dimensions and the layers inside the dashed box represents n duplicate convolutional layers. Source: [3].	33
3.7	GoogLeNet Architecture. Source: [4].	35
3.8	DenseNet Architecture. Source: [5].	36
3.9	EfficientNet-B0 architecture. Source: [6].	38
3.10	Word2Vec architecture. In addition to an input layer and an output layer, shallow architectures have a small number of hidden layers (one in this case). Source: [7].	40
4.1	Example of the Dataset	44
4.2	Example of annotations using Roboflow	47
5.1	System detecting character bounding boxes in word images	60
5.2	System detecting character bounding boxes in word images	61
5.3	System recognizing texts from word images	62

List of Tables

1.1	Intricacies of Bangla Handwritten Characters	3
3.1	Comparison of various deep learning models with their key architectural features.	27
3.2	Comparison of various models with their strengths and suitability for Bangla handwritten character isolation.	28
3.3	Comparison of Various Neural Network Architectures	32
4.1	Word Length Distribution ¹	44
5.1	Performance of YOLOv8 Models	63
5.2	Impact of Synthetic Noise Augmentation	63
5.3	Accuracy Metrics	64
5.4	Loss and Accuracy Metrics	65
5.5	Grapheme Metrics	65
5.6	Vowel Metrics	66
5.7	Consonant Metrics	66
5.8	Overall Pipeline Metrics	68
5.9	Metrics of recognition model trained on augmented data	68
5.10	Overall Pipeline Metrics with recognition model trained on augmented data	68
5.11	OCR Pipeline Processing Times and Bottlenecks	68
5.12	Detailed I/O Wait Times, Throughputs, and Latencies	69

List of Algorithms

1	Preprocessing with BanglaGraphemeDataset	46
2	Custom Loss Function: loss_fc	50

ABSTRACT

The digitization of handwritten documents, particularly those in complex scripts such as Bangla, presents substantial challenges. These include the intricate character set, variability in individual handwriting styles, and the scarcity of annotated datasets essential for training effective models. This thesis proposes a hybrid deep learning approach to develop an Optical Character Recognition (OCR) system tailored for Bangla handwritten text, aiming to address these multifaceted challenges.

Our methodology integrates advanced detection and recognition models to enhance the accuracy and efficiency of the OCR system. The YOLO (You Only Look Once) model is employed for character isolation due to its robustness and speed. For the recognition phase, multiple state-of-the-art deep learning architectures are explored, including ResNet, GoogLeNet, DenseNet, and EfficientNet. Each model is assessed for its performance in recognizing the unique features of Bangla characters. Additionally, a spelling correction module based on a trained Word2Vec model is incorporated to refine the output, correcting common errors and improving the overall reliability of the system.

This research makes a significant contribution to the field of OCR by presenting a scalable and efficient solution for the digitization of handwritten Bangla documents. The hybrid deep learning approach not only addresses the specific challenges associated with the Bangla script but also offers a framework that can be adapted for other complex scripts and multilingual document digitization. Potential applications of this work include the preservation of historical documents, improved accessibility for digitally archiving handwritten materials, and support for educational and research activities involving Bangla text.

In conclusion, the hybrid deep learning approach proposed in this thesis demonstrates a promising advancement in the OCR of handwritten Bangla text. By combining robust detection techniques with sophisticated recognition models and an effective spelling correction mechanism, this research paves the way for more accurate and reliable digitization of handwritten documents in Bangla, contributing to the broader efforts of preserving and accessing valuable textual information in the digital age.

Chapter 1

Introduction

1.1 Motivation

In recent years, with the advancements of digital devices, widespread availability of data has become a crucial part of expanding our collective knowledge. Data digitization is the first pivotal step in ensuring ease of access to data, and Optical Character Recognition (OCR) is a major tool in the process [8]. It yields an agile workflow by providing ways for image-to-text data conversion, error reduction, data manipulation and extraction in digital copies, and much more. It may serve as a stepping-stone for broader applications such as line OCR, paragraph-to-text conversion, multilingual OCR, etc. However, while considerable progress has been made in OCR technology, challenges persist, particularly in handling handwritten scripts from languages with complex character structures.

Despite the growing significance of OCR technology, there exists a paucity of annotated datasets for Bangla, hindering advancements in this domain. Moreover, the intrinsic complexities of Bangla script, characterized by a plethora of modifiers and compound characters, further exacerbate the difficulty of accurate recognition. Apart from the vowel characters, in most words, the vowels take on modified shape, called modifiers. Some of the modifiers may extend in the upper and lower zone of the words. There are approximately 168 unique compound characters present, composed of 2, 3 or 4 consonants. Moreover, some characters do not contain the ‘Matra’; the upper horizontal line that connects the characters in a word. Some Bangla characters have more than one form in handwritten text. All the complexities of the characters and the words create a more challenging dataset to train an OCR model.

The enhancement of Bangla Handwritten OCR has significant potential to impact the digitalization and quality of life in South Asia [9]. Key areas that stand to benefit include linguistics research and handwriting analysis, digitization of historical and legal

documents, cultural preservation and analysis, and healthcare record management. These advancements will contribute to overall regional development.

While existing approaches to handwritten word recognition, such as BLSTM-CTC models and conventional segmentation methods [10], have achieved notable progress, they struggle with the complexities of Bangla script. The difficulties in recognizing modifiers and compound characters underscore the necessity for innovative solutions tailored to Bangla's unique features. This research addresses these limitations by investigating hybrid deep learning-based methods, which separate character detection and recognition steps, aiming for enhanced accuracy and performance.













1.2 Challenges

Developing an OCR system for Bangla handwritten text presents a unique set of challenges due to the intricate and complex nature of the Bangla script. Among the multitude of complexities, some key challenges are addressed in this study.

1.2.1 Complex Character Set

Due to the large and diverse characters present in the Bengali writing system, some of which can be observed from Table 1.1, it is difficult to create a multiclass dataset for character recognition models which. 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 (matras). 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 (ṣa) becomes ক্ষ (kṣa). There also exists hasanta for suppressing the inherent vowel in a consonant, often leading to the formation of conjuncts. But the main difficulty when designing an OCR arises from the non-linearity 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 difficulty for the OCR model to distinguish them from the base character, example, চাঁদ (chānd) - The nasalization mark "ঁ" is placed above the first character "চ". Though the absence of uppercase and lowercase letters offers some simplifications, it makes it difficult to identify proper nouns.

Table 1.1: 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		

1.2.2 Variability in Handwriting

Handwritten Bangla text exhibits significant variability due to differences in individual handwriting styles. In broad aspects, the writing variations can be observed into following categories:

- **Character Formation:** Individuality in writing can be seen primarily from the stroke order and shape. Some may use a more rounded stroke, while others may write in a more angular style. Also, the size and proportions of the characters vary from writer to writer; from larger, more spaced-out characters to smaller, more compact ones. Finally, the same character may look different each time it is

written by the same writer.

- **Ligatures and Conjuncts:** Writers may use simplified forms of conjuncts, or they may separate them into their base consonants. The commonly faced problem with conjuncts is their legibility in the handwriting of individuals.
- **Diacritical Marks:** Another vastly recognized complexity is from the precision in the placement of the diacritic marks, specially in the positioning of chandrabinu (ঁ). Some place it slightly off from the character it was meant to be above.
- **Line Alignment:** While writing, some people might maintain a consistent baseline, whereas others may have characters that vary above or below an imaginary line. Furthermore, the angle or slant of the text can vary from vertical to a slight slant to the left or right.
- **Spacing and Personal Style:** Inter-word spacing can depend on the writer’s personal style which can impact the overall flow and readability of the text. Moreover, individualistic styles, flourishes such as, loops, curls, or other decorative elements can hugely affect the text’s readability. Lastly, simplification of characters for speed, especially in cursive may negatively impact its legibility.

1.2.3 Lack of Annotated Datasets

A major obstacle in the development of Bangla OCR systems is the scarcity of large, annotated datasets. High-quality, annotated datasets are crucial for training and validating deep learning models. Most datasets focus on the isolated characters, such as BanglaLekha-Isolated (Biswas et al., 2017) [11], EkushNet (Rabby et al., 2018) [12] and Borno (Rabby et al., 2021) [13]. But there is a lack of large, annotated datasets for handwritten words, sentences, or paragraphs, that fully represent the complexities of Bangla writing system. The paucity of such datasets for Bangla handwritten text hinders the ability to develop robust and accurate OCR systems. Some advanced OCR studies have been done on synthetic typed words [14], but these models perform poorly when it comes to analyzing handwritten words.

1.2.4 Presence of Noise and Degradation

OCR noise significantly degrades the performance of event detection models, with varying effects depending on the type of noise introduced. From the studies of this paper [15], character degradation and blur are particularly detrimental, causing significant drops in F1 scores for event identification and classification. This degradation results in higher

error rates and a substantial proportion of event triggers being misrecognized. The degradation caused by OCR noise leads to increased error rates in text recognition, significantly affecting the accuracy of event detection systems.

Specifically, a lot of event triggers, keywords or phrases critical for identifying and categorizing events, are misrecognized due to noise and character degradation. This misrecognition not only reduces the overall precision and recall of event detection models but also compromises the reliability of the extracted information. Consequently, the ability of these systems to accurately detect and classify events from digitized texts is substantially hindered, emphasizing the need for improved OCR algorithms and preprocessing techniques to handle noisy inputs effectively.

The study emphasizes the performance of OCR models in the presence of character degradation, bleed-through, blur, and phantom characters. These imperfections can significantly impact the performance of OCR systems, making it difficult to accurately detect and recognize characters.

1.2.5 Computational Resource Requirements

Training an OCR (Optical Character Recognition) model necessitates substantial computational resources, including multiple high-memory GPUs (16GB or more per GPU) for parallel processing, multi-core CPUs for efficient data preprocessing, at least 32GB of RAM for handling large datasets and augmentations, and high-capacity SSDs for faster data access.

The software stack typically includes deep learning frameworks like TensorFlow, PyTorch, or Keras, along with data processing libraries such as OpenCV and PIL. A 64-bit operating system, preferably Linux, is also essential. Large, diverse, and high-quality datasets, ranging from tens of thousands to millions of labeled images, are crucial for training robust models.

The training process requires numerous epochs, large batch sizes, and complex network architectures, demanding considerable computational power and memory. Hyperparameter tuning through methods like grid search or Bayesian optimization further adds to the computational load. Additional considerations include energy consumption, adequate cooling, physical space, and network bandwidth, especially when using cloud-based resources.

Balancing these requirements ensures efficient training and high-performance OCR models. But the access to such resources can be a limiting factor, particularly in resource-constrained environments.

1.2.6 Multilingual Considerations

Optical Character Recognition (OCR) for Bengali (Bangla) faces significant complexities in multilingual contexts. Bengali script often appears alongside other languages like English, requiring OCR systems to distinguish and accurately interpret different scripts within the same document. This necessitates robust multilingual models capable of handling diverse character sets and linguistic conventions.

Additionally, OCR systems must accommodate variations in font styles and handwritten text across languages, ensuring accurate transcription and recognition. Managing these multilingual complexities demands sophisticated algorithms and comprehensive training datasets tailored to diverse linguistic environments, enhancing the accuracy and reliability of Bengali OCR applications in mixed-language contexts.

Addressing these challenges requires a thoughtful and innovative approach, leveraging advanced techniques in deep learning, data augmentation, and model optimization. The subsequent sections of this thesis will delve into the methodologies and solutions proposed to overcome these obstacles and enhance the performance of Bangla handwritten OCR systems.

1.3 Objectives

The primary objective of this research is to develop and validate a robust hybrid deep learning-based Optical Character Recognition (OCR) system for Bangla handwritten text. The proposed pipeline aims to address the various complexities of Bangla handwriting system, and the lack of extensive annotated datasets. The specific objectives of the research are:

- **Develop a Hybrid OCR Model:** The hybrid OCR model separates the character detection and recognition tasks and employs two different models to accomplish them, a YOLO detection component with an EfficientNet recognition component. This model aims to leverage the strengths of both techniques to achieve high accuracy and efficiency in recognizing Bangla handwritten text. For increased accuracy, a spell checker model is also deployed at the end of the pipeline.
- **Enhance Text Detection Accuracy:** In dealing with complex backgrounds, different font styles, and low-quality images, isolating the individual characters can increase the accuracy of text detection. Isolating the characters reduces the complexity of the images and simplifies the problem. As there is a greater availability of isolated character-based datasets, it is easier to train a character detection model

that accounts for noise and degradation in single character images than it is to train a superior word or sentence recognition model.

Moreover, isolated characters have less variations than complete words, helping to streamline the model's learning and generalization. It is also easier to locate and troubleshoot errors if the problem is broken down into multiple steps. Furthermore, misclassification of one character is less consequential than an error occurring in the sequence generation.

- **Optimize Character Recognition:** Isolating the single character images will help train the recognition model to target features unique to each character, improving overall recognition accuracy. EfficientNet models are well-suited for Bangla character detection due to their advanced feature extraction capabilities, scalability, and efficiency. Bangla script includes intricate characters and diacritical marks, which EfficientNet's architecture can capture effectively. The scalability of EfficientNet, with models ranging from B0 to B7, allows for balancing accuracy and computational efficiency based on specific needs. Leveraging pre-trained EfficientNet models through transfer learning accelerates training and enhances accuracy for Bangla characters. Additionally, EfficientNet's design ensures high performance with optimized resource usage, making it suitable for various hardware platforms. The models' ability to handle large and diverse character sets and robustness against variability in handwriting styles further contributes to their effectiveness in Bangla character recognition. Furthermore, the EfficientNet V2 version introduces enhancements in architecture that offer even better accuracy and efficiency, making it a powerful choice for handling the complexities of Bangla handwritten script. EfficientNet V2's improvements in training speed and parameter efficiency provide additional advantages, ensuring high performance while minimizing computational overhead. The character recognition process is optimized by training the EfficientNet model to handle the specific complexities of Bangla script, including modifiers, compound characters, and varied handwriting forms.
- **Data Preprocessing and Augmentation:** Data preprocessing and augmentation are essential for training robust OCR models. Preprocessing tasks like normalization, resizing, and noise reduction ensure uniformity and clarity in input images, optimizing them for accurate character detection. Normalization standardizes brightness and contrast levels, while resizing standardizes dimensions for consistent model input. Noise reduction techniques enhance image quality, especially beneficial for scanned documents or images with artifacts. Augmentation techniques such as rotation, scaling, flipping, and adding noise diversify the dataset, improving model generalization and resilience to variations in text appearance and environmental conditions. Together, these preprocessing and augmentation steps enable

OCR models to achieve higher accuracy and reliability in recognizing characters across different contexts and image qualities.

- **Evaluate Model Performance:** Evaluation techniques for OCR models are crucial in assessing their accuracy and effectiveness. Common metrics include Character Error Rate (CER), and accuracy, which measures the percentage of correctly recognized characters; precision and recall, which evaluate the model's ability to correctly identify relevant characters and avoid false positives; and F1-score, which balances precision and recall. These metrics are supplemented by confusion matrices to analyze specific errors and identify patterns in misclassifications, helping to refine the pipeline through targeted improvements in preprocessing, training data quality, or model architecture.
- **Contribute to Bangla OCR Resources:** The collection and annotation of our own handwritten Bangla words represent a significant contribution to the scant Bangla OCR resources. By curating this dataset, we enrich the availability of diverse and authentic handwritten samples, essential for training and validating OCR models specifically tailored to Bangla script. This effort not only enhances the accuracy and reliability of our OCR system but also fosters advancements in character recognition technology for Bangla, addressing the unique nuances and variability inherent in handwritten texts.
- **Investigate Ethical Considerations:** Ensure that the development and deployment of the OCR system uphold ethical standards, especially regarding data privacy and usage. Implement protocols to responsibly manage sensitive information and secure necessary permissions for data utilization. This approach safeguards user confidentiality and builds trust in the system's ethical operation.

Through achieving these objectives, this research aims to make significant strides in the field of Bangla handwritten OCR, providing a foundation for further technological advancements and practical applications in various domains.

1.4 Contribution

This research makes several significant contributions to the field of Optical Character Recognition (OCR), specifically focusing on the complexities of Bangla handwritten text. The key contributions of this thesis are as follows:

- **Development of a Hybrid OCR Model:** This research presents a novel hybrid OCR model that combines YOLO for text detection with EfficientNet for character

recognition. The integration of these two powerful models aims to address the unique challenges of Bangla script recognition, resulting in improved accuracy and efficiency over traditional OCR methods.

- **Enhanced Text Detection and Recognition Techniques:** The study introduces advanced techniques for the detection and recognition of Bangla handwritten text, focusing on overcoming the intricacies of modifiers, compound characters, and diverse handwriting styles. These techniques contribute to the development of more robust and reliable OCR systems.
- **Creation of Preprocessing and Augmentation Pipelines:** A comprehensive data preprocessing and augmentation pipeline is developed to enhance the quality and variability of the training dataset. These steps, including image resizing, noise removal, normalization, and segmentation, are critical for preparing handwritten Bangla text for effective OCR.
- **Annotated Datasets for Bangla Handwritten Text:** This research contributes to the OCR community by providing annotated datasets specifically curated for Bangla handwritten text recognition. These datasets address the current paucity of resources and facilitate further research and development in this area.
- **Evaluation Framework for Bangla OCR:** An evaluation framework is established, using standard metrics such as Character Error Rate (CER), Precision, Recall, and F1 score, to rigorously assess the performance of the hybrid OCR model. This framework ensures a thorough and systematic evaluation of the model's effectiveness.
- **Practical Applications and Impact:** The developed OCR system has significant potential for practical applications in linguistics research, handwriting analysis, digitization of historical and legal documents, cultural preservation, and healthcare record management. By enhancing Bangla OCR capabilities, this research contributes to the digitalization and overall development in South Asia.
- **Open-Source Code and Methodologies:** To support the wider research community, the code and methodologies developed in this research are made available as open-source resources. This contribution promotes transparency, reproducibility, and collaborative advancements in the field of Bangla OCR.

By addressing the unique challenges of Bangla handwritten text recognition and providing valuable resources and methodologies, this research significantly advances the state of OCR technology and opens new avenues for future exploration and application.

1.5 Thesis Outline

This thesis is structured to provide a comprehensive exploration of the development and evaluation of a hybrid deep learning model for Bangla handwritten optical character recognition (OCR). The preceding outline of the thesis is as follows:

Chapter 2: Literature Review

- **Feature Extraction Methods:** Examines existing methods for feature extraction, including handcrafted and deep learning-based features.
- **Segmentation Techniques and Recognition Systems:** Reviews traditional and advanced segmentation techniques along with various recognition systems.
- **Classifier Approaches:** Compares traditional classifiers with deep learning classifiers.
- **Integration of Deep Learning in OCR:** Discusses the role of transfer learning, data augmentation, and sequence-to-sequence models in OCR.

