

group_1_200201015.pdf

by Mustafa Fahim

Submission date: 11-Jul-2024 06:42AM (UTC-0600)

Submission ID: 2415195456

File name: group_1_200201015.pdf (7.03M)

Word count: 19369

Character count: 115082

**BANGLA DIGIT RECOGNITION ON AIR WRITING USING
DEEP LEARNING TECHNIQUES**

By

- | | |
|-------------------------------|---------------|
| 1. Md. Yeamin Islam Sakib | ID: 200201015 |
| 2. Harishankar Barman | ID: 200201001 |
| 2. Musharrat Binte Alam Mithy | ID: 200201006 |

2

B.Sc. Engineering in Computer Science and Engineering



Department of Computer Science and Engineering

Faculty of Electrical and Computer Engineering

Bangladesh Army University of Science and Technology (BAUST)

JUNE 2024

Bangla Digit Recognition on Air Writing Using Deep Learning Techniques

²

This thesis is submitted in partial fulfillment of the requirement for the degree of B.Sc. Engineering in Computer Science and Engineering

By

- | | |
|-------------------------------|---------------|
| 1. Md. Yeamin Islam Sakib | ID: 200201015 |
| 2. Harishankar Barman | ID: 200201001 |
| 2. Musharrat Binte Alam Mithy | ID: 200201006 |

Supervised by

Hasan Muhammad Kafi

¹⁴

Assistant Professor, Department of CSE

Co-Supervised by

Md. Mahadi Hasan

²

Lecturer, Department of CSE

Department of Computer Science and Engineering

Bangladesh Army University of Science and Technology (BAUST)

Saidpur Cantonment, Nilphamari, Bangladesh

The thesis titled “**Bangla Digit Recognition on Air Writing Using Deep Learning Techniques**” submitted by Md. Yeamin Islam Sakib (ID: 200201015), Harishankar Barman (ID: 200201001) and Musharrat Binte Alam Mithy (ID: 200201006) session ¹⁶ 2020-2021 has been presented on 24 / 06 / 2024 and accepted as satisfactory in fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Engineering (CSE) as B.Sc. Engineering to be awarded by the Bangladesh Army University of Science and Technology (BAUST).

Board of Examiners

1. _____ Supervisor

Hasan Muhammad Kafi

Assistant Professor

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

2. ¹⁰ _____ Co-Supervisor

Md. Mahadi Hasan

Lecturer

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

3. _____ Member

Dr. S.M. Jahangir Alam

³⁵

Professor and Head

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

4. -----

Member

Dr. Engr. Mohammed Sowket Ali

Associate Professor

10

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

5. -----

Member

Md. Zahid Hassan

Lecturer

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

6. -----

Member

Md. Khalid Syfullah

Lecturer

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

Dr. S.M. Jahangir Alam

2

Professor and Head

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

CANDIDATE'S DECLARATION

It is hereby declared that the contents of this thesis are original and any part of it has not been submitted elsewhere for the award of any degree or diploma.

SN.	Student Name	ID	Signature & Date
1.	Md. Yeamin Islam Sakib	200201015	
2.	Harishankar Barman	200201001	
3.	Musharrat Binte Alam Mithy	200201006	

5 ACKNOWLEDGMENTS

First of all, we are grateful to almighty Allah who enabled us to complete this thesis successfully. Thereafter, our sincerest thanks and gratitude to our honorable thesis supervisor Hasan Muhammad Kafi, Assistant Professor, Department of Computer Science & Engineering (CSE) at Bangladesh Army University of Science Technology (BAUST), Saidpur, and our co-supervisor, Md. Mahadi Hasan, Lecturer, Department of Computer Science & Engineering (CSE) at BAUST, for their invaluable guidance, constructive criticism, continuous support, and inspiration that have stimulated us to complete our thesis. We also convey our special thanks to the Head of the Department of Computer Science & Engineering (CSE) and gratitude to all the respected teachers of the department of CSE, including other departments for their invaluable support and guidance. We would like to thank all of our friends and the staff of the department for their valuable suggestions and assistance. Finally, we would like to thank our parents and our family for their steady love and great support during our study.

ABSTRACT

95 Bangla digit recognition on air writing has become a significant area of research due to its potential applications in human-computer interaction, digital learning, and assistive technologies. Many studies have been carried out in this area, leading to notable breakthroughs and great accuracy. Although the results were promising, there are still certain limitations to be addressed. These include lack of diversity in the datasets, misclassification of similar looking characters. To address these limitations, we collected a dataset comprising approximately 5314 images from 200 participants, categorized into ten classes (0-9). Preprocessing steps, including resizing and data augmentation, were applied to enhance model robustness. Various deep learning models, including EfficientNetB0, InceptionV3, Inceptionresnetv2, Mobilenetv2, Resnet50, Resnet101, Vgg16, Vgg19 and Xception were utilized and fine-tuned for effective Bangla digit classification. Additionally early stopping and dropout (0.5) were employed during training as regularization the process and prevent overfitting. Furthermore models were evaluated on a validation dataset that contain 20% of the total images. Demonstrating high accuracy and strong performance metrics. It was found that VGG16 performed the best, with an accuracy of 98.91%, while MobileNet had the lowest accuracy, at 97.18%. Our findings contribute to the field of digit recognition on air written and lay the groundwork for future research in air-writing recognition systems.

Keywords: Air writing, Bangla digit recognition, Gesture-based Bangla digit classification, Gesture recognition.

120
CONTENTS

CANDIDATE'S DECLARATION	iii
ACKNOWLEDGEMENTS	iv
ABSTRACT	v
CONTENTS	vi
9 LIST OF FIGURES	ix
LIST OF TABLES.....	xi
CHAPTER 1.....	1
INTRODUCTION.....	1
1.1 Overview.....	1
1.2 Problem Statement	2
1.3 Motivation	2
1.4 Objectives	3
1.5 Social Impact	3
1.6 Issues Encountered.....	4
1.7 Stakeholders	4
1.8 Summary	5
CHAPTER 2	5
LITERATURE REVIEW	5
2.1 Overview	6
2.2 Related Works	6
2.3 Summary	11
CHAPTER 3.....	11
METHODOLOGY	11

3.1	Overview.....	76 12
3.2	Data Collection and Preprocessing.....	12
3.2.1	Data Sources	13
3.2.2	Data Collection Techniques.....	13
3.3	Preprocessing Steps	14
3.3.1	Data Preprocessing Procedure	15
3.4	Pre-trained Models.....	17
3.4.1	Xception	17
3.4.2	InceptionV3.....	18
3.4.3	Inceptionresnetv2	18
3.4.4	MobileNet	19
3.4.5	MobileNetV2	20
3.4.6	EfficientNetB0	21
3.4.7	Resnet50	22
3.4.8	Resnet101.....	22
3.4.9	Vgg16.....	23
3.4.10	Vgg19.....	24
3.5	Flow Diagram of Methodology	24
3.6	Summary.....	26
CHAPTER 4	26
IMPLEMENTATION	26
4.1	Overview.....	27
4.2	Tools & Technologies	27
4.2.1	Hardware	28
4.2.2	Software	28
4.2.3	Packages and Libraries	29
4.3	Dataset Collection	30
4.4	Dataset Preparation	32
4.4.1	Data Preprocessing	34

4.4.2 Data Augmentation	37
4.4.3 Data Splitting	38
4.5 Model Training	40
4.5.1 Experimental Setup	40
4.5.2 Load of Pre-trained Models.....	43
4.5.3 Model Configuration.....	45
4.5.4 Training Pre-trained Models	46
4.6 Summary	46
CHAPTER 5.....	47
RESULT AND ANALYSIS	47
5.1 Overview.....	48
5.2 Model Performance	48
5.3 Loss and Accuracy Curve	49
108 5.4 Result Analysis	55
5.4.1 Performance comparison	55
5.4.2 AUC-ROC Analysis	56
71 5.5 Summary	58
CHAPTER 6.....	59
CONCLUSIONS AND FUTURE WORKS	59
6.1 Conclusions	60
6.2 Future Recommendations	60
REFERENCES	62
APPENDIX A	66
APPENDIX B.....	76
BIOGRAPHY	79