Chapter 3: Proposed Methodology

- **Theoretical Framework and System Design:** Describes the theoretical basis and design of the proposed hybrid OCR system.
- **Choosing Detection Model for Character Isolation:** Details the selection process for the YOLO model architecture.
- **Selecting Recognition Model for Identification:** Evaluates different recognition models including ResNet, GoogLeNet, DenseNet, and EfficientNet.
- **Spelling Correction Model and Architecture:** Explains the design and integration of the spelling correction model.

Chapter 4: Data Processing and Training

- **Datasets:** Provides information on the datasets used for training detection, recognition, and spelling correction models.
- **Data Preprocessing:** Describes the preprocessing steps for both image and text datasets.
- **Training Process:** Outlines the training procedures for detection, recognition, and Word2Vec models, including configuration and check-pointing.

- **Evaluation Metrics:** Defines the metrics used for evaluating detection tasks and recognition accuracy.
- **Data Collection and Evaluation Procedures:** Discusses the methods used for data collection and the procedures followed for evaluating the models.

Chapter 5: Results and Observations

- **Performance Analysis of Detection Model:** Presents the results, discussions, and performance analysis of the YOLO-based detection model.
- **Performance Analysis of Recognition Model:** Provides an overview, results, and discussions on the recognition model's performance.
- **Overall System Evaluation:** Integrates the entire pipeline, presenting overall results and discussions.

Future Works which includes identifying potential improvements, addressing data scarcity, model robustness, scalability, and computational resource challenges. References

Comprehensive list of all the references cited throughout the thesis and appendices

Provides sample codes used in the research for replication and further study.

Chapter 2

Literature Review

Optical Character Recognition (OCR) for Bangla handwritten script presents unique challenges due to the language’s complex orthography and diverse writing styles. Over the years, various approaches have been developed to tackle these challenges, focusing on feature extraction, segmentation techniques, and classifier strategies. This section highlights their methodologies, achievements, and limitations.

2.1 Feature Extraction Methods

Feature extraction is a pivotal step in OCR systems, particularly for Bangla handwritten characters due to the script’s intricate and overlapping nature. Effective feature extraction methods transform raw input data into meaningful representations, significantly enhancing OCR performance. These methods can be broadly categorized into handcrafted features and deep learning-based features.

2.1.1 Handcrafted Features

Handcrafted feature extraction involves manually designing algorithms to identify and extract relevant features from the input data. These features are typically based on domain knowledge and specific characteristics of the script. A notable study on Bangla handwritten digit recognition by Saha et al. (2023) [16] benchmarks four classifiers—K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Random Forest (RF), and Gradient-Boosted Decision Trees (GBDT)—against three handcrafted feature extraction techniques: Histogram of Oriented Gradients (HOG), Local Binary Pattern (LBP), and Gabor filters. Notable handcrafted feature extraction techniques described:

- **Histogram of Oriented Gradients (HOG):** HOG captures edge or gradient structure by computing the gradient orientation histograms in localized portions of an image. It is effective for capturing shape and structure, making it useful for distinguishing between different Bangla characters despite their complex forms. The study identifies HOG combined with SVM (HOG+SVM) as the most effective configuration, achieving recognition accuracies of 93.32%, 98.08%, 95.68%, and 89.68% on the NumtaDB, CMARtdb, Ekush, and BDRW datasets respectively.
- **Local Binary Pattern (LBP):** LBP encodes the texture information of an image by thresholding the neighborhood of each pixel and considering the result as a binary number. It is particularly useful for capturing texture variations, which can differentiate Bangla characters with similar shapes but different textures. While LBP is effective in capturing local texture information, LBP can be sensitive to noise and variations in handwritten text. A paper by Lin et al. (2018) [17] demonstrate that LBPNet achieved competitive performance with significantly reduced model size and improved speed. For instance, on the MNIST dataset, LBPNet(RDP) attained a classification error rate of 0.50% with a model size of 1.59KB and a speedup of 1188.7 times compared to the baseline CNN's error rate of 0.29%, model size of 7.00MB, and speedup of 1X. On the SVHN dataset, LBPNet(1x1) achieved an error rate of 8.33%, whereas the baseline CNN had a lower error rate of 2.53%. For CIFAR-10, LBPNet(1x1) reached an error rate of 22.94%, significantly higher than the baseline CNN's 8.39%.
- **Gabor Filters:** Gabor filters are used for texture analysis and pattern recognition, capturing spatial frequency, orientation, and texture. Although they are useful in detection of specific orientations and patterns in Bangla characters, aiding in the differentiation of intricate and overlapping structures, they may require fine-tuning of filter parameters to achieve optimal performance. The experimental results from the paper by Ramanathan et al. (2009) [18] show that the proposed Local Binary Pattern Network (LBPNet) achieves competitive performance across three datasets. For MNIST, LBPNet with 1x1 convolution (LBPNet(1x1)) achieved a classification error rate of 0.51%, while the version with random projection (LBPNet(RDP)) achieved 0.50%. On the SVHN dataset, LBPNet(1x1) reached an error rate of 4.54%, and LBPNet(RDP) achieved 4.37%. For CIFAR-10, LBPNet(1x1) had an error rate of 15.79%, and LBPNet(RDP) achieved 16.73%. These results indicate that LBPNet can efficiently reduce model size and computational latency while maintaining accuracy comparable to conventional convolutional neural networks (CNNs).

The authors of the study faced significant challenges due to the structural similarities and intricate shapes of Bangla digits, which complicated both feature extraction and classification. They successfully addressed these issues through meticulous classifier tuning and found that handcrafted features greatly improved performance. But handcrafted features, although effective, have limitations in handling diverse and noisy datasets. They require significant domain expertise and may not generalize well to varied handwriting styles and environmental conditions.

So, the approach's reliance on robust feature extraction and potential limitations with more complex or noisy datasets suggest that future research might benefit from incorporating deep learning techniques for more comprehensive feature representation and extraction. Handcrafted features, while effective in controlled environments, often fail to generalize well to diverse and noisy real-world data.

2.1.2 Deep Learning Features

Deep learning models, particularly Convolutional Neural Networks (CNNs), have revolutionized feature extraction by learning hierarchical features directly from raw images. This reduces the dependency on manual feature engineering and allows for the automatic extraction of robust and comprehensive feature representations.

- **Convolutional Neural Networks (CNNs):** Deep learning models, especially Convolutional Neural Networks (CNNs), have shown great promise in automatically learning hierarchical features directly from raw images, reducing the dependency on manual feature engineering. CNNs consist of multiple layers of convolutional and pooling operations, which automatically learn to detect relevant features such as edges, textures, and complex patterns. These models can learn to extract relevant features through multiple layers of convolutional and pooling operations, capturing intricate patterns and structures in handwritten Bangla text. Studies leveraging CNNs for feature extraction have reported significant improvements in recognition accuracy. For example, a study by Alam et al. (2017) [19] demonstrated the effectiveness of using deep CNNs for Bangla handwritten character recognition, achieving state-of-the-art results on several benchmark datasets. The ability of CNNs to learn robust feature representations has been a major factor in their success in OCR applications.
- **Hybrid Approaches:** In some cases, combining handcrafted features with deep learning models can leverage the strengths of both approaches. For instance, features extracted using HOG or LBP can be fed into CNNs or other deep learning

classifiers. This enhances the robustness and accuracy of OCR systems, particularly in scenarios where deep learning models might struggle with limited data or require additional contextual information from handcrafted features. For example, a study titled "Arabic and Latin Scene Text Recognition by Combining Handcrafted and Deep-Learned Features" explores combining handcrafted features with deep learning methods to enhance OCR systems [20]. This approach integrates features extracted using traditional methods like HOG and SIFT with deep learning models such as Convolutional Neural Networks (CNNs) and Deep Sparse Auto-Encoders (SAEs). The combination aims to improve recognition accuracy by leveraging the complementary strengths of both techniques

Studies like these suggest that integrating handcrafted features with deep learning models can significantly improve OCR performance, especially in complex or noisy environments where deep learning models alone might not suffice.

The success of deep learning in OCR applications is largely due to its ability to automatically learn and adapt to the complex patterns and variations inherent in handwritten Bangla text. This adaptability makes deep learning a promising direction for future research, aiming to further improve recognition accuracy and generalization across diverse datasets and real-world conditions.

2.2 Segmentation Techniques and Recognition Systems

Segmentation is another critical aspect of OCR systems, especially for scripts like Bangla with complex characters and varying document types. Effective segmentation ensures that the input image is divided into meaningful regions, which can then be processed by recognition models.

2.2.1 Traditional Segmentation Techniques

Traditional segmentation techniques in Optical Character Recognition (OCR) are fundamental processes that involve dividing an input image into meaningful regions for further processing. These techniques have been instrumental in early OCR systems but often face challenges with complex scripts such as Bangla, which features intricate characters and variable document types. The paper by Sonkusare et al. (2021) [21] discusses various segmentation methods used in the segmentation of Devanagari scripts:

- **Connected Component Analysis (CCA)** is one of the cornerstone methods in traditional segmentation. It identifies and labels all connected regions in a binary image. Each region is a set of pixels connected by a given connectivity criterion (e.g., 4-connectivity or 8-connectivity in a 2D grid). This approach is effective for printed text where characters are usually well-separated. It isolates characters or symbols by finding contiguous areas of similar pixel values. However, it struggles with handwritten or closely spaced text, where characters might overlap or touch, leading to segmentation errors and reduced recognition accuracy. In Bangla, for instance, the script's propensity for character ligatures and complex shapes can cause connected component analysis to fail in distinguishing individual characters.
- **Morphological Operations** involve applying transformations to the image to enhance the segmentation process. These operations involve the manipulation of the shape and structure of features in an image, primarily using two basic operators: dilation and erosion. Morphological operations help in enhancing or suppressing specific structures in the image, which can be then used to separate text from the background and from other text elements. Morphological operations can be particularly useful for removing noise and isolating characters in noisy images. They also help in separating characters that may be touching or connected. However, they can sometimes distort the shapes of the characters themselves, which is a critical drawback for languages like Bangla where precise character shapes are crucial for accurate recognition.
- **Projection Profiles** are another traditional technique that involves projecting the pixel values of an image along the horizontal or vertical axis to create "profiles". These profiles are then analyzed to find peaks and valleys, which correspond to the lines and spaces in the text. This method works well for printed text with regular spacing, as they detect boundaries between lines of text by identifying the white spaces but can struggle with skewed or slanted handwritten text. In Bangla OCR, where text lines may not be perfectly horizontal due to the script's natural handwriting style, projection profiles can often result in incorrect line and character segmentation.

2.2.2 Advanced Segmentation Methods

In response to the limitations of traditional methods, advanced segmentation techniques have been developed to handle the complexity and variability of scripts like Bangla. The paper by Ahmed (2014) describes [22] various advanced segmentation methods that enhance the accuracy and efficiency of image segmentation processes. Among these, the

Template Matching technique is highlighted for its ability to identify and match patterns within an image, thereby facilitating precise segmentations based on predefined templates. The Mean Shift Iterative Algorithm is another significant method, which iteratively shifts data points towards the mode of a density function, leading to the identification of high-density regions that represent coherent segments within the image. Additionally, the paper discusses the Constrained Compound Markov Random Field (CCMRF) model, which integrates spatial context to improve the segmentation of complex scenes by modeling the dependencies between pixels. The Statistical Pattern Recognition (SPR) approach is also mentioned, focusing on the statistical properties of image regions to differentiate between various segments based on their statistical distributions. These advanced methods collectively represent a significant advancement in the field of image segmentation, offering robust and scalable solutions for segmenting complex images.

One of the key components of any system with a robust segmentation method is a self-attentional VGG-based multi-headed model. This advanced segmentation method combines convolutional neural networks (CNNs) with self-attention mechanisms [21]. The VGG-based model provides the backbone for feature extraction, while the multi-headed attention captures the dependencies among different characters or components in the text, focusing on different parts of the text simultaneously, improving its ability to accurately segment and recognize characters in varying contexts. This model leverages the power of attention mechanisms to capture the intricate dependencies among Bangla characters, which are often context-dependent and connected in complex ways.

Overall, while traditional segmentation techniques provide a solid foundation, the complexity of Bangla script necessitates the use of advanced methods to achieve reliable and accurate OCR results.

2.3 Classifier Approaches

Classifiers play a pivotal role in the OCR process by categorizing characters based on extracted features. Various classifier approaches have been explored in the context of Bangla OCR, each with its own strengths and limitations.

2.3.1 Traditional Classifiers

Traditional classifier approaches have laid the foundation for OCR technology, providing critical insights and methodologies that continue to influence modern techniques.

- **Nearest Neighbour (NN) Classifier:** The Nearest Neighbour classifier is one of the simplest and most intuitive classification methods [23]. It operates by finding the closest training example to the input data point, based on a predefined distance metric such as Euclidean distance. The class of this nearest neighbour is then assigned to the input. Its strengths are its simplicity of implementation, effectiveness for small datasets, and requiring no training phase, making it straightforward to deploy. However, NN classifiers become computationally expensive as the size of the dataset increases. It is also sensitive to noise and irrelevant features. It also struggles with recognizing patterns in complex datasets due to their reliance on local data points.
- **Hidden Markov Models (HMM):** HMMs are statistical models that are particularly well-suited for recognizing sequences of data, making them useful for tasks involving temporal patterns, such as handwritten text recognition [24]. HMMs are effective for sequential data, capturing temporal dependencies. They can handle variations in handwriting and different writing styles, and they provide a probabilistic framework for handling uncertainties in data. But Training HMMs can be computationally intensive and require large amounts of data. They assume that the states are independent of each other given the current state, which may not hold true in all scenarios. Especially, variability in data quality can significantly affect the performance of HMMs.
- **Multilayer Perceptron (MLP):** MLPs are a type of artificial neural network composed of multiple layers of interconnected neurons, capable of learning non-linear decision boundaries [25]. They excel at learning non-linear patterns, relationships between inputs and outputs, and have a huge versatility in applications. Although MLPs generally provide higher accuracy compared to linear classifiers, training them requires substantial computational resources and time, and they can easily overfit the training data without proper regularization. Also, their performance is highly sensitive to the choice of hyperparameters, such as the number of layers and neurons.
- **Support Vector Machines (SVM):** SVMs are powerful classifiers that find the optimal hyperplane separating different classes in a high-dimensional space and are good at high-dimensional space handling, can efficiently handle non-linear data using kernel functions but are prone to overfitting [26]. Moreover, training is computationally expensive and are less scalable to very large datasets compared to other methods.

2.3.2 Deep Learning Classifiers

Deep learning approaches have revolutionized OCR technology, especially for languages with complex scripts like Bangla. These models leverage large datasets and powerful computational resources to achieve superior performance in character recognition tasks.

- **Deep Convolutional Neural Networks (DCNN):** DCNNs have become the cornerstone of image-based recognition tasks due to their ability to automatically learn hierarchical features from raw data. They offer hierarchical feature learning, high accuracy, and can handle large datasets and benefit from the availability of massive computational resources [27]. But training DCNNs requires large amounts of labeled data and significant computational power because they can overfit if not properly regularized, especially when training data is limited.
- **Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM):** RNNs and LSTMs are specialized for sequence data and are particularly effective for tasks involving sequential information, such as handwriting recognition [28]. They are the most used model in Bangla OCR as they are ideal for recognizing sequences of characters and words. Furthermore, LSTMs are adept at handling long-term dependencies and variations in sequences. However, training RNNs and LSTMs can be challenging due to issues like vanishing and exploding gradients. Especially, the quality and variability of training data can significantly impact the performance of RNNs and LSTMs.

However, the transition from traditional classifiers to deep learning models highlights the ongoing need for extensive computational resources and annotated datasets, which remain a challenge in the context of Bangla OCR. The scarcity of large, high-quality annotated datasets for Bangla script continues to be a significant hurdle.

2.4 Integration of Deep Learning in OCR

The integration of deep learning techniques into OCR systems for Bangla handwritten text has shown promising results but also brought new challenges. Techniques such as transfer learning and data augmentation have been employed to mitigate the limitations of small datasets.

2.4.1 Transfer Learning

Transfer learning involves using pre-trained models on large datasets like ImageNet and fine-tuning them on specific OCR tasks. This approach leverages the knowledge learned from a broader set of images, which can be particularly beneficial when working with limited annotated data. Studies have shown that transfer learning can significantly improve the performance of Bangla OCR systems, as it allows models to generalize better from pre-learned features.

2.4.2 Data Augmentation

Generative Adversarial Networks (GANs) have been explored for data augmentation, generating synthetic handwritten Bangla text to augment training datasets. These approaches help in addressing the data scarcity issue and improving the robustness of OCR models. By generating diverse and realistic handwritten samples, GANs can enhance the variability in training data, leading to more robust models.

2.4.3 Sequence-to-Sequence Models

The use of sequence-to-sequence models and attention mechanisms has also been investigated to improve the accuracy of Bangla handwritten text recognition. These models can dynamically focus on different parts of an input sequence, making them particularly effective for scripts with complex character dependencies and variations. Attention mechanisms allow the model to weigh different parts of the input differently, improving recognition accuracy for complex and noisy text.

2.5 Research Gaps and Opportunities

The literature on Bangla handwritten OCR underscores the complexity and richness of the field. While significant progress has been made through innovative feature extraction techniques, sophisticated segmentation methods, and advanced classifier approaches, challenges remain, particularly in dealing with complex document layouts and noisy data.

2.5.1 Data Scarcity and Quality

One of the major challenges in Bangla OCR is the scarcity of large, high-quality annotated datasets. Collaborative efforts to create and share large, annotated datasets for Bangla handwritten text are crucial. Open-source initiatives and crowdsourcing could play a significant role in overcoming the data scarcity barrier. Ensuring data diversity in terms of handwriting styles, document types, and noise levels is essential for developing robust OCR systems.

2.5.2 Model Robustness and Scalability

Future research is poised to benefit from integrating more advanced deep learning techniques and developing robust, scalable solutions that can effectively handle the diverse and intricate nature of Bangla handwritten script. Exploring end-to-end trainable models that can jointly perform detection, segmentation, and recognition tasks could further enhance the performance and applicability of Bangla OCR systems. Additionally, improving model robustness to handle noisy and complex document layouts is an ongoing challenge.

2.5.3 Computational Resources

The transition to deep learning models also highlights the need for extensive computational resources. Efficient training and inference on large datasets require powerful hardware such as GPUs and TPUs. Research into model optimization techniques, such as quantization and pruning, can help reduce the computational demands of deep learning models, making them more accessible and scalable.

In conclusion, the field of Bangla handwritten OCR is evolving, with deep learning offering new possibilities to tackle long-standing challenges. Continued research and innovation are essential to unlock the full potential of OCR technology for Bangla and other complex scripts, ultimately contributing to the digitalization and preservation of linguistic heritage.

Chapter 3

Proposed Methodology

The methodology section outlines the systematic approach adopted to develop and validate a hybrid model for Bangla handwritten Optical Character Recognition (OCR). This section is crucial as it provides a detailed account of the procedures, techniques, and tools utilized in the research, ensuring reproducibility and transparency. By presenting a comprehensive methodology, this section highlights the rigor of the research process and its alignment with the overall research objectives.

3.1 Theoretical Framework

This research aims to address these challenges by developing a hybrid model that combines a detection component for isolating characters and a recognition component for accurately identifying them. This dual approach allows for the separation of the two tasks, leading to improved performance and accuracy in Bangla handwritten OCR.

Just like it is shown in, Figure 3.1 A spelling correction model can be optionally plugged in at the end of the pipeline for increased accuracy and error correction.

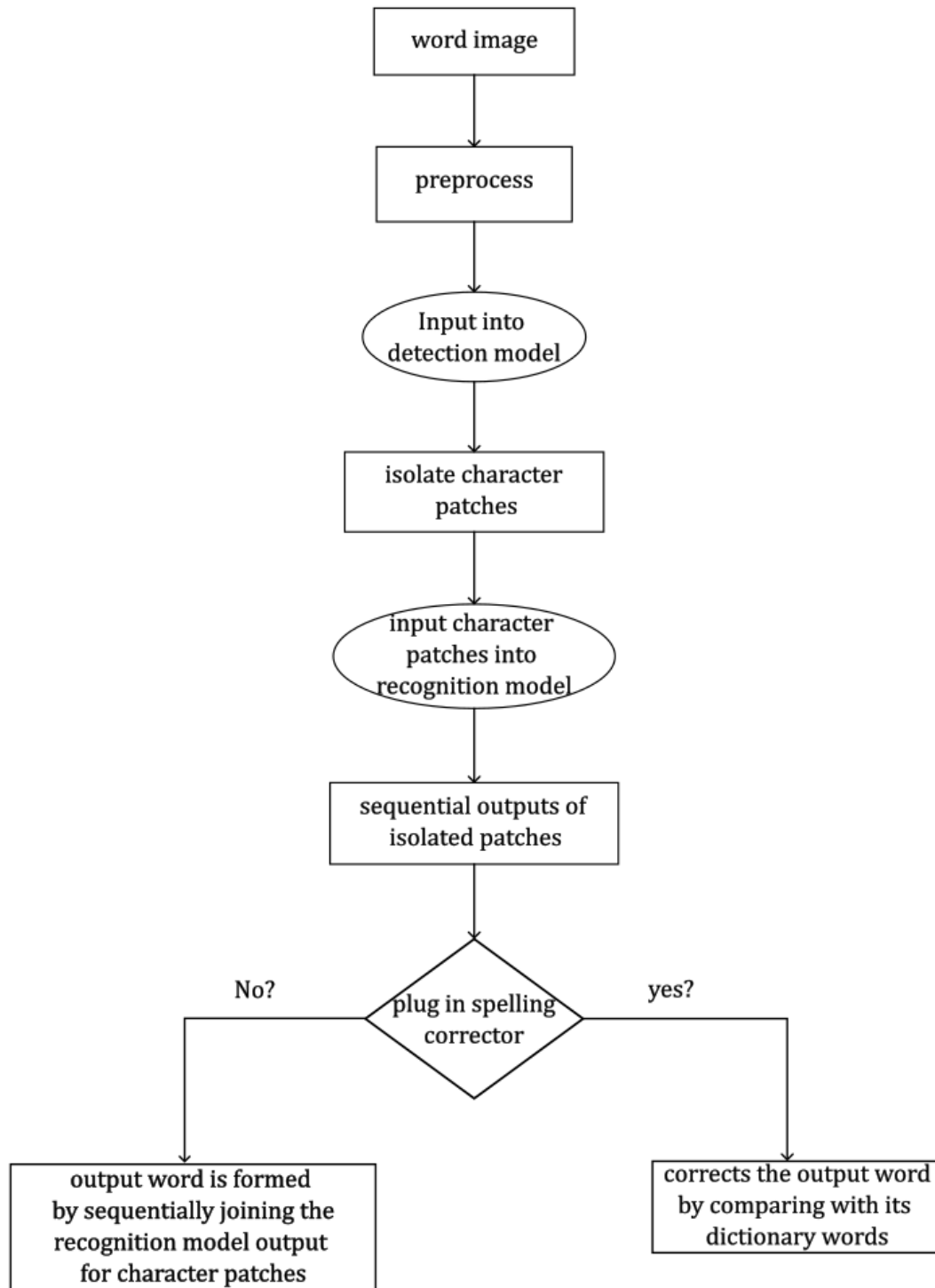


Figure 3.1: Framework for the proposed pipeline

3.1.1 Pipeline Example

Detection Part of the Pipeline:

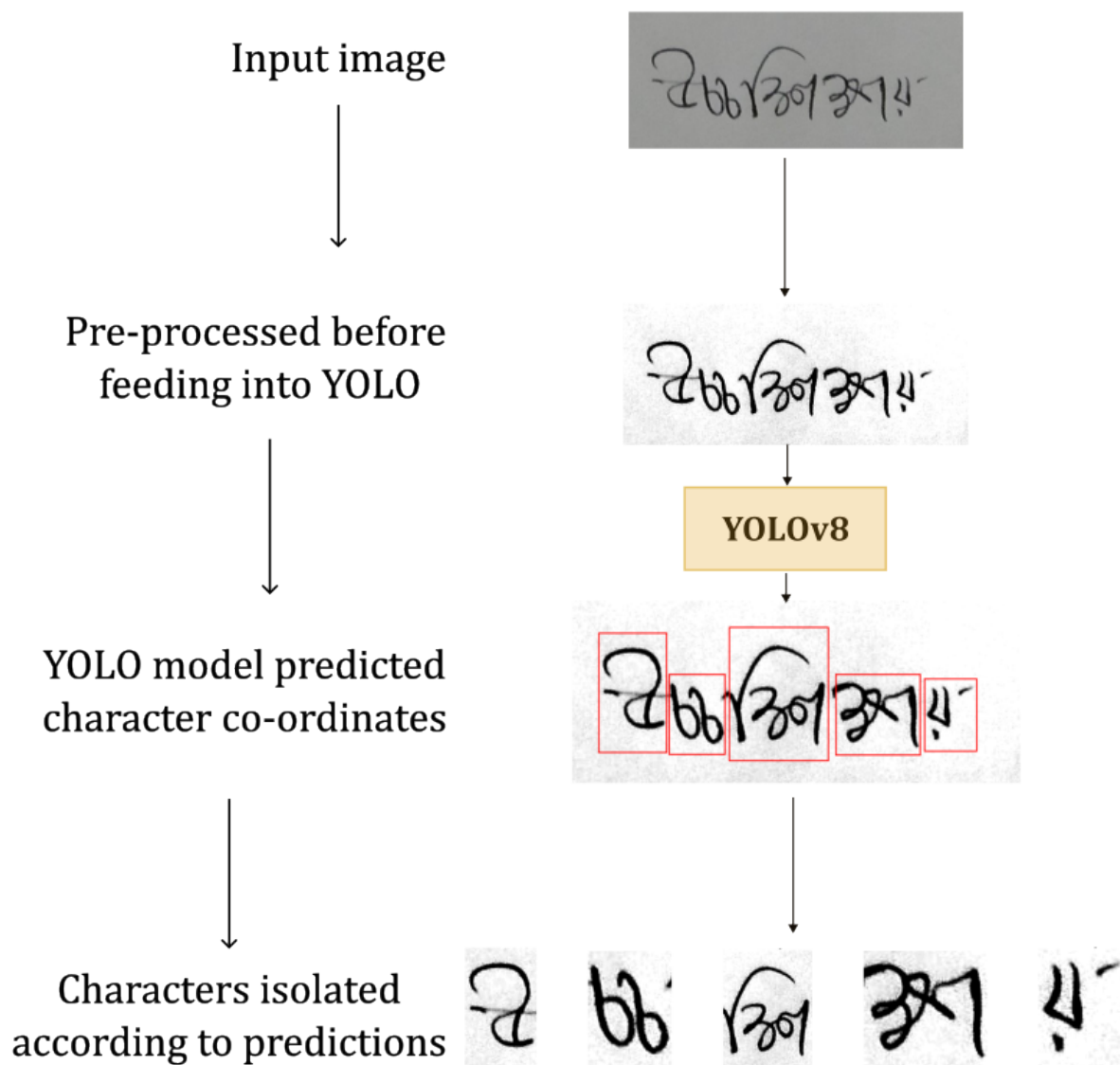


Figure 3.2: Utilisation of the proposed pipeline(Detection part)

Recognition Part of the Pipeline:

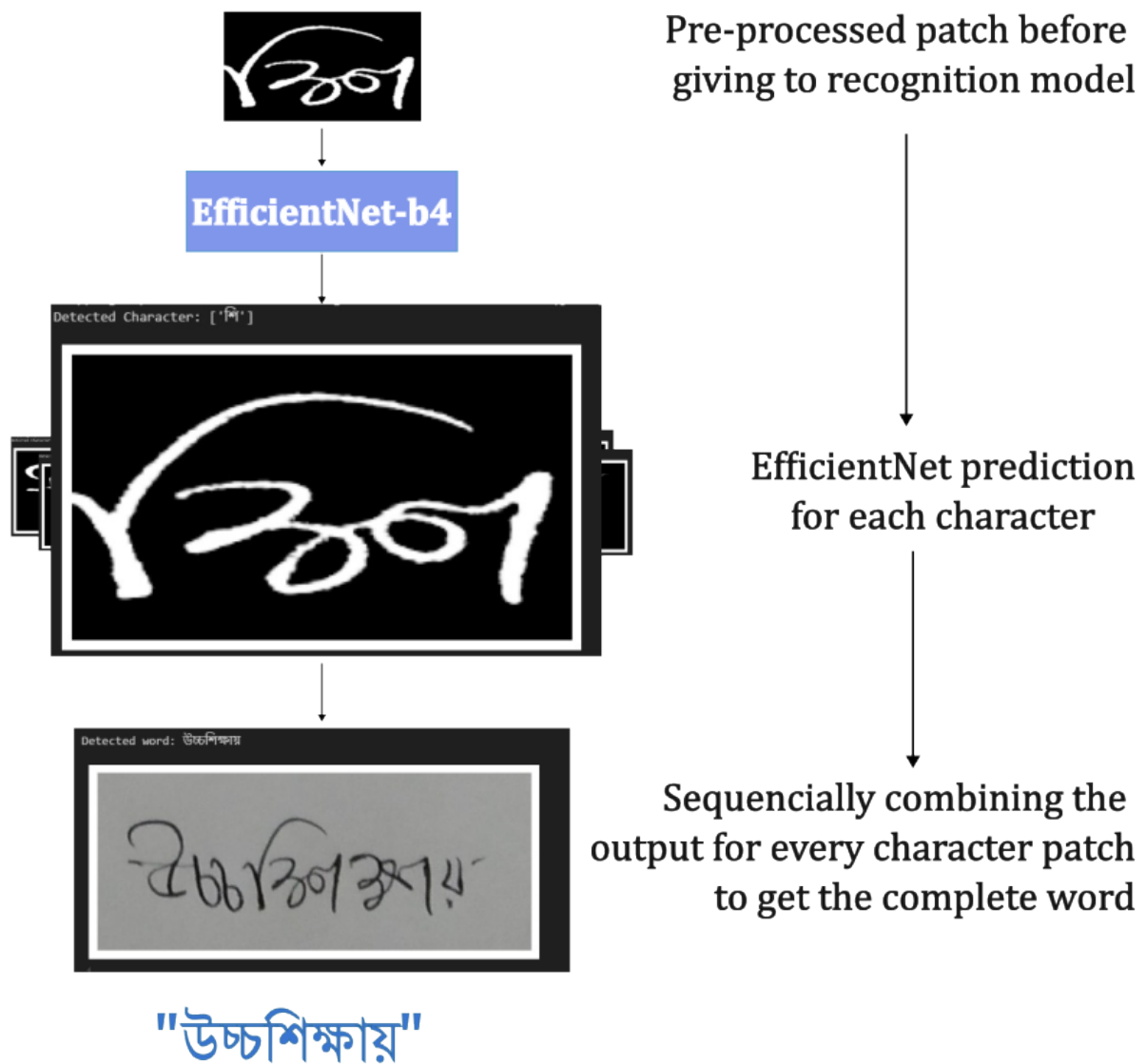


Figure 3.3: Utilisation of the proposed pipeline(Recognition part)

3.2 System Design

3.2.1 Choosing Detection Model for Character Isolation

Due to the previously discussed complexities of the Bangla writing system, a versatile model is needed to tackle the problems intrinsic to the Bangla handwriting. The available models suitable for character isolation from Bangla words are shown in Table 3.1.

Model	Convolutional Layers	Fully Connected Layers	Parameters	Other Key Features
YOLOv8	67	2	85 million	Batch normalization, leaky ReLU activation, multiple anchors per grid cell
Faster R-CNN	VGG16: 13 conv layers	3	134 million	Region Proposal Network (RPN), feature pyramid network (FPN)
Mask R-CNN	ResNet-50: 50 conv layers	3	44 million	Adds segmentation mask prediction, ROI Align
U-Net	Varies (e.g., 23 layers)	2	31 million	Encoder-decoder structure, skip connections
SegNet	VGG16: 13 conv layers	0	29 million	Encoder-decoder architecture, max pooling indices
PSPNet	ResNet-101: 101 conv layers	2	65 million	Pyramid pooling, global context embedding
DeepLabv3+	Xception: 71 conv layers	1	43 million	Atrous convolution, spatial pyramid pooling
CRAFT	VGG16-based: 13 conv layers	2	23 million	Character region awareness, link refiner for character separation
EAST	VGG16-based: 13 conv layers	2	23 million	Efficient, single-pass detection, score maps for text regions
SSD	VGG16-based: 13 conv layers	2	24 million	Multiscale feature maps, default bounding boxes

Table 3.1: Comparison of various deep learning models with their key architectural features.

But the model most suitable for Bangla handwriting needs to handle the complexity of the script, including various modifiers, conjunct characters, and potentially overlapping or touching characters. A comparative analysis of the best three models based on these requirements is briefly discussed in Table 3.2.

Model	Strengths	Suitability for Bangla Handwritten Character Isolation
YOLO	Real-time performance, high efficiency, robust to noise, multiple object detection	Ideal for fast processing, handling varying complexity and noise effectively.
Mask R-CNN	Pixel-level segmentation, handles complex scripts, versatile with shapes and sizes	Excellent for detailed segmentation and complex, closely spaced characters.
CRAFT	Character-level detection, high precision, robust on diverse data, link refinement	Perfect for accurately isolating individual characters in noisy or variable handwritten text.

Table 3.2: Comparison of various models with their strengths and suitability for Bangla handwritten character isolation.

Mask R-CNN is generally the best option for Bangla handwritten word images due to its ability to perform detailed pixel-level segmentation, which is crucial for dealing with the intricacies of Bangla script, including closely connected characters and modifiers. However, if real-time performance and efficiency are critical, **YOLO** provides a good balance between speed and accuracy, especially with well-tuned bounding boxes. **CRAFT** stands out for its precision in character-level detection, making it ideal for applications requiring high accuracy and detailed isolation of characters.

However, as our proposed system requires for a detection model with dependable real-time-processing, we have chosen to employ the YOLO model as the detection component.

3.2.2 YOLO Model Architecture

The You Only Look Once (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. The model architecture of YOLO can be described as follows:

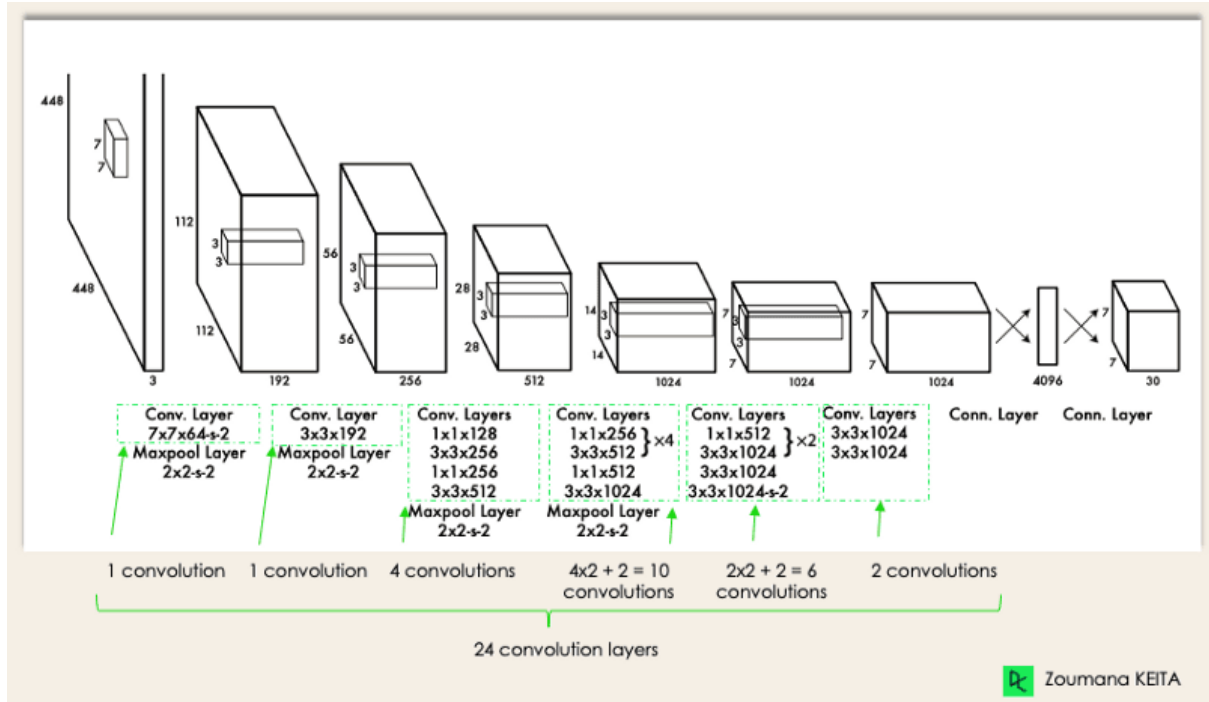


Figure 3.4: YOLO Architecture. Source: [1].

Backbone Network

- **Convolutional Layers:** The core of YOLO's architecture is a series of convolutional layers. These layers extract features from the input image by applying convolution operations with various kernel sizes and strides, capturing spatial hierarchies and patterns essential for object detection.
- **Batch Normalization:** To improve convergence and regularization, batch normalization layers are integrated after most convolutional layers. These layers normalize the input to each layer, stabilizing the learning process and allowing for higher learning rates.
- **Activation Functions:** YOLO employs leaky ReLU (Rectified Linear Unit) activation functions after each convolutional layer. The leaky ReLU allows a small, non-zero gradient when the unit is not active, addressing the "dying ReLU" problem and promoting better gradient flow.