LIST OF FIGURES

Figure 3.1:	Data preprocessing steps.....	15
Figure 3.2:	Loaded data.....	15
Figure 3.3:	Augmented data.....	16
Figure 3.4:	The Xception architecture.....	17
Figure 3.5:	The Inception-V3 architecture.....	18
Figure 3.6:	The Inceptionresnetv2 architecture.....	57 19
Figure 3.7:	The MobileNet architecture.....	20
Figure 3.8:	The MobileNetV2 architecture.....	20
Figure 3.9:	The EfficientNetB0 architecture.....	21
Figure 3.10:	The Resnet50 architecture.....	22
Figure 3.11:	The Resnet101 architecture.....	23
Figure 3.12:	The Vgg16 architecture.....	23
Figure 3.13:	The VGG19 architecture.....	24
Figure 3.14:	The flow diagram of methodology for this research.....	25
Figure 4.15:	Sample images from our dataset.....	33
Figure 5.16:	EfficientnetB0 model's loss and accuracy.....	49
Figure 5.17:	Inceptionv3 model's loss and accuracy.....	50
Figure 5.18:	Inceptionv3 model's loss and accuracy.....	51
Figure 5.19:	MobilenetV2 model's loss and accuracy.....	51
Figure 5.20:	Resnet50 model's loss and accuracy.....	52
Figure 5.21:	Resnet50 model's loss and accuracy.....	52
Figure 5.22:	Vgg16 model's loss and accuracy.....	53
Figure 5.23:	Vgg19 model's loss and accuracy.....	54
Figure 5.24:	Xception model's loss and accuracy.....	54
Figure 5.25:	Comparison between different models.....	55
		56

Figure 5.26: AUC-ROC curves for EfficientnetB0, InceptionV3 , InceptionResNetV2 , MobileNet models. 57

Figure 5.27: AUC-ROC curves for Mobilenet, Resnet50, Resnet101, Vgg16, Vgg19, Xception models. 58

86

LIST OF TABLES

Table 1.1:	List of Bangla Digits and their corresponding English counterparts. . .	1
Table 2.2:	An overview of the reviewed literature.....	10
Table 4.3:	No. of images for collected Bangla digit.....	32
Table 5.4:	Performance metrics of various Deep learning models.....	49 139

CHAPTER 1

Introduction

1.1 Overview

Bangla, or Bengali, used as an official language in Bangladesh and the second one in India, by the number of people who use it as their first language. Out of over 230 million speakers in the world, Bangla is counted among the most popular languages being used today [1]. Bangla digits are used in the Bangla numerals which is in use in the present day Bangladesh and some of the part of India [2]. Gesture recognition can be considered as a more distinct approach than the mere traditional ways of using knobs and buttons as it is based on detecting the users' intentions from their actions [3].

The Bangla digits and their English counterparts are shown in the Table 1.1. Writing

Table 1.1: List of Bangla Digits and their corresponding English counterparts.

English Digits	Bangla Digits
0 (Zero)	০ (শূন্য)
1 (One)	১ (এক)
2 (Two)	২ (দুই)
3 (Three)	৩ (তিনি)
4 (Four)	৪ (চারি)
5 (Five)	৫ (পাঁচ)
6 (Six)	৬ (ছয়)
7 (Seven)	৭ (সাত)
8 (Eight)	৮ (আট)
9 (Nine)	৯ (নয়)

12

in the air is a process to write something in a 3D space by using gestures or trajectory

information, allowing users to write in a touchless system [4]. Bangla digit not the same with other symbols or numbers used in English language.

1.2 Problem Statement

The increasing number of applications requiring accurate and efficient digit recognition systems is driving the demand for digit recognition technology. These include, among other things, digital interfaces, security systems, and automated data entry. A shortage of Bangla digit datasets, however, poses an important challenge to this field and impedes the advancement of researchers and scientists who are developing Bangla digit recognition systems [2]. Another challenge in air-writing recognition is the non-uniformity of characters and the variability in writing styles, which complicates the consistent recognition of trajectories accurately [4]. There is an additional layer of complexity involved in accurately tracking small objects like the fingertip in images, especially when using webcams or standard laptop cameras. Identifying the writing hand pose, precisely identifying and tracking the fingertip, and identifying the air-writing character trajectory are the main challenges in this context [5]. Furthermore, air-writing requires continuous motion data with no discernible boundaries between writing and non-writing movements, so a strong system that can handle both detection and recognition tasks is required [6].

1.3 Motivation

Air-writing is fundamentally different from traditional handwriting because it doesn't involve writing on a surface and does not provide tactile feedback. This difference can lead to more variability and complexity in the motion data captured [3, 7]. Air writing technology enables touchless interaction, making it highly applicable in situations where traditional writing methods are challenging [4]. The motivation for this work is driven by the growing need for natural human-computer interaction systems due to the rise of virtual and augmented reality [8]. Also this technology carries the potential to assist individuals with speech challenges. Many individuals who struggle with speech are still capable of writing, and this application could be harnessed to track hand move-

ments and convert them into synthesized human speech [3]. Writing with a fingertip in the air, using controller-free tracking systems, offers an innovative interface for text input when conventional devices are unavailable [6]. Traditional input methods are being replaced by more intuitive methods. Developing a reliable air-writing recognition system addresses the limitations of existing hand gesture inputs, providing a more seamless and efficient way to interact with technology [8].

1.4 Objectives

Our thesis objectives are derived from the limitations of the existing thesis. These objectives are:

1. Create a diverse dataset of 5314 Bangla digit images contributed by 200 participants.
2. Train and Fine-tune different pre-trained models (EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18]) on the dataset.
3. Compare the performance of the aforementioned deep learning models on our newly build dataset to determine which is most effective for Bangla digit classification on air writing.

1.5 Social Impact

The development of air-written Bangla digit image classification systems has the potential to create significant social impacts. Primarily, it enhances accessibility for individuals with disabilities by enabling interaction with computers through simple hand gestures, serving as an alternative to traditional input devices [5, 6]. This technology can greatly improve the quality of life for people with motor impairments, providing greater independence in their digital interactions [12]. Additionally, air-written Bangla digit recognition can be integrated into educational tools to create interactive and engaging learning experiences. It helps young learners practice and learn Bangla numerals in a fun and intuitive way, promoting better retention and understanding [8]. In regions with limited access to advanced technology, this system offers an affordable and

intuitive way to interact with digital devices, helping to bridge the digital divide and foster digital inclusion [9] Furthermore, the integration of air-writing recognition into applications can lead to innovative human-computer interaction experiences, such as natural and immersive interactions ¹²⁶ in virtual reality (VR) and augmented reality (AR) environments. Implementing this technology also contributes to the preservation and promotion of the Bangla language in the digital age by facilitating its use in modern technology and supporting cultural heritage initiatives [4, 5].

1.6 Issues Encountered

These difficulties were resolved through research, expert consultation, and code customization. During the dataset collection phase, several challenges were encountered that affected the overall process. One significant issue was the difficulty in effectively communicating the instructions to the participants. Many participants struggled to understand how to write in the air in front of the camera ¹², which led to inconsistencies in the collected images. Ensuring that each participant correctly performed the air writing task required additional time and effort. Another problem arose during the actual writing process. Participants often faced interruptions while writing, causing them to stop mid-way frequently. This disruption made it difficult to capture continuous and clear images of the air-written digits. Additionally, some participants had trouble maintaining consistent writing, leading to the need for repeated attempts to obtain usable images. To address this, clearer instructions and live demonstrations of the air writing process were provided. Additional difficulties have arisen. For example, confronting problems similar to those encountered while learning machine learning coding that were previously unfamiliar to us. Attempting to run code on several platforms has caused major delays and trouble.