Detection Head

- **Fully Connected Layers:** Traditional YOLO models utilize fully connected layers at the end of the network to predict the final output. These layers convert the

high-dimensional feature maps into a structured output that includes bounding box coordinates, objectness scores, and class probabilities.

- **Bounding Box Prediction:** The model divides the input image into an $S \times S$ grid. Each grid cell predicts a fixed number of bounding boxes (typically 2) and corresponding confidence scores. The confidence score reflects the likelihood that a bounding box contains an object and the accuracy of the predicted bounding box.
- **Class Prediction:** For each grid cell, the model also predicts conditional class probabilities. These probabilities indicate the likelihood of each class being present in the grid cell, assuming that an object is indeed present.

Output Representation

- **Bounding Box Coordinates:** Each bounding box prediction consists of five components: (t_x, t_y, t_w, t_h) , and (t_c) . Here, (t_x) and (t_y) represent the center coordinates relative to the grid cell, (t_w) and (t_h) denote the width and height of the bounding box, and (t_c) indicates the confidence score.
- **Class Probabilities:** The class probabilities are predicted for each grid cell, providing a distribution over all possible classes. These probabilities are multiplied by the bounding box confidence scores to yield class-specific confidence scores for each bounding box.

Architectural Variants

- **YOLOv8:** Introduced advanced feature extraction techniques such as EfficientNet-based backbones, self-attention mechanisms, and improved anchor-free detection heads to enhance accuracy and efficiency. YOLOv8 also incorporates dynamic anchor boxes and a better feature pyramid network (FPN) to capture multi-scale features effectively.
- **YOLOv9:** Emphasized the use of transformers and attention mechanisms to improve the model's ability to focus on relevant regions in the image. YOLOv9 adopted a hybrid architecture combining CNNs with vision transformers (ViTs) for enhanced contextual understanding and spatial awareness.
- **YOLOv10:** Focused on further refining the integration of transformers, leveraging advanced techniques such as Swin Transformers and dynamic convolutional layers to enhance feature representation and model performance. YOLOv10 also introduced advanced data augmentation strategies and self-supervised learning techniques to improve robustness and generalization.

In this study, we have focused on the integration and training of the YOLOv8 model shown in Figure 3.5.

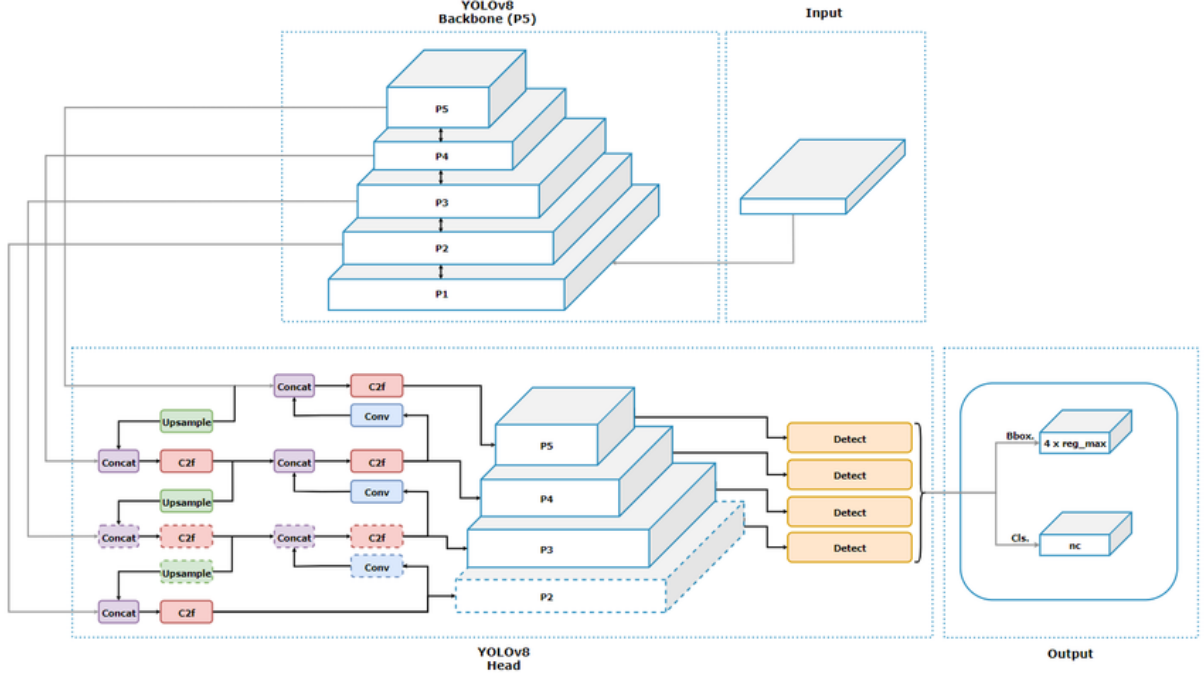


Figure 3.5: The improved YOLOv8 network architecture includes an additional module for the head represented in the rectangle with a dashed outline. Source: [2].

3.2.3 Selecting Recognition Model for Identification

For the recognition task in our proposed pipeline, we have considered four basic model architectures, ResNet, GoogLeNet, DenseNet, and EfficientNet. Table 3.3 includes a summary of the examined models.

Architecture	Conv Layers	FC Layers	Parameters	Additional Info
ResNet-18	20	1	11.7 million	8 residual blocks
ResNet-34	36	1	21.8 million	16 residual blocks
ResNet-50	49	1	25.6 million	16 bottleneck blocks (1x1, 3x3, 1x1 convolutions)
ResNet-101	100	1	44.5 million	33 bottleneck blocks
ResNet-152	151	1	60.2 million	50 bottleneck blocks
GoogLeNet (Inception v1)	57	1	6.8 million	Inception modules, 9 inception modules
DenseNet-121	120	1	8.0 million	4 dense blocks
DenseNet-161	160	1	28.7 million	4 dense blocks
DenseNet-169	166	1	14.3 million	4 dense blocks
DenseNet-201	200	1	20.0 million	4 dense blocks
EfficientNet-B0	16	1	5.3 million	8 MBConv blocks
EfficientNet-B1	16	1	7.8 million	8 MBConv blocks
EfficientNet-B2	16	1	9.2 million	8 MBConv blocks
EfficientNet-B3	16	1	12.0 million	8 MBConv blocks
EfficientNet-B4	16	1	19.0 million	8 MBConv blocks
EfficientNet-B5	16	1	30.0 million	8 MBConv blocks
EfficientNet-B6	16	1	43.0 million	8 MBConv blocks
EfficientNet-B7	16	1	66.0 million	8 MBConv blocks
EfficientNetV2-S	19	1	24.2 million	17 MBConv/Fused-MBConv blocks
EfficientNetV2-M	35	1	54.2 million	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 3.3: Comparison of Various Neural Network Architectures

3.2.4 ResNet Model Architecture

The ResNet (Residual Network) model is a deep convolutional neural network architecture designed to address the degradation problem in deep networks by introducing residual learning. This model enables the training of extremely deep networks by incorporating shortcut connections, which allow gradients to propagate more effectively through the network.

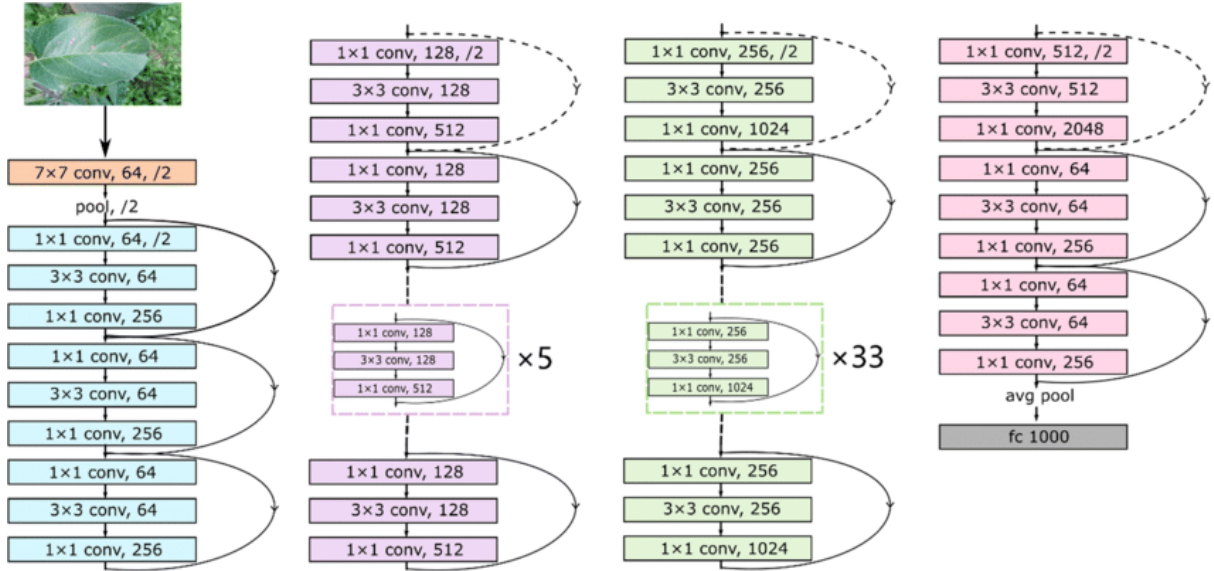


Figure 3.6: Neural network architecture of ResNet152. The dotted shortcuts increase dimensions and the layers inside the dashed box represents n duplicate convolutional layers. Source: [3].

Convolutional Layers:

ResNet begins with a conventional convolutional layer, typically followed by a batch normalization layer and a ReLU activation function. This initial stage is designed to extract low-level features from the input image.

Residual Blocks: The core innovation of ResNet is the introduction of residual blocks. Each residual block consists of a series of convolutional layers with batch normalization and ReLU activation functions. The key feature of a residual block is the shortcut connection that bypasses one or more convolutional layers. The output of the shortcut connection is added to the output of the stacked layers. Mathematically, if the input to a residual block is denoted as x , the output y is given by

$$y = F(x, \{W_i\}) + x,$$

where $F(x, \{W_i\})$ represents the residual mapping to be learned. This formulation helps

mitigate the vanishing gradient problem and facilitates the training of very deep networks.

Bottleneck Design: For deeper networks, ResNet employs a bottleneck design within the residual blocks. This design involves three convolutional layers instead of two: a 1x1 convolutional layer that reduces the dimensionality, a 3x3 convolutional layer, and another 1x1 convolutional layer that restores the dimensionality. This bottleneck architecture reduces the computational complexity while maintaining the network's expressiveness.

Network Depth: ResNet comes in various depths, such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. The number in each variant denotes the total number of layers in the network. For example, ResNet-50 consists of 49 convolutional layers followed by a fully connected layer.

Fully Connected Layers: Following the convolutional and residual blocks, ResNet includes a global average pooling layer that reduces the spatial dimensions of the feature maps. This is followed by a fully connected layer that outputs the final class scores.

Shortcut Connections: The shortcut connections in ResNet can be identity mappings or projection mappings. Identity mappings are used when the input and output dimensions are the same, while projection mappings, implemented using 1x1 convolutions, are used when the dimensions differ. This flexibility allows ResNet to adapt to different network depths and configurations.

3.2.5 GoogLeNet Model Architecture

The GoogLeNet model, also known as Inception, represents a significant advancement in deep convolutional neural network architectures. It is designed to achieve high performance while maintaining computational efficiency through a novel module called the Inception module.

Convolutional Layers: GoogLeNet starts with several standard convolutional layers, which include convolution operations followed by rectified linear unit (ReLU) activations. These initial layers extract fundamental features from the input image.

Inception Modules: The cornerstone of GoogLeNet's architecture is the Inception module. An Inception module aims to capture multi-scale features by performing convolutions with different filter sizes (1x1, 3x3, and 5x5) within the same module. Each module includes:

- 1x1 convolutions that reduce the dimensionality and computational cost.
- 3x3 and 5x5 convolutions that capture spatial features at different scales.
- 3x3 max pooling operations for additional spatial feature extraction.

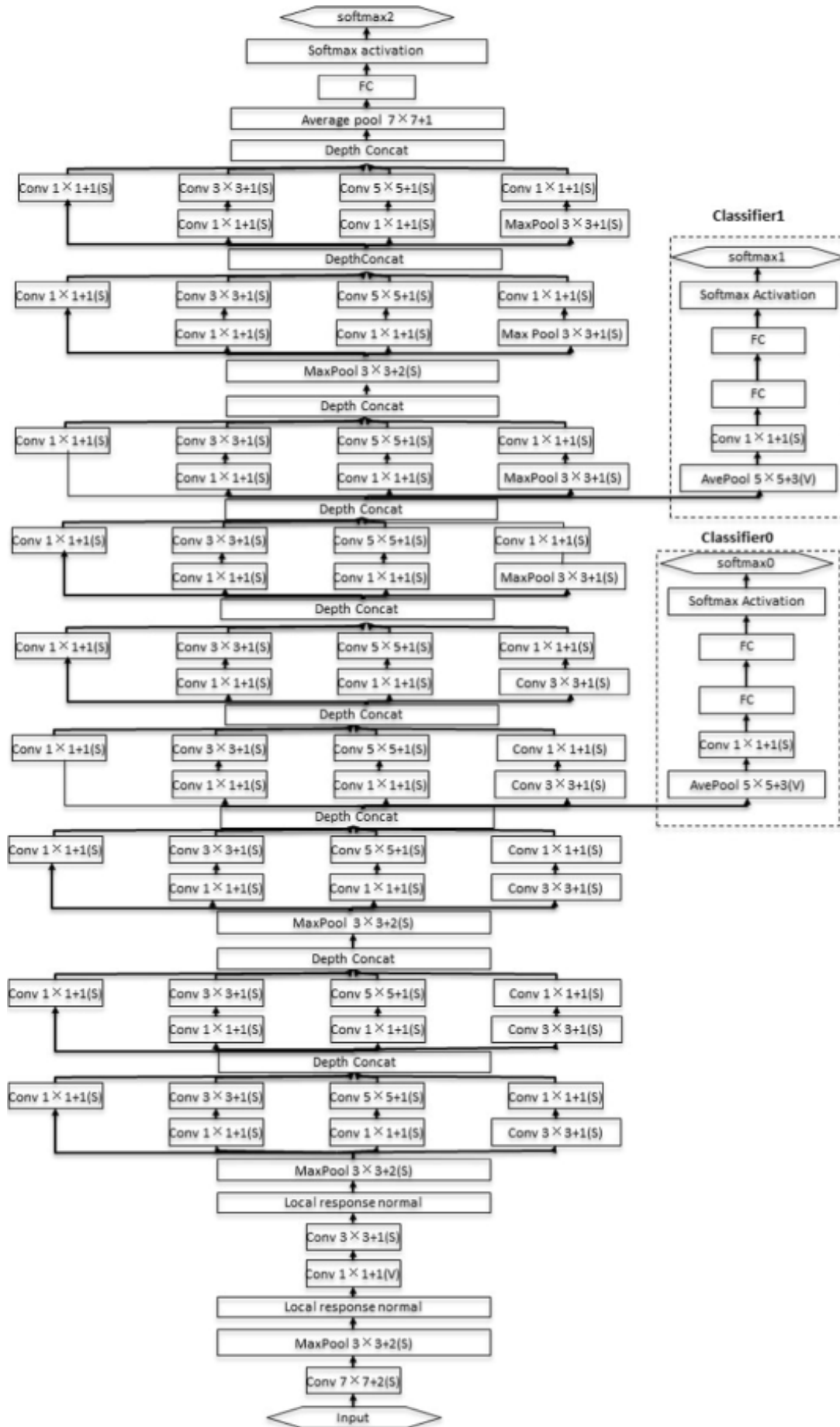


Figure 3.7: GoogLeNet Architecture. Source: [4].

The outputs of these operations are concatenated along the channel dimension, allowing the network to process information at multiple scales simultaneously.

Dimensionality Reduction: A crucial aspect of the Inception module is the use of 1x1 convolutions for dimensionality reduction. These operations precede the more computationally expensive 3x3 and 5x5 convolutions, significantly reducing the number of parameters and thus the computational cost.

Network Depth: GoogLeNet employs a deep architecture with 22 layers (excluding pooling layers) to enhance feature extraction and representation learning. Despite its depth, the model is computationally efficient due to the inception modules' design.

Auxiliary Classifiers: To improve gradient flow and address the vanishing gradient problem in deeper networks, GoogLeNet introduces auxiliary classifiers at intermediate layers. These auxiliary classifiers, which consist of a few additional convolutional layers and a fully connected layer, act as regularizers during training. They are connected to softmax layers that compute auxiliary loss, encouraging better feature learning in earlier layers.

Fully Connected Layers: Unlike traditional CNNs, GoogLeNet replaces fully connected layers with global average pooling layers towards the end of the network. This pooling layer averages the spatial dimensions of the feature maps, significantly reducing the number of parameters and preventing overfitting.

3.2.6 DenseNet Model Architecture

The DenseNet (Densely Connected Convolutional Networks) model is a deep learning architecture designed to address the challenges of vanishing gradients, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. This is achieved through its distinctive dense connectivity pattern.

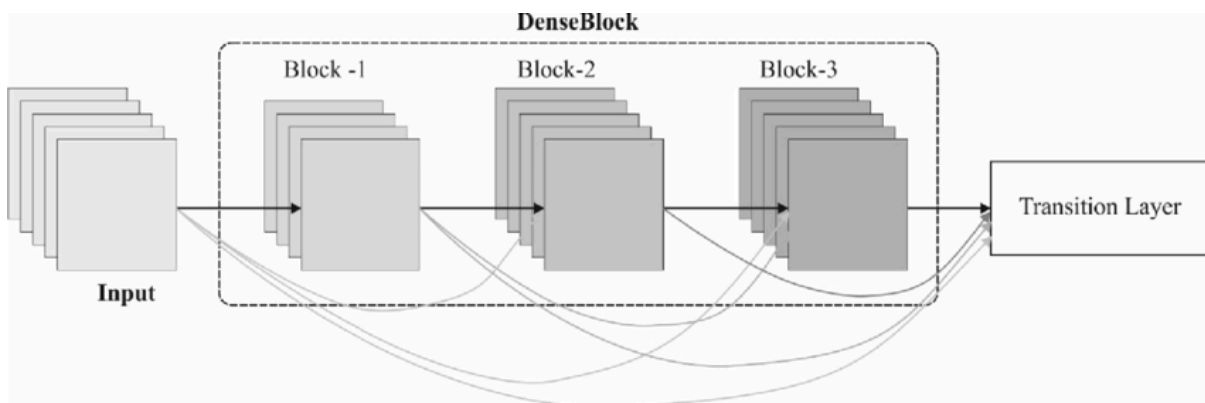


Figure 3.8: DenseNet Architecture. Source: [5].