1.7 Stakeholders

1. **Researcher:** Principal investigator responsible for the design, execution, and analysis of the study. Developed the data collection script, gathered and processed the dataset, trained machine learning models, and compiled results into

the thesis.

2. **Persons with Disabilities:** For people with disabilities, improved air writing recognition technology can facilitate communication and increase their access to digital content, fostering greater independence and inclusion.
3. **Education Sector:** Better methods for air writing Bangla numeral recognition and digitization can improve instructional materials and accessibility for pupils, including those with disabilities.

1.8 Summary

This chapter explains the importance of Bangla as a major language and the particular difficulties in reading handwritten numbers in Bangla, particularly when using air writing techniques. It outlines the problem statement, highlighting the need for efficient and accurate Bangla digit recognition systems due to limited existing datasets and the growing demand for advanced HCI technologies. The chapter discusses the motivation behind using air-writing technology for Bangla digit recognition, emphasizing its potential to enhance human-computer interaction and its applications in education and assistive technologies. The objectives include creating a comprehensive dataset from 200 participants and evaluating various pre-trained models such as EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18] for classifying Bangla digit on air writings. It also addresses the social impact of this research, including improved accessibility, interactive learning, and cultural preservation. Challenges encountered during the dataset collection process, such as participant instructions and maintaining writing consistency, are also discussed. The chapter concludes with an overview of the thesis organization.

CHAPTER 2

Literature Review

2.1 Overview

Bangla digit recognition in air-written input utilizes deep learning and computer vision to interpret hand gestures in free space as characters, offering significant potential for accessibility and interactive applications. Accurate character and digit recognition in air-written input is greatly increased by using sophisticated models such as custom ¹²¹ deep learning models like Long short-term memory (LSTM), Dynamic Time Warping (DTW) ¹³² and Support Vector Machines (SVM) have been used, enhancing human-computer interaction and benefiting virtual reality, educational tools, and accessibility technologies [2, 4, 5].

2.2 Related Works

Numerous studies ¹ have been carried out in the field of air writing. For instance, Mahmud et al. ¹²⁸ [5] conducted a study on the recognition of English capital alphabets written in the air using depth information from a depth camera. The research involved capturing hand finger movements and representing them as depth images to extract ¹²⁵ trajectory data, which were then converted into time-series data. These data were classified using ¹⁰⁰ Dynamic Time Warping (DTW) distances and a multi-class Support Vector Machine (SVM), leveraging depth information to enhance the accuracy of air-writing gesture recognition. The methodology of the study included several key techniques. ⁵³ For data collection, two sets of datasets were prepared: one without depth information (390×260) and the other with depth information (390×312), along with their re-sampled ²³ versions. Additionally, three sets of datasets containing features related to correlation

analysis were prepared, resulting in a total of 12 datasets for evaluation. The dataset was split into k-folds for cross-validation (2-Fold, 5-Fold, 10-Fold, and 15-Fold) to train the model on all samples except the (k1) folds and evaluate it on the remaining fold, repeating this process k-times to record the average accuracy. Hand gestures were captured using a Microsoft Kinect depth camera, yielding a dataset of uppercase English letters at 640×480 pixels, 30 fps. Pre-processing included hand segmentation via depth information, extracting x, y coordinates for movement tracing, and applying a moving average filter for smoothing jerky motions. They used feature selection to avoid overfitting in machine learning, using Matlab and Weka with 155 DTW distance features. They checked accuracy using metrics like TPR, FPR, precision, recall, F-measure, ROC area, and PRC area, achieving 96.85% accuracy with Support Vector Machine classification. The study's limitations include a dataset created by only 15 users for air-writing recognition, possibly not reflecting broader handwriting or gesture styles. It focused solely on depth-based features and point vectors, neglecting other potentially informative feature types used in gesture recognition. Additionally, using only Support Vector Machine (SVM) without comparing it to other models or ensemble methods might have limited accuracy and robustness in classification tasks. However, the study urges further research with diverse datasets, more feature comparisons, and testing of different classifiers to enhance reliability and applicability of the findings.

In reference [2] Chayti Saha et al. conducted a study titled "Real-time Bangla Digit Recognition through Hand Gestures on Air Using Deep Learning and OpenCV". This research explores the real-time recognition of Bangla numerals through hand gestures, utilizing a Convolutional Neural Network (CNN) model for this purpose. The study addresses a significant gap in the field, as previous research on Bangla handwritten digit recognition had not successfully implemented real-time recognition using hand gestures. The methodology involves using the BanglaLekha-Isolated dataset, which contains a comprehensive set of handwritten Bangla characters. The authors preprocess the dataset through steps such as resizing, gray scaling, and augmentation. These preprocessed images are then fed into a CNN model designed to predict Bangla digits based on hand movements captured by a camera using OpenCV. The CNN was trained over 34 epochs, and the training and validation losses and accuracies were recorded.

1

The results demonstrate the system's effectiveness, achieving a 98.37% accuracy on the BanglaLekha-Isolated dataset. Additionally, without augmentation, the model reached 3 a training accuracy of 99.36%, a validation accuracy of 97%, and a test accuracy of 97.30%. However, the study also highlights some limitations. Despite high training accuracy, the model struggled with recognizing enhanced real-time inputs, indicating potential overfitting. This suggests the need for further refinement in the model or improvements in data preprocessing techniques. Three primary limitations have been found: the size of the dataset used, which may constrain the model's generalization and 130 performance on unseen data; variability in hand gestures, as differences in hand size, writing styles, and gestures among individuals could impact the model's accuracy; and limited computational resources, with the model implemented on an integrated computer with limited capabilities, potentially affecting its scalability and real-time performance on more resource-intensive platforms. The study successfully developed a real-time Bangla digit recognition system using hand gestures, setting a precedent for future research in this domain.

78

In study [4] Alam et al. present a trajectory-based air-writing recognition system using deep neural networks and depth sensors, focusing on recognizing characters written in free space. The methodology consists of four main parts: fingertip detection, data collection, 12 normalization, and network design. Fingertip detection is performed using an 12 Intel RealSense SR300 camera, which tracks the index fingertip for trajectory writing. Data is collected through an interactive UI from 10 participants aged 23 to 30, recording spatial trajectory sequences of written digits. Normalization techniques address challenges like shaky trajectories, with deep learning models (LSTM and CNN) eliminating the need for manual feature selection. The dataset comprises 21,000 digit trajectories, validated on both the self-collected RTD dataset and the 6D motion gesture (6DMG) dataset, achieving accuracies with LSTM of 99.17% and 99.32%, respectively, when normalized, and between 98.68% to 99.08% without normalization. The CNN model achieved 99.06% accuracy on both datasets with normalization, and between 98.26% to 98.89% without normalization. The LSTM model outperforms CNN, particularly with normalized data, showing superior performance across various conditions. The system's limitations include potential misclassification due to similar-looking characters

and reliance on RGB webcams without depth information, which can impact finger-tip detection accuracy. Additionally, the system is primarily suited for gestures where the finger and palm lie in the same plane, with performance degrading when the fingertip points toward the camera. The study underscores the potential of deep learning in air-writing recognition, providing a practical and efficient solution while suggesting future work to enhance robustness, such as incorporating 3D hand pose estimation techniques. This innovative system offers significant contributions to the field, including a new dataset and improved recognition accuracy, highlighting the effectiveness of LSTM networks in trajectory-based air-writing recognition .²¹