Dense Blocks: The core component of DenseNet is the dense block. Within a dense block, each layer is connected to every other layer in a feed-forward fashion. This means that the input to each layer consists of the feature maps of all preceding layers within the same block. Formally, the output of the l -th layer is defined as

$$x_l = H_l([x_0, x_1, \dots, x_{l-1}]),$$

where $[x_0, x_1, \dots, x_{l-1}]$ represents the concatenation of the feature maps produced by layers 0 to $l - 1$ and H_l denotes the composite function of batch normalization, ReLU, and convolution operations.

Transition Layers: Between dense blocks, DenseNet includes transition layers that serve to control the complexity of the model. A transition layer typically consists of a batch normalization layer, a 1x1 convolutional layer (which reduces the number of feature maps), and a 2x2 average pooling layer (which halves the spatial dimensions). These layers reduce the dimensions of feature maps and help in controlling the model's capacity.

Growth Rate: A key hyperparameter in DenseNet is the growth rate, denoted by k . The growth rate defines the number of feature maps added by each layer within a dense block. If a dense block has l layers, the total number of input feature maps to the last layer in this block is

$$k_0 + k \times (l - 1),$$

where k_0 is the number of input feature maps to the block. The growth rate ensures a manageable increase in the number of parameters and promotes feature reuse.

Network Depth: DenseNet can vary in depth, similar to other deep learning architectures. DenseNet-121, DenseNet-169, DenseNet-201, and DenseNet-264 are common configurations, with the numbers indicating the total number of layers in the network. These configurations typically comprise multiple dense blocks interspersed with transition layers.

Bottleneck Layers: To further reduce computational complexity, DenseNet often employs bottleneck layers within dense blocks. A bottleneck layer consists of a 1x1 convolution followed by a 3x3 convolution. This design, similar to that used in ResNet's bottleneck blocks, reduces the number of input feature maps before applying the computationally expensive 3x3 convolution, thereby enhancing efficiency.

Fully Connected Layers: DenseNet concludes with a global average pooling layer that reduces the spatial dimensions of the feature maps to a single value per feature map. This is followed by a fully connected layer that produces the final classification scores.

3.2.7 EfficientNet Model Architecture

The EfficientNet model architecture represents a significant advancement in the design of convolutional neural networks (CNNs), emphasizing a balanced scaling approach to improve performance while maintaining computational efficiency. EfficientNet is based on a systematic scaling method that uniformly scales all dimensions of depth, width, and resolution using a compound coefficient.

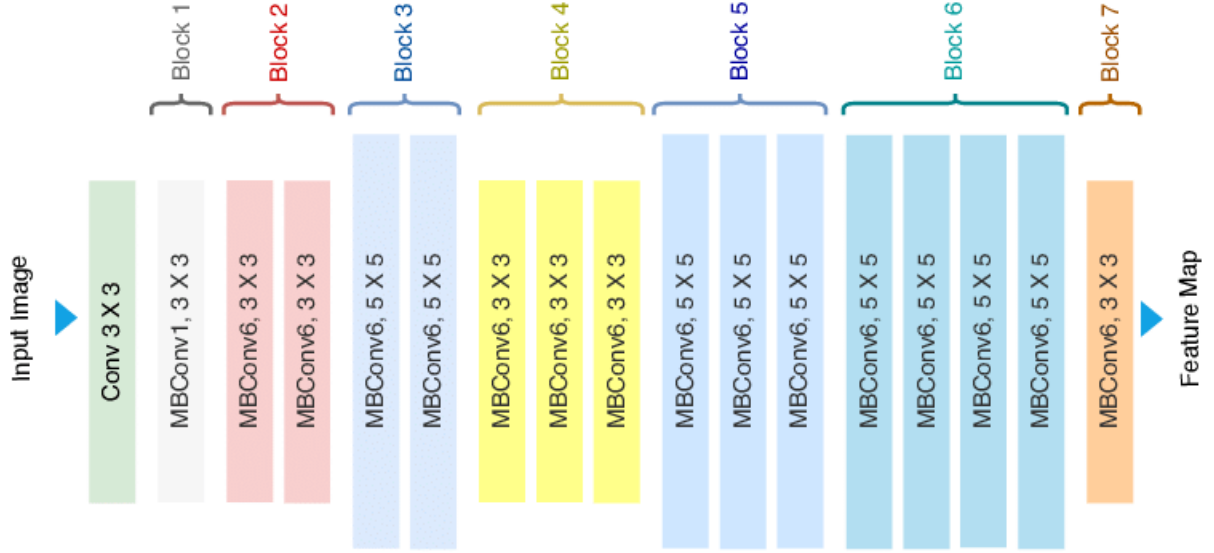


Figure 3.9: EfficientNet-B0 architecture. Source: [6].

Compound Scaling Method:

EfficientNet employs a compound scaling method to uniformly scale network dimensions. The key idea is to use a compound coefficient ϕ to scale the depth (d), width (w), and resolution (r) of the network as follows:

$$d = \alpha^\phi$$

$$w = \beta^\phi$$

$$r = \gamma^\phi$$

where α , β , and γ are constants determined through a grid search to balance these dimensions effectively. This method ensures that all dimensions are scaled proportionally, leading to improved accuracy and efficiency.

Mobile Inverted Bottleneck Convolution (MBConv):

At the core of EfficientNet is the MBConv block, which is derived from MobileNetV2. An MBConv block consists of:

- A pointwise 1x1 convolution for dimensionality expansion.
- A depthwise convolution that applies a single convolutional filter per input channel, reducing computational cost.
- A pointwise 1x1 convolution for dimensionality reduction.
- A squeeze-and-excitation optimization to recalibrate channel-wise feature responses adaptively.

This structure enhances the network’s ability to capture fine-grained features while maintaining computational efficiency.

Network Architecture:

EfficientNet starts with a simple stem consisting of a standard convolutional layer followed by a batch normalization layer and a ReLU6 activation. The network then consists of a series of MBConv blocks, organized into stages with varying numbers of filters and resolutions. Each stage increases the number of filters and reduces the spatial dimensions of the feature maps through stride-2 convolutions, capturing more complex features at higher levels.

Scaling Variants:

EfficientNet comes in various scaling variants, such as EfficientNet-B0 to EfficientNet-B7. These variants are created by applying different values of the compound coefficient ϕ , systematically scaling the network’s depth, width, and resolution. For example, EfficientNet-B0 is the baseline model, while EfficientNet-B7 is the most scaled-up version, providing the highest accuracy at the cost of increased computational resources.

Fully Connected Layers:

The network concludes with a global average pooling layer that reduces the spatial dimensions of the feature maps to a single value per feature map. This is followed by a fully connected layer that produces the final classification scores. The global average pooling layer helps prevent overfitting by reducing the number of parameters in the final fully connected layer.

3.2.8 Spelling Correction Model and Architecture

The Word2Vec model is a neural network-based approach designed to learn vector representations of words, capturing their semantic relationships and contextual meanings. Developed by Mikolov et al. (2013), Word2Vec is widely used in natural language processing (NLP) tasks for its ability to generate high-quality word embeddings.

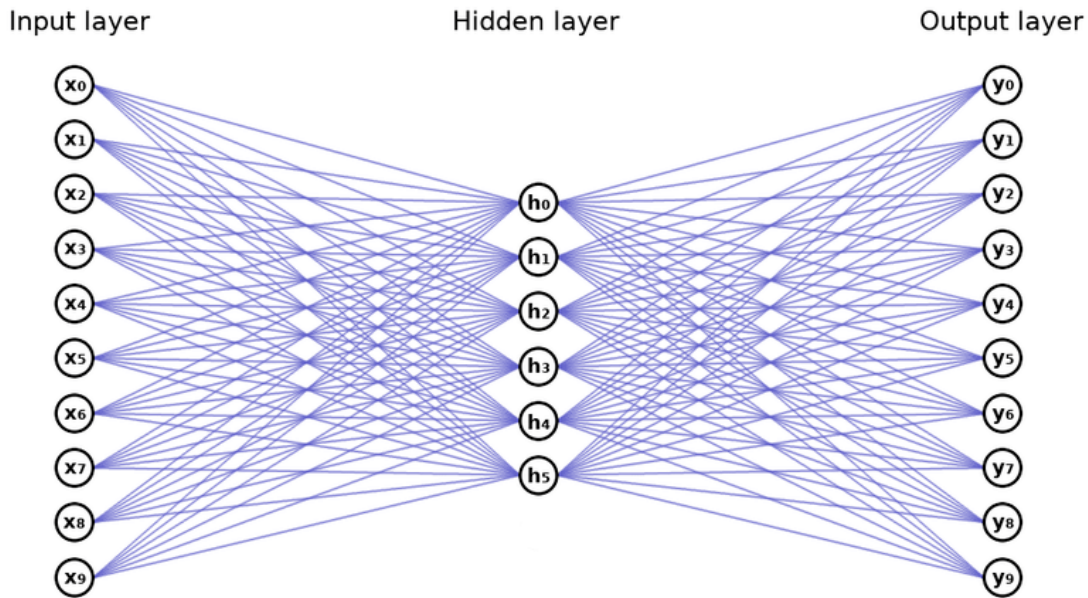


Figure 3.10: Word2Vec architecture. In addition to an input layer and an output layer, shallow architectures have a small number of hidden layers (one in this case). Source: [7].

Model Overview: Word2Vec employs two main architectures: Continuous Bag of Words (CBOW) and Skip-Gram. Both architectures aim to predict the contextual relationships between words, but they do so in opposite ways. CBOW predicts the target word given its context, while Skip-Gram predicts the context words given a target word.

Continuous Bag of Words (CBOW): In the CBOW architecture, the objective is to predict a target word based on its surrounding context words. The model consists of the following components:

- *Input Layer:* The input layer receives a context window of words surrounding the target word. This window size is a hyperparameter that determines the number of context words considered.
- *Projection Layer:* The input words are represented as one-hot vectors, which are then mapped to dense vectors (embeddings) through a shared weight matrix. The embeddings of the context words are averaged to form a single vector.

- *Output Layer:* The averaged context vector is fed into a softmax classifier that predicts the target word's probability distribution over the vocabulary.

Skip-Gram In the Skip-Gram architecture, the objective is to predict context words given a target word. The model consists of the following components:

- *Input Layer:* The input layer receives a single target word represented as a one-hot vector.
- *Projection Layer:* The target word vector is mapped to a dense vector (embedding) through a weight matrix.
- *Output Layer:* The embedding is used to predict the probability distribution of each context word within the defined window size using softmax classifiers. Multiple context words are predicted for each target word.

Training Objective: Both CBOW and Skip-Gram models are trained using the negative sampling or hierarchical softmax techniques to approximate the softmax function and reduce computational complexity. The training objective is to maximize the likelihood of predicting the correct context words (in Skip-Gram) or the target word (in CBOW) while minimizing the likelihood of incorrect predictions.

Application in NLP Pipelines: Word2Vec can be integrated into NLP pipelines to enhance performance in various tasks such as text classification, machine translation, and sentiment analysis. By providing high-quality word embeddings, Word2Vec captures semantic similarities and contextual nuances that improve the accuracy and robustness of downstream models. Additionally, Word2Vec embeddings can be fine-tuned or used as pre-trained vectors to initialize neural network layers, contributing to faster convergence and better generalization.

3.3 Reasoning behind the Hybrid Model Approach

The core of our methodology revolves around the implementation of a hybrid model that combines the strengths of EfficientNet for character recognition and YOLO (You Only Look Once) for object detection. This hybrid approach is chosen for several reasons:

- **Separation of Concerns:** By segregating the detection and recognition tasks, we can optimize each component individually, leading to more precise and efficient overall performance. YOLO excels in object detection, providing high accuracy in identifying and localizing characters within an image. EfficientNet, on the other hand, is a state-of-the-art convolutional neural network architecture known for its scalability and efficiency in image recognition tasks.
- **Handling Complex Scripts:** The Bangla script's complexity, with its numerous modifiers and compound characters, requires a robust detection mechanism to accurately segment characters before recognition. YOLO's capability to detect multiple objects in a single image frame makes it an ideal choice for this purpose. Once the characters are accurately detected, EfficientNet can effectively recognize them, even in the presence of intricate variations.
- **Performance and Accuracy:** Combining YOLO and EfficientNet leverages the high detection speed and accuracy of YOLO with the recognition power of EfficientNet, creating a synergistic effect that enhances the overall OCR system's performance. This hybrid approach allows us to tackle the specific challenges posed by Bangla handwritten text, ensuring higher accuracy and reliability compared to traditional methods or single-model approaches.
- **Flexibility and Scalability:** The modular nature of the hybrid model offers flexibility in fine-tuning and scaling the system. Improvements or updates in either the detection or recognition models can be incorporated independently, allowing for continuous enhancement of the OCR system.

Chapter 4

Data Pre-processing and Training

4.1 Datasets

4.1.1 Detection Model Training Data

Synthetically Generated Bangla Corpus: The training data for the detection models consisted of a synthetically generated Bangla corpus containing 2 million samples [14]. This dataset encompasses a diverse array of fonts and domains, ensuring a broad representation of vocabulary sizes. Its primary objective is to enhance the recognition accuracy of conjunct characters in Bangla text, thereby improving the overall performance of the OCR pipeline.

Collected Data: For training the detection models, we used a dataset of Bangla handwritten text comprising 300 word images. These images were collected from 20 participants representing diverse educational backgrounds, including individuals in higher secondary education and various undergraduate years. Each participant contributed 15 handwritten words. Which can be seen in Figure 4.1.

The length distribution of commonly used words, as seen in Table 4.1, was analyzed based on Bangla Wikipedia articles. The selection of 15 words per participant was designed to align with this distribution, covering a range of word lengths commonly found in Bangla text. The dataset includes real-world noises encountered in handwritten text, such as variations in writing styles, ink blots, and smudges. This ensured that the feature learning of the YOLO model conducted under realistic conditions, reflecting practical challenges.

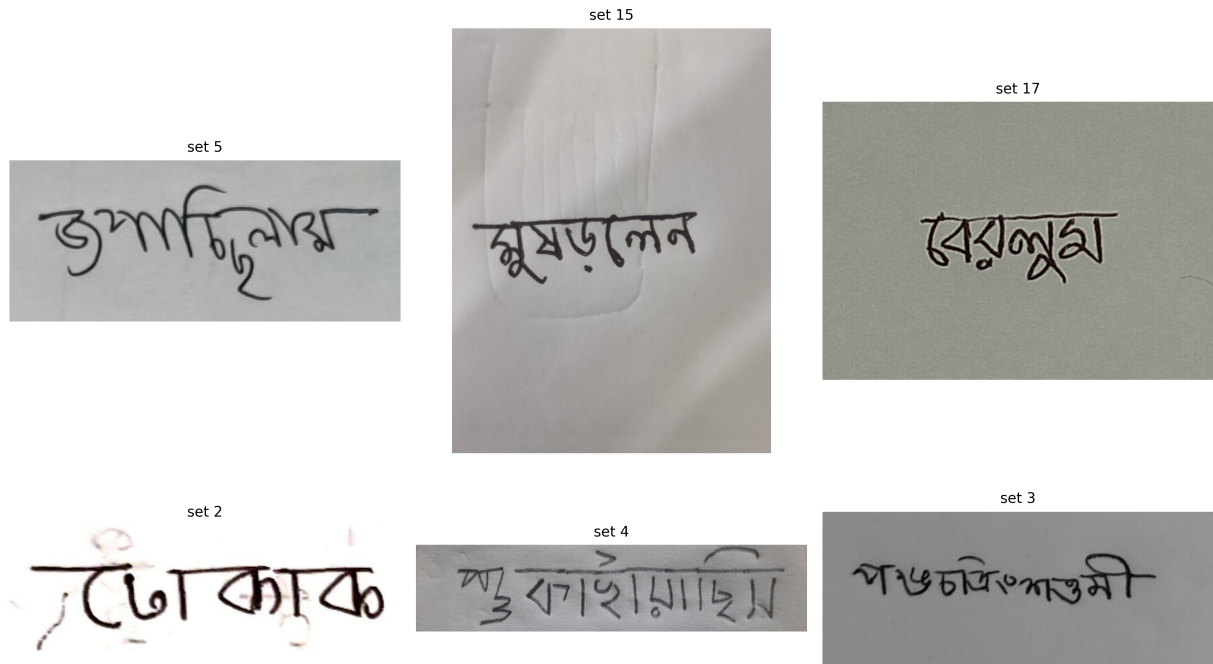


Figure 4.1: Example of the Dataset

Length	Count
1	24
2	429
3	2620
4	8729
5	15108
6	18148
7	16938
8	15187
9	12450
10	9865
11	6906
12	5795
13	1796
14	965
15	371
16	98
17	42
18	16
19	6
20	2
21	2
23	1

Table 4.1: Word Length Distribution¹

4.1.2 Recognition Model Training Data

Handwritten Grapheme Dataset: The recognition models were trained using the Handwritten Grapheme Dataset, which includes 411,882 images. This dataset is curated for the Bengali Grapheme Classification competition on Kaggle [29], aiming to benchmark algorithms for multi-target grapheme classification. It uses a unique grapheme-based labeling scheme for simple and compound characters [30], categorizing graphemes into roots, vowel diacritics, and consonant diacritics as separate targets. This dataset serves as a valuable resource for evaluating multi-target classification algorithms and supports word-level OCR studies.

BanglaLekha-Isolated: Additionally, the recognition models were trained on the BanglaLekha-Isolated dataset [11], which contains 84 different characters, including 50 Bangla basic characters, 10 Bangla numerals, and 24 selected compound characters. After pre-processing and error removal, this dataset comprises 166,105 handwritten character images. Each image is annotated with the age and gender of the participant who provided the handwriting sample, ensuring diversity and representativeness in the dataset.

4.1.3 Spelling Correction Model Training Data

Bangla Dictionary Words: For training the spelling correction model, a dataset was compiled from a large collection of nearly 11,000 Bangla words [31], district names [32], and Bangla names from HuggingFace [33]. This dataset serves as a comprehensive resource for training and evaluating the spelling correction capabilities within the OCR pipeline.

4.2 Data Preprocessing

To ensure the datasets were in optimal condition for training machine learning models, several preprocessing steps² were performed. These steps are crucial for standardizing the datasets, enhancing image quality, and preparing them for partitioning into training, validation, and testing sets. The preprocessing details are divided based on the different datasets and models used.

¹The length includes single characters as well as modifiers.

²The preprocessing steps were essential for standardizing the datasets, enhancing image quality, and preparing them for training. These steps included handling noise, improving image clarity, resizing, and annotation to ensure the datasets were in the optimal format for model training and evaluation.

4.2.1 Image Datasets

Handwritten Grapheme Dataset and BanglaLekha-Isolated Dataset

- **Loading Images:** The images were loaded from Parquet files, each containing numerous grayscale images. Each row in the Parquet files consisted of an image ID and the corresponding flattened image data.
- **Enhancement and Inversion:** The loaded images underwent enhancement techniques to improve their quality and clarity. The images were then inverted to ensure a consistent background and foreground representation across all samples, which is crucial for effective feature extraction during training.
- **Resizing:** The enhanced and inverted images were resized to a uniform dimension of 28x28 pixels. Resizing the images helps to reduce computational complexity and memory usage during model training.
- **Saving Images:** The preprocessed images were saved using a standardized naming convention. Each image was labeled with its corresponding grapheme and Parquet file number, along with its image number within the file, in the format “*image-Label_parquetFileNo_imageNo*”. This ensures that the images are organized and easily accessible for training and evaluation.
- **Aggregation and Partitioning:** After preprocessing, the datasets were aggregated and partitioned into training, validation, and test sets based on predefined ratios. This step is essential to create informative subsets for the machine learning models.

Algorithm 1 Preprocessing with BanglaGraphemeDataset

Define a custom dataset class **BanglaGraphemeDataset** to handle image loading and transformations.

Load CSV files containing the dataset information.

for each image in the CSV file **do**

Load the image.

Resize the image.

Convert the image to a tensor.

Normalize the image using standard mean and standard deviation values.

end for

Create **DataLoader** objects for training and validation datasets.

Enable **batch processing** using the DataLoader objects.

Synthetically Generated Bangla Corpus and Collected 300 Data

- **Dataset Preparation:** Given its synthetic nature, the preprocessing of the Synthetically Generated Bangla Corpus required a unique approach. This involved preparing the dataset specifically for training the YOLO (You Only Look Once) model, a popular object detection architecture. For the collected data, background-noise removal and enhancement techniques were employed.
- **Annotation:** Random samples were selected from the corpus and character regions within the images were annotated using Roboflow, as can be seen from Figure 4.2, a tool commonly used for object detection. Another dataset was created using the 300 images from the collection sample. These annotations served as ground truth labels for training the YOLO model.
- **Partitioning and Formatting:** The annotations and images were divided into training, validation and test dataset, and formatted appropriately for YOLO model training. This included ensuring the annotations were in the correct format and the images met the required specifications for the model.

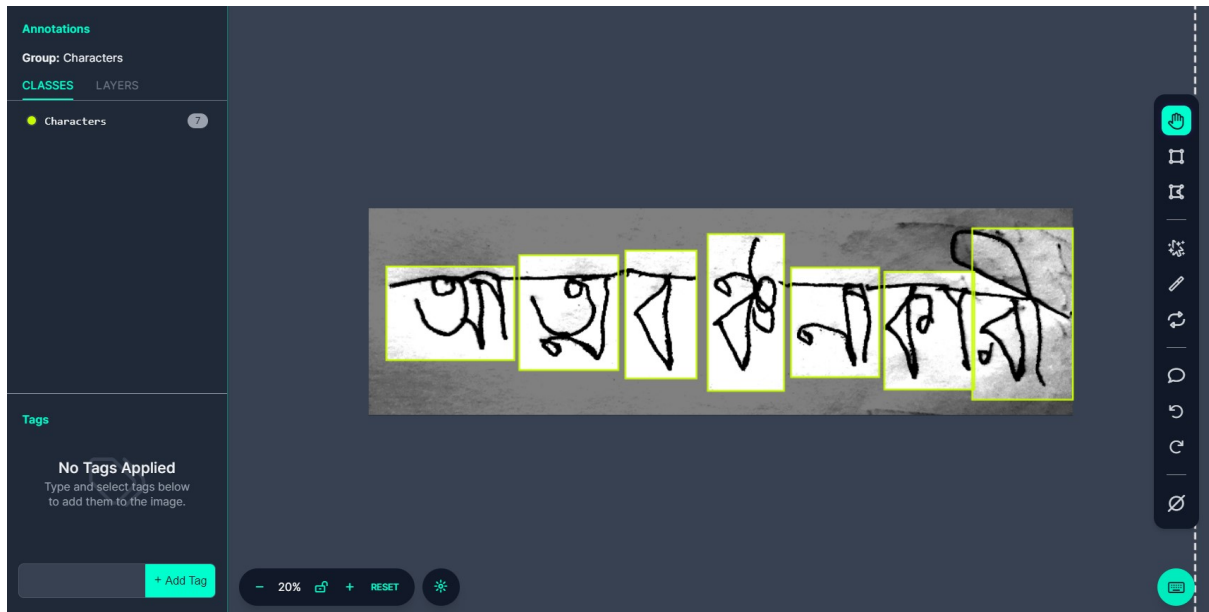


Figure 4.2: Example of annotations using Roboflow

4.2.2 Text Dataset

Spelling Correction Model

- **Data Cleaning:** For the spelling correction model, specific preprocessing steps were applied to the BengaliWordList_112 dataset. Words containing special char-

acters such as ‘\’, ‘-’, ‘ “ ’, and ‘,’ were excluded to maintain consistency.

- **Data Augmentation:** Names of districts and people were added to the list to increase the dataset’s comprehensiveness and ensure that the model could handle a wider range of inputs.
- **Saving Processed Data:** The cleaned and augmented list of words was then serialized and saved in a pickle file format for training. This ensures that the dataset is readily available for efficient loading and use in subsequent training phases.

Overall, these preprocessing steps ensured that the datasets were properly formatted and ready for use in training, validating, and testing the models. The careful attention to detail in preprocessing was vital for the subsequent experiments and the overall success of the machine learning models.

4.3 Training Process

The main components of the proposed pipeline, the detection model and the recognition model, are trained separately.

4.3.1 Detection Models

For the YOLOv8 models (small and medium):

Initial Training on Typed Data:

- The models were first trained on a synthetic word dataset to learn basic features and patterns relevant to single character detection. This step trained the model in isolating based alphabets and less complicated conjuncts.
- YOLOv8 models were optimized using AdamW optimizer with a learning rate of 0.002 and momentum of 0.9.
- Batch size was set to 16, and training was set for 300 epochs with early stopping enabled for patience of 50 epochs.

Fine-Tuning on Handwritten Word Images:

- After initial training, the best models from each YOLOv8 variant were fine-tuned on a handwritten word dataset.

- Fine-tuning aimed to adapt the models to recognize handwritten features and improve detection accuracy on real-world data. These data further enhanced the models detection capabilities with longer words, more complex compound character forms, and different unique writing styles.
- Early stopping was applied with a patience of 50 epochs to prevent overfitting and ensure optimal performance.

Impact of Synthetic-Noise Augmented Training:

- Synthetic-noise augmentation techniques were employed in parallel to non-augmented techniques during the final fine-tuning stage.
- These techniques helped in enhancing the robustness of the models against noise and variability in real-world handwritten images.

4.3.2 Recognition Models

Evaluation of Various Architectures:

- ResNet (18, 34, 50, 101, 152),
- GoogLeNet,
- DenseNet (121, 161, 169, 201),
- EfficientNet (b0-b7),
- EfficientNetV2 (s, m).

Each architecture was evaluated for its performance in a recognition task involving grapheme, vowel diacritic, and consonant diacritic classification.

Training Setup:

Hyperparameters such as batch size, learning rate, and optimizer were standardized across all models:

- 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.

Basic Training Steps

Model Definition

- **Class Definition:** A custom class, i.e. `EfficientNet` is defined by subclassing `nn.Module`.
- **Loading:** The pretrained model is loaded without pretrained weights and the final classification layer.
- **Feature Extraction:** Layers from the model are used except for the classifier, and a dropout layer is added.
- **Output Layers:** Three fully connected layers are added for predicting `grapheme_root`, `vowel_diacritic`, and `consonant_diacritic`.

Training Loop

- **Hyperparameters:** Essential parameters like epochs, `batch_size`, and learning rate are set.
- **Model Initialization:** The model is instantiated and moved to the appropriate device (CPU or GPU).
- **Optimizer and Scheduler:** An Adam optimizer and a learning rate scheduler (`ReduceLROnPlateau`) are initialized.
- **Loss Function:** The custom loss function '`loss_fc`' computes the average CrossEntropy loss across three outputs:

Algorithm 2 Custom Loss Function: `loss_fc`

```

0: function LOSS_FC(outputs, targets)
0:   out1, out2, out3  $\leftarrow$  outputs
0:   t1, t2, t3  $\leftarrow$  targets
0:
0:   loss1  $\leftarrow$  CrossEntropyLoss(out1, t1)
0:   loss2  $\leftarrow$  CrossEntropyLoss(out2, t2)
0:   loss3  $\leftarrow$  CrossEntropyLoss(out3, t3)
0:
0:   return (loss1 + loss2 + loss3)/3
0: end function
```

This algorithm calculates the average of three CrossEntropy losses, each computed between model outputs and corresponding targets (labels). It is used to optimize a model with multiple outputs (in this case, three outputs for grapheme_root, vowel_diacritic, and consonant_diacritic predictions).

Training and Validation The training process iterates over multiple epochs. In each epoch, there is a training phase and a validation phase.

- **Training Phase:** The model is set to training mode.
 - **Batch Processing:** Images and labels are loaded in batches, and predictions are made.
 - **Loss Calculation:** The loss for each output is calculated and combined.
 - **Backpropagation:** Gradients are computed, and the optimizer updates the model parameters.
 - **Loss Tracking:** The running loss is tracked for each epoch.
- **Validation Phase:** The model is set to evaluation mode.
 - **Batch Processing:** Validation images and labels are processed in batches.
 - **Loss Calculation:** Validation loss is computed for each batch.
 - **Loss Tracking:** The average validation loss is tracked, and early stopping is enabled according to the training setup.

Model Saving and Early Stopping

- **Best Model Saving:** The model state is saved if the validation loss improves.
- **Early Stopping:** The training process stops early if the validation loss does not improve for six consecutive epochs.
- **Model Saving at Last Epoch:** The model state is saved at the last epoch as **Last Model** regardless of validation performance.

4.4 Word2Vec Model Training Process

The Word2Vec model is a neural network-based approach designed to learn vector representations of words, and is widely used in natural language processing (NLP) tasks for its ability to generate high-quality word embeddings.

The Word2Vec model training process involves several key steps, as outlined below:

4.4.1 Data Preprocessing

The data preprocessing stage begins with the script `preprocess.py`, which prepares the raw text data for training:

- **Text Reading and Sentence Splitting:** The script reads a text file (`bangla_words.txt`) containing Bengali words and splits it into sentences using delimiters ('।' and '?.').
- **Sentence Cleaning:** Each sentence undergoes cleaning to remove newline characters (`\n`).
- **Serialization:** The cleaned sentences are saved as a list in a pickle file (`bn_words_corpus.pickle`) in the `data/` folder, preparing the data for model training.

4.4.2 Training Configuration

The training parameters for the Word2Vec model are configured in the script `train.py`:

- **Skip-gram (`sg`):** Determines the training algorithm (1 for skip-gram, 0 for CBOW).
- **Window Size (`window`):** Window size of 4 for word vectorization.
- **Vector Size (`vector_size`):** Specifies the dimensionality of the word vectors(=300).
- **Minimum Word Count (`min_count`):** Ignores words with fewer occurrences than specified, in this case minimum occurrence is 1.
- **Workers (`workers`):** Sets the number of threads(=8) to use during training.
- **Epochs (`iters`):** The model is trained over 100 iterations (epochs).
- **Sampling (`sample`):** Controls the downsampling of frequent words, here, sampling is set to 0.01.

4.4.3 Checkpointing and Training

The training process utilizes the configured parameters and optionally saves checkpoints using the `EpochSaver` callback after each epoch:

- The model is trained on the preprocessed Bengali text data loaded from `bn_words_corpus.pickle`.

- Checkpointing is controlled by the `checkpoint` flag in `train.py`, allowing for the model's state to be saved to the `./checkpoints/` directory after each epoch.
- After completing training, the trained Word2Vec model (`word2vec_new.model`) is saved in the `results/` folder.

4.4.4 Exporting Pretrained Weights

The script `export_weights.py` exports the pretrained word embeddings (weights) from the trained Word2Vec model:

- The script loads `word2vec_new.model` from `results/` and extracts the word vectors.
- The extracted word vectors are serialized as a pickle file (`pretrained_weights.pickle`) in the `results/` folder.
- This file contains the learned embeddings, ready for use in various natural language processing tasks.

This comprehensive process ensures that the Word2Vec model effectively learns meaningful representations of words based on their context in the Bengali text corpus, enabling accurate and context-aware language processing applications.

4.5 Evaluation Metrics

To comprehensively evaluate the performance of our OCR system, we employed a range of metrics tailored to both the detection and recognition tasks.

4.5.1 Detection Task (YOLO Model)

For evaluating the detection task of the YOLO model, the following metrics were used to assess the model's accuracy and reliability:

- **Precision:** This metric measures the proportion of correctly detected text regions among all detected regions. It is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

High precision indicates that the model produces a low number of false positives, meaning that most of the detected text regions are indeed correct.

- **Recall:** This metric measures the proportion of correctly detected text regions among all actual text regions in the dataset. It is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

High recall indicates that the model successfully detects most of the actual text regions, producing a low number of false negatives.

- **IoU:** This metric measures the overlap between the predicted bounding box and the ground truth bounding box. It is defined as:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

IoU is crucial for evaluating the accuracy of detected regions. Higher IoU values indicate better performance, as it signifies greater overlap between the predicted and actual bounding boxes.

These metrics collectively provide a comprehensive evaluation of the model's performance, ensuring that both the accuracy of detected regions and the quality of character-level detection are considered.

4.5.2 Metrics for Recognition

Performance of the EfficientNet-based recognition model was mainly critiqued using the following metrics:

- **Precision:** Measures the proportion of correctly recognized characters or words among all recognized characters or words.
- **Recall:** Measures the proportion of correctly recognized characters or words among all actual characters or words in the dataset.
- **F1 Score:** The F1 Score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The harmonic mean gives equal weight to both precision and recall. F1 Score is useful for assessing the overall performance of the recognition model, especially when there is an imbalance between precision and recall.

- **Accuracy:** Accuracy measures the proportion of correctly recognized characters or words among all characters or words in the dataset. It is defined as:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Population}}$$

Accuracy provides a straightforward measure of the model's overall correctness in recognizing characters or words. It is useful for evaluating the general performance of the recognition model across the entire dataset.

4.5.3 Overall Evaluation

To evaluate the overall performance of the hybrid OCR system, we combined the metrics for both detection and recognition tasks. The system's performance was assessed on its ability to accurately detect and recognize Bangla handwritten text from start to finish.

Precision and Recall

Precision and recall are crucial metrics for evaluating the performance of an OCR system. These measures the overall correctness of the OCR system in converting handwritten text images into digital text, accounting for the entire pipeline.

- **Precision::** Precision measures the proportion of correctly recognized words among all words that the system identifies as recognized. It indicates how many of the detected words are actually correct. It is given by:

$$\text{Precision} = \frac{\text{Number of True Positive Characters}}{\text{Number of True Positive Characters} + \text{Number of False Positive Characters}}$$

– Characters are detected and recognised from Words

High precision indicates that the model produces a low number of false positives, meaning that most of the recognized words are indeed correct.

- **Recall::** Recall measures the proportion of correctly recognized words among all actual words in the dataset. It reflects the system's ability to detect as many

relevant words as possible. It is given by:

$$\text{Recall} = \frac{\text{Number of True Positive Characters}}{\text{Number of True Positive Characters} + \text{Number of False Negative Characters}}$$

- Characters are detected and recognised from Words

High recall indicates that the model successfully detects most of the actual words, producing a low number of false negatives.

Error Rates

Error rates provide a measure of the OCR system's robustness and reliability by quantifying the frequency and types of errors made during text recognition. Key error rates include the Character Error Rate (CER).

- **Character Error Rate (CER):** CER measures the proportion of incorrectly recognized characters to the total number of characters.

$$\text{CER} = \frac{\text{Number of Substitutions} + \text{Insertions} + \text{Deletions}}{\text{Total Number of Characters}}$$

- **Substitutions:** Incorrectly recognized characters.
- **Insertions:** Extra characters added by the OCR system.
- **Deletions:** Characters omitted by the OCR system.

Bottleneck Identification

To ensure the OCR system operates efficiently, we conducted an in-depth performance analysis focusing on various time and throughput metrics. Identifying bottlenecks in the pipeline is essential for optimizing the system's performance.

- **Average Processing Time:** This metric measures the average time taken by the OCR system to process a single image from detection to recognition. It provides an overview of the system's speed and helps identify stages that might be causing delays.

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

By analyzing this metric, we can pinpoint stages within the pipeline that consume excessive time, indicating potential inefficiencies.

- **Average I/O Waiting Time:** This metric assesses the average time the system spends waiting for input/output operations, such as reading images from storage and writing recognized text to files. High I/O waiting time can suggest bottlenecks related to data access and storage performance.

$$\text{Average I/O Waiting Time} = \frac{\text{Total I/O Waiting Time}}{\text{Number of I/O Operations}}$$

Reducing I/O waiting time can significantly enhance the overall system throughput, particularly in scenarios with large datasets.

- **Average Throughput:** Throughput measures the number of images processed by the system per unit time. It is a crucial indicator of the system's capacity to handle large volumes of data efficiently.

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

Improving throughput involves optimizing various stages of the pipeline to process more images in less time, thereby increasing the system's overall efficiency.

- **Average Latency:** Latency refers to the delay from the time an image is input into the system until the final recognition result is output. It encompasses both processing and I/O delays.

$$\text{Average Latency} = \frac{\text{Total Latency for All Images}}{\text{Number of Images Processed}}$$

Lower latency is critical for real-time applications where quick turnaround is essential. Analyzing latency helps identify and address delays that affect the system's responsiveness.

This comprehensive evaluation ensures that the integrated approach effectively captures and processes textual information, optimizing accuracy, reliability, and practical applicability in real-world applications.

4.6 Data Collection

We collected a dataset of 300 word images to evaluate the performance of the overall pipeline for Bangla handwritten text recognition. These word images were sourced from various handwritten documents contributed by individuals at different educational stages, ensuring the presence of real-world noises typical of natural handwriting.

The dataset was utilized exclusively for evaluating the overall pipeline’s performance in detecting and recognising handwritten text. This focused approach allowed us to assess the models’ effectiveness in handling real-world variations and challenges present in handwritten Bangla documents.

4.7 Evaluation Procedures

4.7.1 Detection Models

For the detection task, we evaluated YOLOv8 models of varying sizes (small and medium) initially trained on typed data. The top-performing two models were selected and further fine-tuned using handwritten word images. Subsequently, these fine-tuned models were evaluated, and the best performing model among them was identified.