Research by Mukherjee et al. [8] studied the detection and tracking of fingertips for recognizing air-writing in videos. This study focuses on detecting and tracking fingertips for recognizing air-writing in videos. The authors address the challenge of accurate and robust detection and tracking of fingertips, which are often small and difficult to detect, using web-cam video inputs. The methodology involves several key steps:
²¹ 1 Hand Pose Detection utilizing the Faster R-CNN framework for precise hand detection, Hand Segmentation by segmenting the detected hand and counting the number of raised fingers based on geometrical properties, Fingertip Detection and Tracking employing a new signature function called distance-weighted curvature entropy, and Termination Criterion using a fingertip velocity-based termination criterion to mark the end of the air-writing gesture. The techniques used include Faster R-CNN for hand detection, geometrical analysis for finger counting, distance-weighted curvature entropy for fingertip detection and tracking, and a fingertip velocity-based criterion for gesture termination. The dataset consists of web-cam videos capturing mid-air finger writing, used to evaluate the proposed methodologies for hand pose detection, fingertip detection, and tracking. The results showed that the proposed fingertip detection and tracking algorithm achieved a mean precision of 73.1%, with real-time performance at 18.5 frames per second (fps). Character recognition experiments resulted in a mean accuracy of 96.11%, comparable to existing handwritten character recognition systems. Specific models achieved the following accuracies: Inception-v4 Accuracy: 98.12%, Inception-v3 Accuracy: 97.96%, ResNet Accuracy: 97.29%, VGG-16 Accuracy: 96.75%, and AlexNet Accuracy: 96.11%, with Inception-v4 achieving the highest accuracy and

AlexNet the lowest. The study acknowledges challenges in the initialization and termination of mid-air finger writing due to the absence of standard delimiting criteria, and although the precision of fingertip detection has improved, it still faces difficulties due to the small dimensions of the fingertip. Table 2.2 has been created from the summary of the literature review.¹⁰⁹

Table 2.2: An overview of the reviewed literature.

Reference	Dataset	Method	Result	Limitation
Mukherjee et al [8].	Class: 36 Image: 11800	Faster R-CNN, DWCE	Precision: 73.1%, Real-time: 18.5 fps, Char. recognition: 96.11%, Incep-v4: 98.12%, Incep-v3: 97.96%, ResNet: 97.29%, VGG-16: 96.75%, AlexNet: 96.11%	Challenges in initialization and termination, Precision affected by small fingertip size
Alam et al [4].	Class: 10 Total Images: 21000	LSTM,CNN	-LSTM normalization: 99.17%, without normalization: 98.68% - 99.08%, -CNN with normalization: 99.06%, without normalization: 98.26% - 98.89%	Misclassification of similar-looking characters, Accuracy impacted by RGB webcam limitations, Reduced performance when fingertip faces the camera
Chayti Saha et al [2].	Class: 10 Training image : 13,748 Testing image: 4000 validation image: 200	CNN model	Test Accuracy: 97.30%	Dataset size, Variability in hand gestures, Limited computational resources may affect generalization and real-time performance
Mahmud et al [5].	Class: 26 Total image :13950	SVM with DTW distance	Accuracy 96.85%	Small dataset , Limited to depth-based features

This research offers an effective framework for detecting and tracking fingertips in air writing, significantly enhancing precision and real-time performance, with promising practical applications in human-computer interaction.

2.3 Summary

The second chapter includes a comprehensive assessment of relevant literature, focusing on research that has examined Bangla digit recognition on air writing. The chapter methodically reviews the methodologies used in these investigations, emphasizing their strengths and shortcomings. This study provides a strong foundation for understanding the current state of research in the field of Bangla digit recognition on air writing. Key themes and findings from the literature highlight the gaps that this thesis aims to address, setting the stage for the following chapters.

103
CHAPTER 3

Methodology

3.1 Overview

This chapter describes the steps involved in air writing Bangla digit recognition, including data collection, preprocessing, model development, and evaluation. Using cameras, Bangla digit air writing images were taken for data collection. A dataset containing thorough descriptions of sample numbers and participant demographics was then assembled. Normalization for image standardization and augmentation for dataset diversity were two examples of data preprocessing. The process of developing a model involved choosing an algorithm, training with preprocessed data, and assessing performance with metrics. The validity and reliability of the study's conclusions are guaranteed by this methodical approach.

3.2 Data Collection and Preprocessing

For this study, a comprehensive dataset was collected, consisting of 5,314 images categorized into 10 classes representing the Bangla digits from 0 to 9. The air writing gestures were performed in controlled environments and captured using advanced sensors and high-resolution cameras to ensure high-quality data. The dataset includes diverse writing styles and variations, providing a robust foundation for training and evaluating the classification model.⁶ This extensive collection of images contributes significantly to the reliability and validity of the study's findings. The gathered dataset was carefully preprocessed in order to remove noise and standardize the images. Comprehensive data augmentation methods were also used to improve the model's resilience and capacity for generalization.¹¹⁷

3.2.1 Data Sources

The dataset was gathered from 200 participants selected from BAUST, ensuring a diverse range of writing styles and variations. Among these participants, 70 were female and 130 were male, coming from various backgrounds. The air writing gestures were performed in controlled environments and captured using advanced sensors and high-resolution camera (Logitech C270 720p Hd Webcam) to ensure high-quality data. This robust dataset, consisting of 5,314 images categorized into 10 classes representing the Bangla digits from 0 to 9, provides a solid foundation for training and evaluating the classification model, contributing significantly to the reliability and validity of the study's findings.⁶

3.2.2 Data Collection Techniques

1. **Participant Recruitments:** Gather a diverse group of approximately 200 participants to ensure variability in handwriting styles. Participants were recruited from BAUST. Recruitment ensured a mix of different genders and backgrounds for comprehensive data collection. Among the 200 participants, 70 were women, and the remaining were men.
2. **Setting Up the Environment**
 - (a) **Hardware Requirements:** Data collection for the Bangla digit image classification thesis was conducted through air writing using a customized Python [19] script called "Airwriting.py". The script recorded air writing movements using the built-in cameras of two different laptops, HP laptop and an ASUS TUF F15. The laptop's camera was set to record video input automatically when the "Airwriting.py" script ran.
 - (b) **Software Requirements:** The "Airwriting.py" script was used to capture images of airwritten Bangla digits. This script was developed in a Python [19] environment with several necessary libraries installed, including OpenCV [20] for real-time computer vision tasks, NumPy [19] for numerical operations, and other standard libraries required for video processing and image manipulation. The combination of these software tools ensured the accu-

rate detection and capture of digits written on air during the data collection process.

3. **Data Collection:** To begin capturing digits written on air for analysis, first ensure we initialize the necessary components. This involves loading essential libraries like OpenCV [20] and NumPy [19], crucial for image processing and data manipulation tasks. Additionally, set up the camera to seamlessly capture video feeds, ensuring clarity and consistency in the digit recognition process. Once initialized, the capturing process can commence effectively. Participants are prompted to air-write each digit (0-9) multiple times in front of the camera. This process involves executing the "Airwriting.py" script within the Python environment, where participants enter their ID to begin. They follow on-screen instructions to position their hand correctly and execute the airwriting motion. Frames are captured when the digit is clearly visible, with options to save each frame by pressing 's' or quit the process with 'q'. Each captured frame is then extracted and saved as an individual image, properly labeled for future analysis or machine learning model training. This structured approach ensures a comprehensive dataset for accurate digit recognition research or application development.
4. **Organizing the Data:** To organize air-written digit images, first create a folder for each person. Inside each person's folder, make separate folders for digits 0 through 9. When capturing images, save them directly into the correct digit folders. Name each image using the format "digit-label-image_number," ensuring each file is named with the digit written, its label (like "0" for the digit '0'), and a number to sequence them. This method ensures clarity and easy sorting of the images for future reference or analysis.

3.3 Preprocessing Steps

15

Data preprocessing is a crucial step in preparing the raw collected dataset for training a machine learning model. Images were preprocessed using Keras [21] ImageDataGenerator [22]. Figure 3.1 illustrates a process for preparing raw images for inclusion in a dataset. For training, augmentations included rescaling, random shear, zoom, brightness and contrast adjustment, noise injection, random cropping, and elastic distortion

to increase dataset diversity. Validation data was only rescaled for normalization. Both

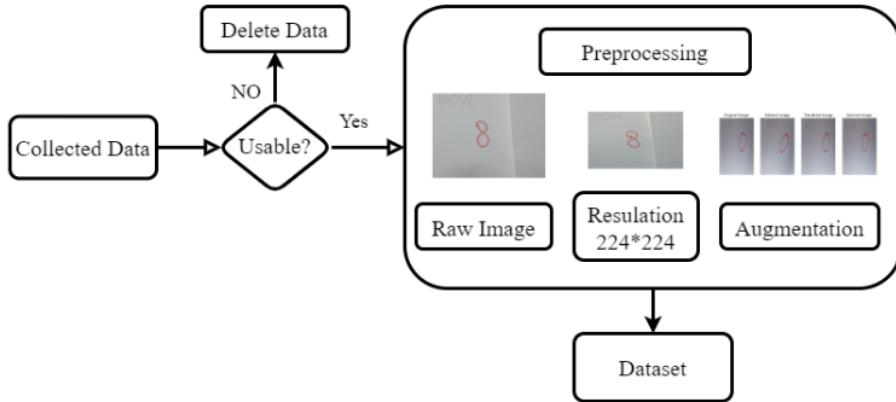


Figure 3.1: Data preprocessing steps.

datasets were resized to 224x224 pixels to ensure uniform input size for the neural network.

3.3.1 Data Preprocessing Procedure

- 1. Import Necessary Libraries:** Import essential libraries such as TensorFlow [23], Keras [21], NumPy [19], and OpenCV [20] for handling image data and model training.
- 2. Load and Inspect Data:** Load the collected dataset from its directory. Inspect a few samples to understand the structure and quality of the data.



Figure 3.2: Loaded data.

3. **Image Resizing:** Resize all images to a uniform size of 224x224 pixels. This ensures that the input dimensions are consistent, which is crucial for feeding the images into the neural network.
4. **Data Augmentation:** To enhance training data diversity and improve model generalization, use Keras' ImageDataGenerator to perform real-time data augmentation with techniques such as random rotations, shifts, flips, and zooms. This generator creates batches of augmented tensor image data on-the-fly during training, thereby increasing dataset variability without additional data collection.

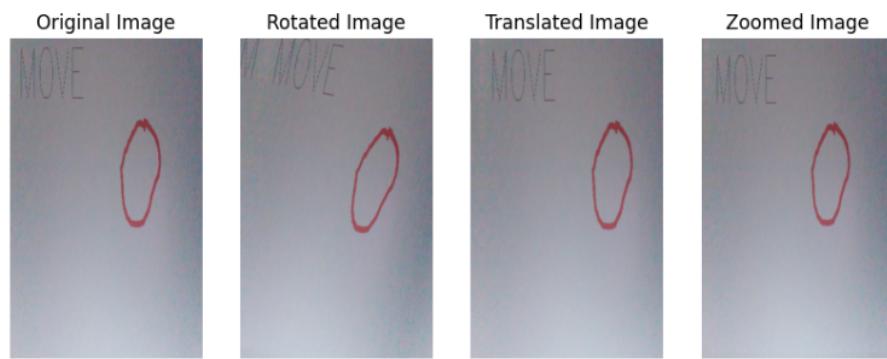


Figure 3.3: Augmented data.

87

Random Rotation: Calculate the rotation matrix:

$$\text{Rotation Matrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

For each pixel in the original image, apply the rotation matrix to calculate the new coordinates:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \text{Rotation Matrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

3

Random Zooming: For each pixel in the original image, scale the coordinates by the random zoom factor:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \alpha \begin{bmatrix} x \\ y \end{bmatrix}$$

151

Random Contrast Adjustment: For each pixel intensity value I in the original image, adjust the intensity using the random contrast adjustment factor β :

$$I' = \beta \cdot I + I$$

37

Split Dataset: Split the dataset into training, validation, and test sets. Typically, the dataset is split into 80% for training, for validation & testing 20%. This helps in evaluating the model's performance and fine-tuning it effectively.

3.4 Pre-trained Models

3.4.1 Xception

142

Xception[18] is a highly accurate and computationally efficient convolutional neural network, ideal for real-time applications. It employs depthwise separable convolutions, reducing parameters and computations. Residual connections in Xception prevent van-

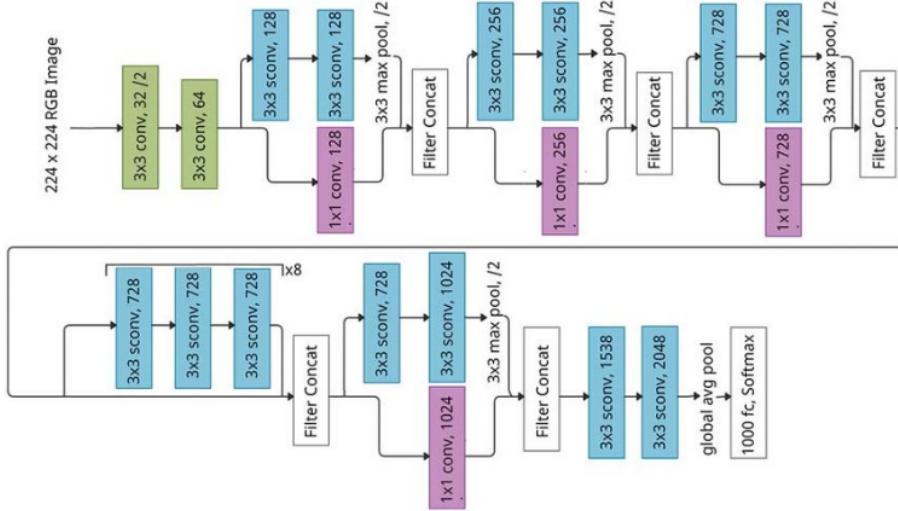


Figure 3.4: The Xception architecture.

ishing gradients, enhancing training stability and effectiveness. The architecture balances high performance with reduced computational cost. Figure 3.4 shows the Xception architecture.

3.4.2 InceptionV3

InceptionV3 [10] is a convolutional neural network architecture renowned for its strong performance in image classification, largely due to its inception modules. These modules incorporate convolutional filters of varying sizes within a single layer, enabling the network to effectively capture features at multiple scales. The architecture of InceptionV3 includes layers of stacked inception modules, each featuring simultaneous 1x1, 3x3, and 5x5 convolutional filters, as well as a pooling layer. This arrangement allows the network to learn a variety of features concurrently, enhancing its ability to quickly extract and interpret complex visual data. Figure 3.5 shows InceptionV3 architecture.

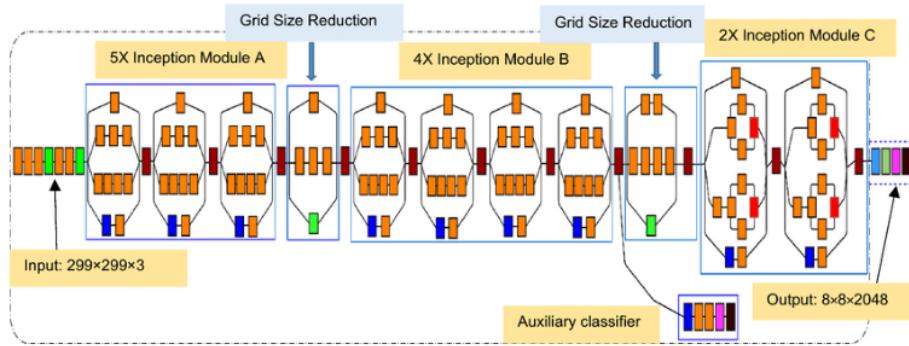


Figure 3.5: The Inception-V3 architecture.