Next, the selected model from the previous step, trained on typed characters, underwent two additional fine-tuning phases. First, it was fine-tuned using synthetic-noise augmented images, followed by fine-tuning using images without noise. The performance of these two fine-tuned models was then compared to assess the impact of augmented data training on model performance. The best performing model from these evaluations was chosen for integration into the OCR pipeline.

4.7.2 Recognition Models

For the recognition task, we evaluated a variety of models including ResNet (18, 34, 50, 101, 152), GoogLeNet, DenseNet (121, 161, 169, 201), EfficientNet (B0 to B7), and EfficientNetV2 (s, m). Each model was evaluated using the previously mentioned metrics: Precision, Recall, F1 Score, and Accuracy. The model demonstrating the highest performance across these metrics was selected for integration into the OCR pipeline.

4.7.3 Control Measures

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

- Cross-validation: Ensuring 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 dataset for training, validation, and testing across all models to maintain fairness and comparability.

These procedures and control measures were critical in systematically evaluating and selecting the most effective models for each component of the OCR pipeline, ensuring optimal performance in real-world applications.

Chapter 5

Results and Observation

5.1 Demonstrating OCR System in Action

Some workig example of the pipeline is shown in Figure 5.1 and Figure 5.3

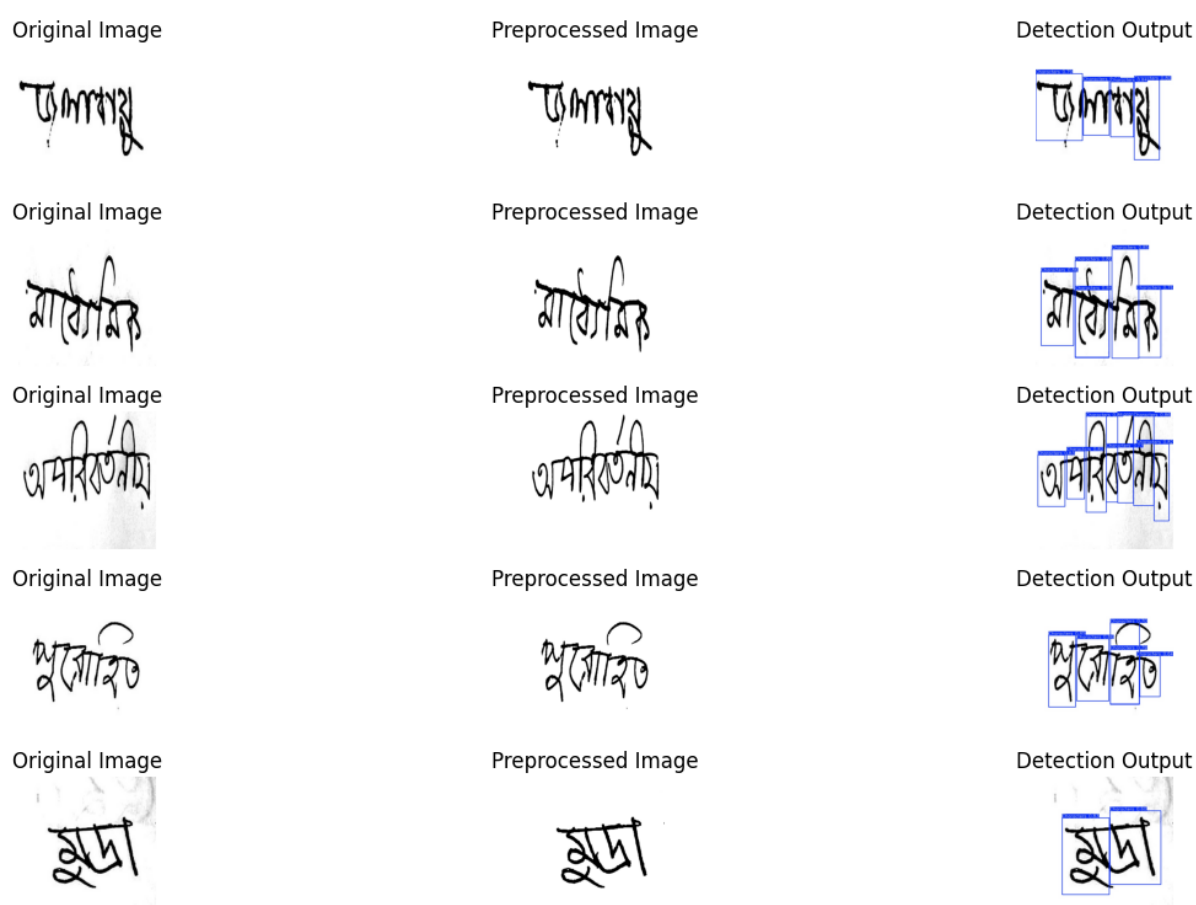


Figure 5.1: System detecting character bounding boxes in word images



Figure 5.2: System detecting character bounding boxes in word images

	Image Index	Recognized Text	Corrected Word
0	1	জলাবায়ু	জলবায়ু
1	2	মাধ্যমিক্ত	মাধ্যমিক
2	3	অপরিবর্তনীয়	অপরিবর্তনীয়
3	4	পুরোহিত	পুরোহিত
4	5	মুদ্রা	মুদ্রা
5	6	কয়েকটি	কয়েকটি
6	7	অজানা	অজানা
7	8	রাষ্ট্রীয়	রাষ্ট্রীয়
8	9	জীবন	জীবন
9	10	ক্ষমা	ক্ষমা

Figure 5.3: System recognizing texts from word images

5.2 Performance Analysis of Detection Model

5.2.1 Overview

The first step in our OCR pipeline is the detection of text regions using YOLOv8 models. We trained and evaluated different variants (small and medium) and fine-tuned the best performing one on handwritten data.

5.2.2 Results

Performance of YOLOv8 Models

Table 5.1 shows the average IoU, Precision, and Recall for each YOLOv8 model variant before and after fine-tuning on handwritten data.

Table 5.1: Performance of YOLOv8 Models

Model	Mean IoU (No FT)	Precision (No FT)	Recall (No FT)	Mean IoU (FT)	Precision (FT)	Recall (FT)
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

Comparison: Synthetic Noise Augmentation

The impact of synthetic noise augmentation on model performance is highlighted in Table 5.2, showing results of models trained with and without augmentation.

Table 5.2: Impact of Synthetic Noise Augmentation

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

5.2.3 Discussion

Analyze the results, highlighting key observations:

- **Impact of Fine-tuning on Handwritten Data:** Fine-tuning the YOLOv8 models on handwritten data significantly improves performance across all metrics. The mean IoU increases, indicating better accuracy in character detection. Precision and Recall also improve, suggesting fewer false positives and negatives, respectively.
- **Effect of Synthetic Noise Augmentation:** Introducing synthetic noise during training enhances the model’s robustness to variations in handwritten input. Augmented models show slightly better performance in mean IoU, Precision, and Recall compared to their non-augmented counterparts.

Based on these observations, the **Medium model fine-tuned on augmented handwritten data** emerges as the best-performing model for our OCR pipeline, balancing high metrics and robustness in text detection tasks.

5.3 Performance Analysis of Recognition Model

5.3.1 Overview

The recognition model is critical for accurately converting detected text regions into digital text. We evaluated various models including ResNet, GoogLeNet, DenseNet, and EfficientNet.

5.3.2 Results

Table 5.3: 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 5.4: 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

Table 5.5: Grapheme Metrics

Model	F1 (grapheme)	Precision (grapheme)	Recall (grapheme)
resnet18	0.9007	0.9142	0.8928
resnet34	0.9054	0.9131	0.9027
resnet50	0.9186	0.9250	0.9147
resnet101	0.9224	0.9282	0.9192
resnet152	0.9272	0.9339	0.9229
googlenet	0.9260	0.9361	0.9216
densenet121	0.9261	0.9334	0.9221
densenet161	0.9280	0.9358	0.9255
densenet169	0.9304	0.9367	0.9270
densenet201	0.9270	0.9383	0.9204
efficientnetb0	0.9256	0.9356	0.9187
efficientnetb1	0.9256	0.9324	0.9187
efficientnetb2	0.9227	0.9330	0.9129
efficientnetb3	0.9277	0.9356	0.9201
efficientnetb4	0.9276	0.9355	0.9201
efficientnetb5	0.9265	0.9347	0.9186
efficientnetb6	0.9283	0.9362	0.9206
efficientnetb7	0.9282	0.9371	0.9195
efficientnetV2S	0.9300	0.9374	0.9227
efficientnetV2M	0.9310	0.9390	0.9233

Table 5.6: Vowel Metrics

Model	F1 (vowel)	Precision (vowel)	Recall (vowel)
resnet18	0.9669	0.9714	0.9626
resnet34	0.9661	0.9672	0.9656
resnet50	0.9704	0.9778	0.9636
resnet101	0.9744	0.9774	0.9715
resnet152	0.9752	0.9793	0.9712
googlenet	0.9719	0.9733	0.9705
densenet121	0.9738	0.9735	0.9743
densenet161	0.9751	0.9791	0.9714
densenet169	0.9756	0.9790	0.9723
densenet201	0.9734	0.9772	0.9698
efficientnetb0	0.9756	0.9807	0.9708
efficientnetb1	0.9770	0.9789	0.9752
efficientnetb2	0.9756	0.9803	0.9712
efficientnetb3	0.9744	0.9795	0.9698
efficientnetb4	0.9763	0.9798	0.9726
efficientnetb5	0.9763	0.9782	0.9746
efficientnetb6	0.9749	0.9769	0.9729
efficientnetb7	0.9759	0.9803	0.9718
efficientnetV2S	0.9764	0.9786	0.9742
efficientnetV2M	0.9772	0.9803	0.9743

Table 5.7: Consonant Metrics

Model	F1 (consonant)	Precision (consonant)	Recall (consonant)
resnet18	0.9502	0.9522	0.9494
resnet34	0.9572	0.9659	0.9491
resnet50	0.9608	0.9709	0.9512
resnet101	0.9680	0.9777	0.9589
resnet152	0.9655	0.9647	0.9666
googlenet	0.9641	0.9789	0.9508
densenet121	0.9583	0.9644	0.9524
densenet161	0.9674	0.9693	0.9658
densenet169	0.9639	0.9707	0.9575
densenet201	0.9641	0.9741	0.9540
efficientnetb0	0.9641	0.9751	0.9536
efficientnetb1	0.9662	0.9708	0.9615
efficientnetb2	0.9641	0.9726	0.9558
efficientnetb3	0.9659	0.9712	0.9607
efficientnetb4	0.9712	0.9754	0.9672
efficientnetb5	0.9708	0.9713	0.9704
efficientnetb6	0.9696	0.9532	0.9865
efficientnetb7	0.9587	0.9696	0.9486
efficientnetV2S	0.9757	0.9757	0.9757
efficientnetV2M	0.9655	0.9655	0.9655

All the models have similar accuracy scores but the EfficientNetB4 has the lowest average loss(0.131) as seen from Table 5.4.

5.3.3 Discussion

- **Strengths and weaknesses of each model:**
 - ResNet models (e.g., ResNet50) show consistently high accuracy across all components (grapheme, vowel, consonant).
 - DenseNet models (e.g., DenseNet161) achieve high accuracy but may require more computational resources.
 - EfficientNet models (e.g., EfficientNetB4) offer a good balance between accuracy and loss.
- **Factors influencing the performance of recognition models:**
 - Model architecture: ResNet variants perform well due to their deep layer structure.
 - Training dataset size and quality: Larger, diverse datasets can improve model generalization.
 - Computational efficiency: EfficientNet models achieve good accuracy with fewer parameters.

Considering all these, we have chosen the EfficientNetB4 to add to our pipeline.

5.4 Overall System Evaluation

5.4.1 Pipeline Integration

The integration of the OCR system involved combining the best-performing detection and recognition models into a cohesive pipeline. YOLOv8, trained and fine-tuned on both typed and handwritten Bangla text, was selected for text detection. EfficientNetB4, which provided an optimal balance between accuracy and loss, was chosen for text recognition. This pipeline ensures robust detection and accurate recognition of Bangla handwritten characters, achieving high precision and recall rates.

5.4.2 Results

The performance metrics of the OCR system, shown in Table 5.8, both before and after correction, highlight its efficiency and accuracy.

Table 5.8: Overall Pipeline Metrics

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

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 5.9.

Table 5.9: 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

The overall performance metrics of the recognition model, after training with augmented data, are seen in Table 5.10.

Table 5.10: Overall Pipeline Metrics with recognition model trained on augmented data

With Spelling Correction	Average CER	Precision	Recall
No	0.0940	0.8016	0.8287
Yes	0.0195	0.9701	0.9584

Bottleneck in the Overall Pipeline

The bottlenecks were explored to highlight the areas for potential improvement.

Table 5.11: OCR Pipeline Processing Times and Bottlenecks

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

From Table 5.11, the bottleneck in term of processing time is the **Correction** model.

Table 5.12: 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

5.4.3 Discussion

The integration of YOLOv8 and EfficientNetB4 into the OCR pipeline has resulted in a system capable of highly accurate Bangla handwritten text recognition. The hybrid approach leverages the strengths of both models, ensuring robust detection and high-precision recognition.

- Benefits:

The system effectively increases accuracy while reducing loss, making it suitable for real-time applications. Significant improvements in CER and recognition metrics post-correction highlight the system's reliability.

- Challenges:

Handling diverse handwriting styles and varying document quality remains a challenge. Further optimization may be required to reduce processing time and enhance real-time performance.

- Impact of Augmented Data:

The results from Table 5.8 and Table 5.10 indicates that while the initial CER increased slightly due to the introduction of more challenging augmented data, the overall performance metrics post-correction show a marked improvement. 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.

Overall, the system showcases strong potential for practical applications in digitizing Bangla handwritten documents, with opportunities for further enhancement through continuous learning and dataset expansion.

5.5 Future Works

While the hybrid deep learning approach for Bangla handwritten OCR developed in this thesis has shown promising results, several areas remain open for further research and improvement. Future work can be directed towards addressing the following aspects:

1. Enhancement of Dataset Diversity and Size:

- **Data Collection:** There is a need for larger and more diverse datasets to train more robust models. Future research should focus on collecting and annotating extensive datasets that cover a wider range of handwriting styles, age groups, and regional variations in the Bangla script.
- **Synthetic Data Generation:** Developing methods for generating synthetic data that accurately reflect the variability found in real handwritten documents can augment training datasets and improve model generalization.

2. Improvement in Character Recognition Models:

- **Advanced Architectures:** Exploring more advanced deep learning architectures such as Transformers and attention mechanisms could further enhance the accuracy and efficiency of the recognition process.
- **Transfer Learning and Fine-Tuning:** Applying transfer learning from models pre-trained on larger, multilingual datasets and fine-tuning them on Bangla-specific data may yield better performance.

3. Robustness to Noise and Degradation:

- **Data Augmentation:** Implementing more sophisticated data augmentation techniques to simulate various noise and degradation conditions will help in making the models more robust and adaptable to real-world scenarios.
- **Image Enhancement:** Incorporating pre-processing steps like image denoising and enhancement can improve the quality of input data, thereby enhancing recognition accuracy.

4. Integration with Multilingual OCR Systems:

- **Multilingual Capability:** Extending the system to handle multiple languages, especially those that are commonly written alongside Bangla, will increase its utility. This involves creating a unified framework that can seamlessly switch between different scripts.
- **Cross-lingual Transfer Learning:** Leveraging cross-lingual transfer learning techniques can help in developing models that are efficient in recognizing multiple scripts with minimal additional training.

5. Real-time and Mobile Application Development:

- **Optimization for Real-time Use:** Optimizing the models for real-time processing can enable applications in mobile devices and scanners. This includes reducing the computational complexity and enhancing the speed of the detection and recognition pipelines.
- **Mobile App Development:** Developing user-friendly mobile applications that can leverage the OCR system for real-time document digitization and translation will broaden its practical applications and accessibility.

6. Enhanced Post-processing Techniques:

- **Contextual Post-processing:** Integrating more sophisticated post-processing techniques, such as contextual language models, can improve the accuracy of recognized text by considering the broader context of words and sentences.
- **Interactive Correction Tools:** Developing interactive tools that allow users to easily correct OCR results can significantly enhance user experience and overall system accuracy.

7. Transitioning from Character to Paragraph-level OCR:

- **Segmentation of Paragraphs:** Building on the current character recognition model, future research should focus on developing robust methods for segmenting and recognizing entire paragraphs. This involves enhancing the detection algorithms to accurately identify and isolate paragraphs from handwritten documents.
- **Sequential Text Recognition:** Implementing sequence-to-sequence models and recurrent neural networks can facilitate the transition from recognizing individual characters to entire paragraphs. These models can capture the context and dependencies between words, improving the overall coherence of the recognized text.
- **Text Layout Analysis:** Incorporating text layout analysis to understand the structure of handwritten documents, such as headings, paragraphs, and bullet points, will enable more accurate and contextually relevant text recognition.
- **Holistic Training Approaches** Adopting holistic training approaches that simultaneously train the models on character, word, and paragraph levels can improve the generalization and performance of the OCR system on diverse document structures.

By addressing these areas, future research can build upon the foundation laid by this thesis, pushing the boundaries of Bangla handwritten OCR technology and its applications. These advancements will not only improve the accuracy and efficiency of OCR systems but also contribute to the preservation and accessibility of Bangla written heritage.