3.4.3 Inceptionresnetv2

InceptionResNetV2 [11] is a deep convolutional neural network that combines the strengths of Inception architectures and residual connections. Introduced by Szegedy et al. in 2016, it leverages the advantages of both Inception modules, which are adept at handling various spatial dimensions, and residual connections that help in training very deep networks. The model features repeated Inception blocks interspersed with residual connections, allowing for efficient training and improved accuracy. It is widely used in image recognition tasks due to its robust performance and relatively lower computational cost compared to other deep architectures. The InceptionResNetV2 architecture is designed to handle complex image classification tasks with a high degree of efficiency. Its use of Inception modules allows the network to capture fine-grained details at multiple scales, making it adept at recognizing intricate patterns in images. The inte-

egration of residual connections helps to mitigate the vanishing gradient problem, which is common in very deep networks, ensuring that the gradient flow is preserved during backpropagation. This combination of techniques enables InceptionResNetV2 to achieve state-of-the-art performance on various benchmark datasets. Figure 3.6 shows Xception architecture.

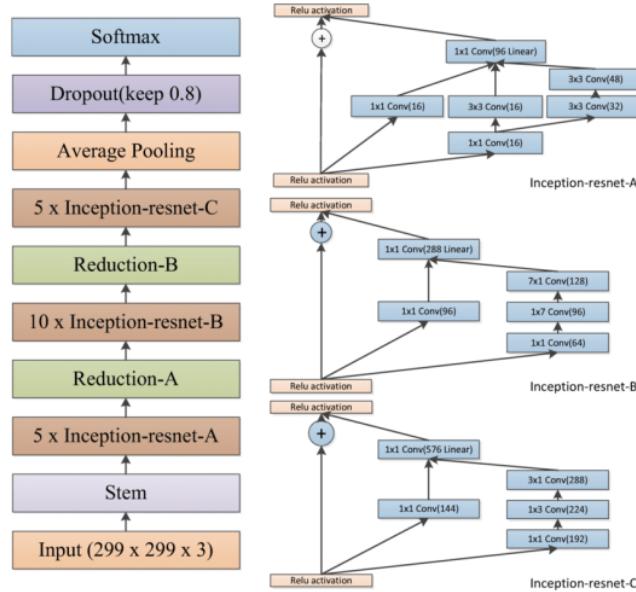


Figure 3.6: The Inceptionresnetv2 architecture.

3.4.4 43 MobileNet

MobileNet [12] is a family of lightweight deep neural network architectures designed by Google for efficient performance on mobile and embedded devices. Introduced in 2017, it employs depthwise separable convolutions, significantly reducing the number of parameters and computational cost compared to traditional convolutional networks. MobileNet models are scalable, allowing adjustments in the width and resolution of the network to balance accuracy and efficiency. They are widely used in various applications, including image classification, object detection, and mobile vision tasks, due to their high performance and low resource requirements. MobileNet's architecture also supports the use of neural network accelerators, further enhancing its suitability for edge computing scenarios. Its flexibility makes it ideal for deploying deep learning models on resource-limited devices. Figure 3.7 shows Xception architecture.

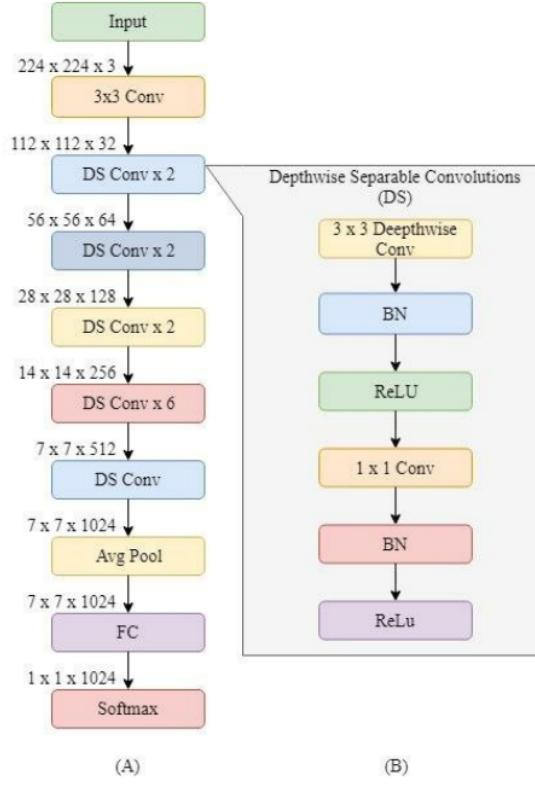


Figure 3.7: The MobileNet architecture.

¹³⁶ 3.4.5 MobileNetV2

MobileNetV2 [13] is a convolutional neural network architecture optimized for mobile and embedded vision applications. It introduces inverted residuals with linear bottlenecks, where intermediate expansion layers have more channels than the input and

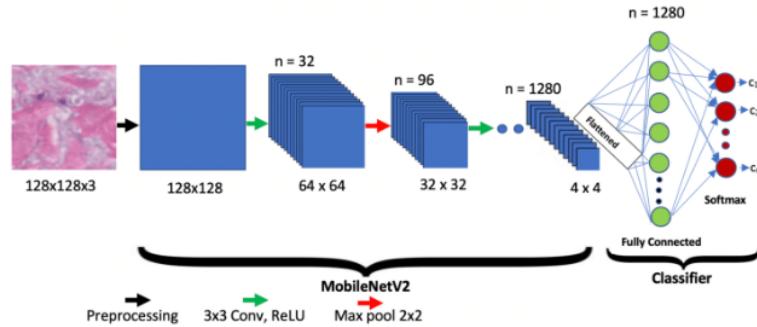


Figure 3.8: The MobileNetV2 architecture.

tlenecks, where intermediate expansion layers have more channels than the input and

output layers, preserving information while reducing dimensions efficiently. Figure 3.8 shows Xception architecture. The architecture uses depthwise separable convolutions to minimize parameters and computations, splitting the convolution operation into independent filtering and combining steps. Additionally, MobileNetV2 eliminates non-linearities in narrow layers to retain important features. These innovations make MobileNetV2 highly efficient, achieving a balance between model size, speed, and accuracy, ideal for mobile and edge device applications.

3.4.6 EfficientNetB0

138
EfficientNetB0 [9] is a convolutional neural network architecture designed for high performance and efficiency in image classification tasks. Figure 3.9 shows Xception architecture. It is the baseline model in the EfficientNet family, which scales up the

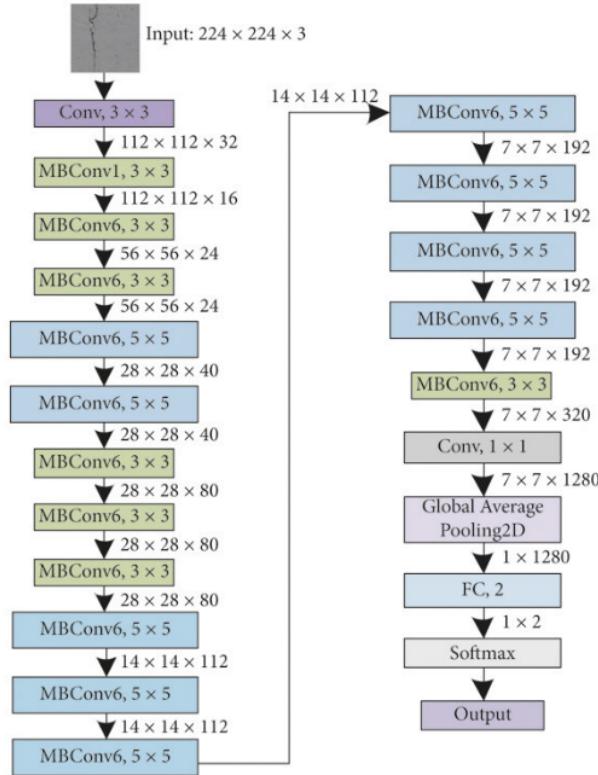


Figure 3.9: The EfficientNetB0 architecture.

network's width, depth, and resolution in a balanced manner using a compound scaling

method. This approach enables EfficientNetB0 to achieve superior accuracy while using fewer parameters and computational resources compared to previous architectures.

¹⁰⁷ EfficientNetB0 utilizes mobile inverted bottleneck convolution (MBConv) layers with squeeze-and-excitation optimization, enhancing feature extraction and reducing model size. This makes EfficientNetB0 ideal for deployment in resource-constrained environments, providing a good balance between accuracy and efficiency.

3.4.7 Resnet50

Figure 3.10 shows Xception architecture. ResNet50's [14] design, featuring residual

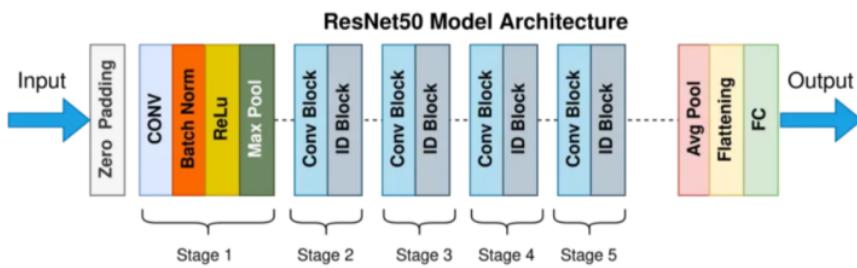


Figure 3.10: The Resnet50 architecture.

connections, addresses the vanishing gradient problem in deep networks, enabling efficient training of its 50 layers. This balance of depth and efficiency makes it a preferred choice for diverse computer vision tasks, from segmentation to image classification.⁴³ Introduced by He et al. in 2015, it utilizes residual connections to allow the training of very deep networks by addressing the vanishing gradient problem. These skip connections help in learning residual functions with reference to the layer inputs, making it easier to optimize deeper networks.

3.4.8 Resnet101

¹⁴⁷ ResNet101 [15] is a deep convolutional neural network that extends the ResNet (Residual Network) architecture to 101 layers. Figure 3.11 shows Resnet101 architecture. ResNet101 is renowned for its strong performance in image classification, object detection, and segmentation tasks, offering higher accuracy compared to its shallower counterparts like ResNet50, while maintaining a manageable computational load.

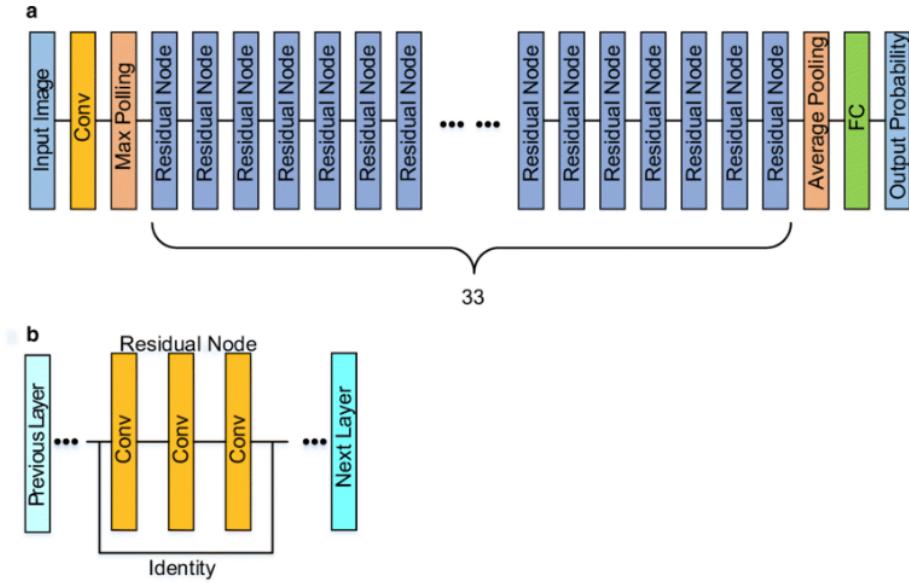


Figure 3.11: The Resnet101 architecture.

3.4.9 Vgg16

The University of Oxford’s Visual Geometry Group (VGG) unveiled the convolutional neural network architecture known as VGG16 [16] in 2014. Figure 3.12 shows Vgg16 architecture. There are three fully connected layers and thirteen convolutional layers

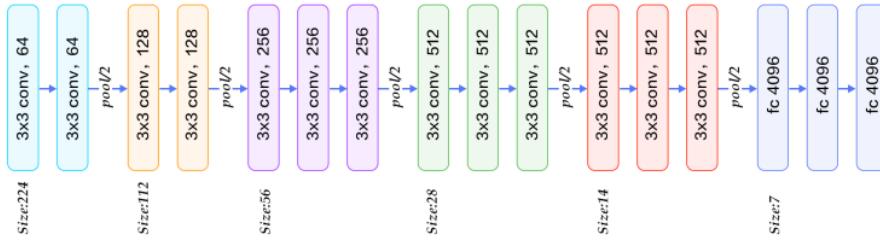


Figure 3.12: The Vgg16 architecture.

among its sixteen weight layers. Utilizing tiny 3x3 convolution filters that are stacked to increase the network's depth, the design places an emphasis on simplicity. The VGG16 architecture is renowned for consistent and for performing exceptionally well in image classification tasks. Because of its great accuracy and simplicity of use, it is still widely used even though it requires a large amount of memory and is computationally demanding. It is frequently utilized as a standard in many different computer applications.

3.4.10 Vgg19

VGG19 [17] is a deep convolutional neural network architecture developed by the Visual Geometry Group (VGG) at the University of Oxford, introduced in 2014. It is an extension of the VGG16 model, featuring 19 weight layers—16 convolutional layers and 3 fully connected layers. The network uses small 3x3 convolution filters throughout its architecture, stacked to create deep networks capable of capturing intricate features. VGG19 is known for its simplicity and effectiveness in image classification tasks, achieving high accuracy. Despite its high computational cost and memory requirements, it remains widely used for its performance and has become a standard benchmark in computer vision research. Figure 3.13 shows Xception architecture.

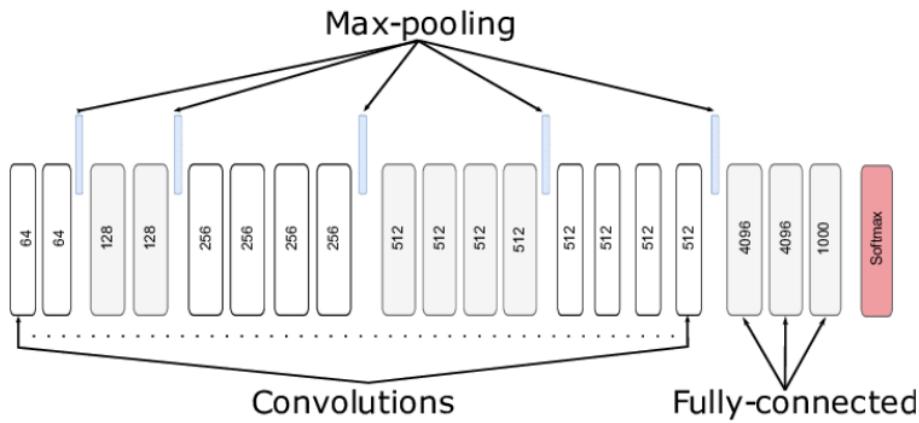


Figure 3.13: The VGG19 architecture.

3.5 Flow Diagram of Methodology

The methodology outlined in the flowchart is designed to classify handwritten digits through air writing, which is particularly relevant to your thesis topic on Bangla digit image classification. The process starts with a video stream that captures the hand movements of users. This video stream serves as the primary input for the system. The first major step in the process is hand pose detection. The system continuously monitors the video stream for the presence of a hand pose. If no hand pose is detected, the system keeps checking the video stream. However, once a hand pose is detected, the system moves to the next step, which is fingertip detection. During fingertip detection,

the system assesses whether the detected fingertip velocity is below threshold (t). This is crucial because it helps in distinguishing intentional writing gestures from random hand movements. If the fingertip velocity is above the threshold, indicating rapid or non-writing movements, the system may disregard this data. But if the velocity is below the threshold, the system assumes the hand is in a writing motion and proceeds to record the digit trajectory. The digit trajectory is then outputted, forming a raw representation of the written digit. To ensure the system is trained on a diverse dataset, data is collected from 200 different individuals. This dataset helps in making the model robust and generalizable to various writing styles and conditions.