References

- [1] ResearchGate,. “Speech bubbles detection/classification - scientific figure on researchgate.” https://www.researchgate.net/figure/YOLO-Architecture-from-the-original-paper-14_fig4_370979689. Accessed: 2024-06-21
- [2] “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-includes-an-additional-module-for-the-h_fig2_372207753. Accessed: 2024-06-21
- [3] “Mobilenet based apple leaf diseases identification - scientific figure on researchgate.” https://www.researchgate.net/figure/Neural-network-architecture-of-ResNet152-The-dotted-shortcuts-increase-dimensions_fig3_344004543. Accessed: 2024-06-21
- [4] “Chinese materia medica resource images screening method study.” Scientific Figure on ResearchGate, 2024. Available from: https://www.researchgate.net/figure/Typical-GoogLeNet-architecture_fig2_321867160
- [5] “An automated deep learning based anomaly detection in pedestrian walkways for vulnerable road users safety - scientific figure on researchgate.” https://www.researchgate.net/figure/DenseNet-Architecture_fig1_352398990, 2024. [accessed 21 Jun, 2024]
- [6] “Efficientnet [model architecture].”, June 20 2024
- [7] “Word2vec architecture.” ResearchGate, 2024. Accessed: 21 Jun, 2024
- [8] Raj, R. and Kos, A. “A comprehensive study of optical character recognition.” In “2022 29th International Conference on Mixed Design of Integrated Circuits and System (MIXDES),” pp. 151–154, 2022

-
- [9] Singh, A., Bacchuwar, K. and Bhasin, A. “A survey of ocr applications.” *International Journal of Machine Learning and Computing (IJMLC)*, 01 2012
- [10] Roy, A. “Handwritten bengali character recognition a study of works during current decade.” pp. 867–875, 07 2019
- [11] Biswas, M., Islam, R., Shom, G.K., Shopon, M., Mohammed, N., Momen, S. and Abedin, A. “Banglalekha-isolated: A multi-purpose comprehensive dataset of handwritten bangla isolated characters.” *Data in Brief*, volume 12:pp. 103–107, 2017
- [12] Rabby, A.S.A., Haque, S., Abujar, S. and Hossain, S.A. “Ekushnet: Using convolutional neural network for bangla handwritten recognition.” *Procedia Computer Science*, volume 143:pp. 603–610, 2018. 8th International Conference on Advances in Computing Communications (ICACC-2018)
- [13] Rabby, A.S.A., Haque, S., Islam, M.S., Abujar, S. and Hossain, S. *Ekush: A Multi-purpose and Multitype Comprehensive Database for Online Off-Line Bangla Handwritten Characters*, pp. 149–158, 07 2019
- [14] Roy, K., Hossain, M.S., Saha, P., Rohan, S., Rahman, F., Ashrafi, I., Rezwan, I.M., Hossain, B.M.M., Kabir, A. and Mohammed, N. “Synthetic Printed Words and Test Protocols Data for Bangla OCR.”, 6 2022
- [15] Boros, E., Nguyen, N.K., Lejeune, G. and Doucet, A. “Assessing the impact of ocr noise on multilingual event detection over digitised documents.” *International Journal on Digital Libraries*, volume 23, no. 3:pp. 241–266, September 2022
- [16] Wahid, M.F., Shahriar, M.F. and Sobuj, M.S.I. “A classical approach to handcrafted feature extraction techniques for bangla handwritten digit recognition.” In “2021 International Conference on Electronics, Communications and Information Technology (ICECIT),” pp. 1–4, 2021
- [17] Lin, J.H., Yang, Y., Gupta, R. and Tu, Z. “Local binary pattern networks.”, 2018
- [18] Ramanathan, R., Nair, A.S., Thaneshwaran, L., Ponmathavan, S., Valliappan, N. and Soman, K. “Robust feature extraction technique for optical character recognition.” In “2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies,” pp. 573–575, 2009
- [19] Alom, M.Z., Sidike, P., Hasan, M., Taha, T.M. and Asari, V.K. “Handwritten bangla character recognition using the state-of-art deep convolutional neural networks.”, 2018

- [20] Tounsi, M., Moalla, I., Pal, U. and et al.,. “Arabic and latin scene text recognition by combining handcrafted and deep-learned features.” *Arab J Sci Eng*, volume 47:pp. 9727–9740, 2022
- [21] Sonkusare, M., Gupta, R. and Moghe, A. “A review on character segmentation approach for devanagari script.” In A. Sheth, A. Sinhal, A. Shrivastava, and A.K. Pandey, editors, “Intelligent Systems,” pp. 181–189. Springer Singapore, Singapore, 2021
- [22] Ahmed, S. “Enhancing the character segmentation accuracy of bangla ocr using bpnn.” Technical report, BUBT, 2014. Accessed: 2024-06-21
- [23] Cunningham, P. and Delany, S.J. “k-nearest neighbour classifiers - a tutorial.” *ACM Computing Surveys*, volume 54, no. 6:p. 1–25, July 2021
- [24] Mor, B., Garhwal, S. and Kumar, A. “A systematic review of hidden markov models and their applications.” *Arch Computat Methods Eng*, volume 28:pp. 1429–1448, 2021
- [25] Kruse, R., Mostaghim, S., Borgelt, C., Braune, C. and Steinbrecher, M. *Multi-layer Perceptrons*, pp. 53–124. Springer International Publishing, Cham, 2022
- [26] Cristianini, N. and Ricci, E. *Support Vector Machines*, pp. 928–932. Springer US, Boston, MA, 2008
- [27] Alzubaidi, L., Zhang, J., Humaidi, A.J. and et al.,. “Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions.” *J Big Data*, volume 8:p. 53, 2021
- [28] Sherstinsky, A. “Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network.” *Physica D: Nonlinear Phenomena*, volume 404:p. 132306, March 2020
- [29] Addison Howard, Ahmed Imtiaz Humayun A.C.R.H.S.T. “Bengali.ai handwritten grapheme classification.”, 2019
- [30] Alam, S., Reasat, T., Sushmit, A.S., Siddique, S.M., Rahman, F., Hasan, M. and Humayun, A.I. “A large multi-target dataset of common bengali handwritten graphemes.” In “International Conference on Document Analysis and Recognition,” pp. 383–398. Springer, 2021
- [31] Kamal, M. “Bengali dictionary.” <https://github.com/MinhasKamal/BengaliDictionary>, 2024. GitHub repository
- [32] Chowdhury, A., HOSSEN, M.A. and Barua, A. “BD_DB₆₄.”, 2023

-
- [33] Faruk, M. “Bengali names vs gender dataset.” <https://huggingface.co/datasets/faruk/bengali-names-vs-gender>, 2024. Accessed: 2024-06-22

Appendix A

Codes

A.1 Sample Code

We use this code to train the recognition model...

```
1 import os
2 import pandas as pd
3 from PIL import Image
4 import torch
5 from torch import nn
6 from torch.utils.data import DataLoader
7 from torchvision import transforms, models
8 from torch.optim.lr_scheduler import ReduceLROnPlateau
9
10 # Dataset Class
11 class BanglaBornoDataset:
12     def __init__(self, img_H, img_W, dtype):
13         df = pd.read_csv(f"data/BanglaGrapheme/{dtype}.csv")
14         df = df[['image_id', 'grapheme_root', 'vowel_diacritic', 'consonant_diacritic']]
15
16         self.image_ids = df.image_id.values
17         self.grapheme_root = df.grapheme_root.values
18         self.vowel_diacritic = df.vowel_diacritic.values
19         self.consonant_diacritic = df.consonant_diacritic.values
20
21         self.width = img_W
22         self.height = img_H
23         self.t = dtype
24
25     def __len__(self):
26         return len(self.image_ids)
27
28     def __getitem__(self, item):
29         image_folder = f"data/BanglaGrapheme/{self.t}"
30         image_path = os.path.join(image_folder, f"{self.image_ids[item]}.jpg")
31         image = Image.open(image_path).resize((self.width, self.height)).convert('RGB')
32
33         transform = transforms.Compose([
34             transforms.ToTensor(),
35             transforms.Normalize((0.5,), (0.5,))
```



```

36         ])
37
38         image = transform(image)
39         return {
40             'image': image,
41             'grapheme_root': torch.tensor(self.grapheme_root[item], dtype=torch.long),
42             'vowel_diacritic': torch.tensor(self.vowel_diacritic[item], dtype=torch.long),
43             'consonant_diacritic': torch.tensor(self.consonant_diacritic[item], dtype=torch.
44             long)
45         }
46
47 # Model Class
48 class EfficientNet(nn.Module):
49     def __init__(self):
50         super(EfficientNet, self).__init__()
51         model = models.efficientnet_b4(weights=None)
52
53         self.features = nn.Sequential(
54             *(list(model.children())[:-1]),
55             nn.Dropout(p=0.2, inplace=True)
56         )
57         self.l0 = nn.Linear(model.classifier[1].in_features, 168)
58         self.l1 = nn.Linear(model.classifier[1].in_features, 11)
59         self.l2 = nn.Linear(model.classifier[1].in_features, 7)
60
61     def forward(self, x):
62         bs = x.size(0)
63         x = self.features(x).view(bs, -1)
64         return self.l0(x), self.l1(x), self.l2(x)
65
66 def loss_fc(outputs, targets):
67     out1, out2, out3 = outputs
68     t1, t2, t3 = targets
69
70     loss1 = nn.CrossEntropyLoss()(out1, t1)
71     loss2 = nn.CrossEntropyLoss()(out2, t2)
72     loss3 = nn.CrossEntropyLoss()(out3, t3)
73
74     return (loss1 + loss2 + loss3) / 3
75
76 def prepare_device():
77     return 'cuda' if torch.cuda.is_available() else 'cpu'
78
79 def create_dataloaders(img_H, img_W, batch_size):
80     datasets = {x: BanglaBornoDataset(img_H, img_W, x) for x in ['train', 'test', 'val']}
81     dataloaders = {x: DataLoader(datasets[x], batch_size=batch_size, shuffle=(x=='train')) for
82         x in ['train', 'test', 'val']}
83     return dataloaders
84
85 def save_model(model, path):
86     torch.save(model.state_dict(), path)
87
88 def train_model(num_epochs, model, device, dataloaders, optimizer, scheduler):
89     best_val_loss = float('inf')
90     early_stop_counter = 0
91
92     for epoch in range(num_epochs):
93         print(f'Epoch [{epoch + 1}/{num_epochs}]')
94         model.train()

```

```

93     running_loss = 0.0
94
95     for _, data in enumerate(dataloaders['train']):
96         image, targets = preprocess_data(data, device)
97
98         optimizer.zero_grad()
99         outputs = model(image)
100        total_loss = loss_fc(outputs, targets)
101
102        total_loss.backward()
103        optimizer.step()
104
105        running_loss += total_loss.item()
106
107    average_loss = running_loss / len(dataloaders['train'])
108    print(f'Training Loss: {average_loss}')
109
110    val_loss = evaluate_model(model, device, dataloaders['val'])
111    print(f'Validation Loss: {val_loss}')
112
113    if val_loss < best_val_loss:
114        best_val_loss = val_loss
115        early_stop_counter = 0
116        save_model(model, f'efficientnet_best.pth')
117    else:
118        early_stop_counter += 1
119
120    save_model(model, f'efficientnet_last.pth')
121
122    if early_stop_counter >= 5:
123        print("Validation loss hasn't improved for 5 epochs. Early stopping...")
124        break
125
126    scheduler.step(val_loss)
127
128    print('Training complete')
129
130 def preprocess_data(data, device):
131     image = data["image"].to(device, dtype=torch.float)
132     grapheme_root = data["grapheme_root"].to(device, dtype=torch.long)
133     vowel_diacritic = data["vowel_diacritic"].to(device, dtype=torch.long)
134     consonant_diacritic = data["consonant_diacritic"].to(device, dtype=torch.long)
135     return image, (grapheme_root, vowel_diacritic, consonant_diacritic)
136
137 def evaluate_model(model, device, val_loader):
138     model.eval()
139     val_loss = 0.0
140     with torch.no_grad():
141         for _, data in enumerate(val_loader):
142             image, targets = preprocess_data(data, device)
143             outputs = model(image)
144             val_loss += loss_fc(outputs, targets).item()
145     return val_loss / len(val_loader)
146
147 # Main Script
148 if __name__ == '__main__':
149     LEARNING_RATE = 1e-2
150     img_H, img_W = 128, 224
151     batch_size = 128

```

```

152     num_epochs = 100
153
154     device = prepare_device()
155
156     model = EfficientNet().to(device)
157     optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
158     scheduler = ReduceLR0nPlateau(optimizer, mode='min', factor=0.5, patience=5, verbose=True,
159                                   threshold=1e-4, cooldown=3)
160
161     if torch.cuda.device_count() > 1:
162         model = nn.DataParallel(model)
163
164     dataloaders = create_dataloaders(img_H, img_W, batch_size)
165
166     train_model(num_epochs, model, device, dataloaders, optimizer, scheduler)

```

A.2 Another Sample Code

We use this code to evaluate the recognition model...

```

1 import warnings
2 import torch
3 import torch.nn as nn
4 from torch.utils.data import DataLoader
5 from sklearn.metrics import f1_score, precision_score, recall_score
6 import torchvision.models as models
7 import dataset.create_charac_dataset2 as D
8 from mapping2 import grapheme_root_components, vowel_diacritic_components,
9   consonant_diacritic_components
10
11 warnings.filterwarnings("ignore") # Ignore DeprecationWarnings
12
13 # EfficientNet Model Class
14 class EfficientNet(nn.Module):
15     def __init__(self):
16         super(EfficientNet, self).__init__()
17         model = models.efficientnet_b4(weights=None)
18         self.features = nn.Sequential(
19             *(list(model.children())[:-1]),
20             nn.Dropout(p=0.2, inplace=True)
21         )
22         self.l0 = nn.Linear(model.classifier[1].in_features, 168)
23         self.l1 = nn.Linear(model.classifier[1].in_features, 11)
24         self.l2 = nn.Linear(model.classifier[1].in_features, 7)
25
26     def forward(self, x):
27         bs = x.size(0)
28         x = self.features(x).view(bs, -1)
29         return self.l0(x), self.l1(x), self.l2(x)
30
31 # Function to remove "module." prefix from state_dict keys
32 def remove_module_prefix(state_dict):
33     new_state_dict = {}
34     for k, v in state_dict.items():
35         new_state_dict[k[7:] if k.startswith('module.') else k] = v
36     return new_state_dict

```

```

36
37 # Loss Function
38 def loss_fc(outputs, targets):
39     loss1 = nn.CrossEntropyLoss()(outputs[0], targets[0])
40     loss2 = nn.CrossEntropyLoss()(outputs[1], targets[1])
41     loss3 = nn.CrossEntropyLoss()(outputs[2], targets[2])
42     return (loss1 + loss2 + loss3) / 3
43
44 # Load Model
45 def load_model(model_path, device):
46     model = EfficientNet().to(device)
47     state_dict = torch.load(model_path, map_location=device)
48     state_dict = remove_module_prefix(state_dict)
49     model.load_state_dict(state_dict)
50     return model
51
52 # Calculate Metrics
53 def calculate_metrics(true_values, pred_values):
54     return {
55         "f1_score": f1_score(true_values, pred_values, average='macro'),
56         "precision": precision_score(true_values, pred_values, average='macro'),
57         "recall": recall_score(true_values, pred_values, average='macro')
58     }
59
60 # Evaluate Model
61 def evaluate_model(model, test_loader, device):
62     model.eval()
63     total = len(test_loader.dataset)
64
65     g_acc, v_acc, c_acc = 0, 0, 0
66     acc1, acc2, acc3, total_loss = 0, 0, 0, 0
67
68     all_grapheme_true, all_grapheme_pred = [], []
69     all_vowel_true, all_vowel_pred = [], []
70     all_consonant_true, all_consonant_pred = [], []
71
72     for idx in range(total):
73         data = test_loader.dataset[idx]
74         img = data["image"].unsqueeze(0).to(device)
75         targets = (data["grapheme_root"].unsqueeze(0).to(device),
76                   data["vowel_diacritic"].unsqueeze(0).to(device),
77                   data["consonant_diacritic"].unsqueeze(0).to(device))
78
79         with torch.no_grad():
80             outputs = model(img)
81
82         total_loss += loss_fc(outputs, targets).item()
83
84         pred_grapheme = torch.argmax(outputs[0]).item()
85         pred_vowel = torch.argmax(outputs[1]).item()
86         pred_consonant = torch.argmax(outputs[2]).item()
87
88         all_grapheme_true.append(targets[0].item())
89         all_grapheme_pred.append(pred_grapheme)
90         all_vowel_true.append(targets[1].item())
91         all_vowel_pred.append(pred_vowel)
92         all_consonant_true.append(targets[2].item())
93         all_consonant_pred.append(pred_consonant)
94

```

```

95     g_acc += (pred_grapheme == targets[0].item())
96     v_acc += (pred_vowel == targets[1].item())
97     c_acc += (pred_consonant == targets[2].item())
98
99     if pred_grapheme == targets[0].item() and pred_vowel == targets[1].item() and
pred_consonant == targets[2].item():
100         acc3 += 1
101     elif (pred_grapheme == targets[0].item() and pred_vowel == targets[1].item()) or \
102          (pred_vowel == targets[1].item() and pred_consonant == targets[2].item()) or \
103          (pred_grapheme == targets[0].item() and pred_consonant == targets[2].item()):
104         acc2 += 1
105     elif pred_grapheme == targets[0].item() or pred_vowel == targets[1].item() or
pred_consonant == targets[2].item():
106         acc1 += 1
107
108     return {
109         "avg_g": g_acc / total,
110         "avg_v": v_acc / total,
111         "avg_c": c_acc / total,
112         "avg_acc1": acc1 / total,
113         "avg_acc2": acc2 / total,
114         "avg_acc3": acc3 / total,
115         "avg_loss": total_loss / total,
116         "metrics_grapheme": calculate_metrics(all_grapheme_true, all_grapheme_pred),
117         "metrics_vowel": calculate_metrics(all_vowel_true, all_vowel_pred),
118         "metrics_consonant": calculate_metrics(all_consonant_true, all_consonant_pred)
119     }
120
121 # Main Script
122 if __name__ == "__main__":
123     bs = 128
124     device = 'cuda' if torch.cuda.is_available() else 'cpu'
125
126     _, test_loader, _ = D.dividing_datasets(128, 224, bs)
127     model = load_model('efficientnetb4_noNum_128_best.pth', device)
128
129     results = evaluate_model(model, test_loader, device)
130
131     print(f'Column wise accuracy:\nGrapheme root: {results["avg_g"]*100:.2f}%, Vowel diacritic
: {results["avg_v"]*100:.2f}%, Consonant diacritic: {results["avg_c"]*100:.2f}%')
132     print(f'Overall accuracy:\nAcc1: {results["avg_acc1"]*100:.2f}%, Acc2: {results["avg_acc2"
]*100:.2f}%, Acc3: {results["avg_acc3"]*100:.2f}%')
133     print(f"Average loss: {results['avg_loss']:.4f}")
134
135     print(f'F1 Scores:\nGrapheme root: {results["metrics_grapheme"]["f1_score"]:.4f}, Vowel
diacritic: {results["metrics_vowel"]["f1_score"]:.4f}, Consonant diacritic: {results["
metrics_consonant"]["f1_score"]:.4f}')
136     print(f'Precision:\nGrapheme root: {results["metrics_grapheme"]["precision"]:.4f}, Vowel
diacritic: {results["metrics_vowel"]["precision"]:.4f}, Consonant diacritic: {results["
metrics_consonant"]["precision"]:.4f}')
137     print(f'Recall:\nGrapheme root: {results["metrics_grapheme"]["recall"]:.4f}, Vowel
diacritic: {results["metrics_vowel"]["recall"]:.4f}, Consonant diacritic: {results["
metrics_consonant"]["recall"]:.4f}')

```

Generated using Undergraduate Thesis L^AT_EX Template, Version 1.0. Department of Electrical
and Electronic Engineering, Bangladesh University of Engineering and Technology, Dhaka,
Bangladesh.

This thesis was generated on June 29, 2024 at 3:19am.

**B.Sc.
Engg.
EEE
BUET**

Bangla Handwritten OCR: A Hybrid Deep Learning Approach

**Aye Thein Maung
Sumaiya Salekin**

**June
2024**