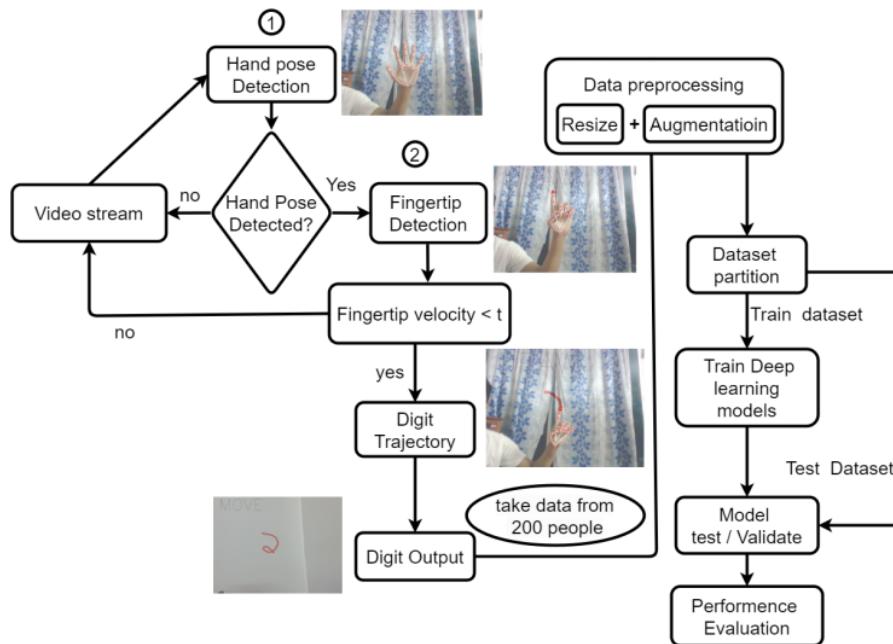


Figure 3.14: The flow diagram of methodology for this research.

Before training the models, the collected data undergoes preprocessing. This involves resizing the digit images to a standard dimension and applying augmentation techniques. Data augmentation is critical as it artificially increases the diversity of the training data by creating modified versions of the original dataset, thereby improving the model's ability to generalize to new, unseen data. The preprocessed data is then partitioned into training and test datasets. The training dataset is used to train multiple deep learning models. The following deep learning models have been executed for

projects: EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18]. Each of these models has its architecture and strengths, making it valuable to train and compare them to determine the most effective one for the task. Once the models are trained, they are tested and validated using the test dataset. This step is crucial for assessing the models' performance in recognizing and classifying the handwritten digits accurately. Performance evaluation metrics are then used to compare the models, ensuring that the best-performing model is selected for the final deployment. This methodology is comprehensive and well-suited for your thesis on Bangla digit recognition on air writing. It encompasses all necessary steps, from capturing hand movements to preprocessing data and training advanced deep learning models, ultimately leading to accurate and reliable digit classification.

3.6 Summary

This chapter of the thesis presents a comprehensive methodology for developing an efficient Bangla digit image classification system using air writing. It outlines the data collection process, gathering diverse samples from various sources, including volunteers performing air writing and synthetic data generation, to ensure a representative dataset. The preprocessing steps involve normalizing the images and converting them into a suitable format for machine learning models. The chapter compares several pre-trained models and introduces ensemble techniques to enhance classification performance. This structured approach aims to optimize the accuracy and effectiveness of Bangla digit recognition through air writing, providing a robust foundation for further research and application.

9 CHAPTER 4

Implementation

4.1 Overview

This chapter outlines the practical steps involved in developing and training models for Bangla digit classification on air writing. It begins with comprehensive data preparation, including dataset collection from various sources, resizing, orientation correction, ¹⁴⁰ and augmentation techniques to enhance dataset diversity and model robustness. The chapter then details the selection and configuration of pretrained CNN architectures such as EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18]. Each model is adapted for Bangla digit recognition through additional classification layers and fine-tuned using transfer learning from ImageNet.

4.2 Tools & Technologies

The implementation and training of the Bangla digit classification on air writing models utilized a blend of hardware and software to ensure efficient processing and model training. High-performance GPUs were employed to expedite the training of deep learning ¹¹⁸ models. Frameworks and libraries such as TensorFlow [23] and Keras [21] were used for implementing various deep learning models such as EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18]. Python [19] served as the primary language for scripting and data preprocessing, with OpenCV [20] aiding in various image processing tasks. This integration of advanced hardware and software tools facilitated the efficient creation and training of the Bangla digit classification models.

4.2.1 Hardware

1. **ASUS TUF F15:** This laptop, equipped with an Intel Core i5 11th Gen processor, 16 GB of RAM and a 512 GB SSD, provided the necessary computational power and storage for training deep learning models efficiently.
2. **HP Pavilion 15:** Featuring an Intel Core i5 10th Gen processor, 8 GB of RAM and a 512 GB SSD, this laptop was used for additional processing tasks. The built-in cameras of both laptops were employed for capturing air writing samples, ensuring a diverse and representative dataset.
3. **Google Colab:** Google Colab [24] was employed for building and training the machine learning models. It provided access to a T4 GPU, significantly accelerating model training compared to CPU-only computations. The Google Colab instances are equipped with an Intel Xeon CPU @ 2.20GHz with 1 core and 2 logical processors due to hyper-threading, an NVIDIA Tesla T4 GPU with 2,560 CUDA cores, and 16 GB GDDR6 memory. 320 Tensor Cores, 8.1 TFLOPS single-precision performance, and 320 GB/s memory bandwidth are all offered by the T4. In addition to providing an extra 100 GB of ephemeral storage and 12–16 GB of RAM, Colab also integrates Google Drive for persistent storage.

4.2.2 Software

1. **Python:** Python 3.10.12 [25] was chosen for its versatility and extensive ecosystem of libraries and frameworks in machine learning and data science. Python's simplicity and readability make it an ideal choice for rapid prototyping and development.
2. **Visual Studio Code:** Visual Studio Code was selected as the Integrated Development Environment (IDE) for coding. It provided a streamlined interface for writing, debugging, and executing Python [25] scripts used in data preprocessing and model development.
3. **Jupyter Notebook:** Jupyter Notebook [26] was utilized for conducting data preprocessing tasks. Its interactive environment facilitated exploratory data analysis,

data manipulation, and visualization, which were crucial for preparing the air writing dataset before training.

4.2.3 Packages and Libraries

1. **CV2:** CV2 4.5.4 [20] short for OpenCV (Open Source Computer Vision Library),
is a widely-used library in computer vision and image processing. It provides
a vast array of functions for tasks such as reading, writing, manipulating, and
analyzing images and videos. OpenCV [20] is renowned for its efficiency and
versatility in applications ranging from object detection and recognition to facial
recognition and augmented reality
2. **numpy:** A core Python [25] library for numerical computation is called NumPy
[27]. Large, multi-dimensional arrays and matrices are supported, and a number
of mathematical operations are available for effective manipulation of these ar-
rays. Because NumPy [27] is the foundation for so many other libraries in the
Python [19] ecosystem, it is necessary for tasks like scientific computing, ma-
chine learning, and data analysis. Numpy 1.25.2 have been used for research.
3. **mediapipe:** Google created the Mediapipe 0.10.14 [28] framework, which pro-
vides a full range of pre-built tools and machine learning models for creating dif-
ferent kinds of perception-based applications. It makes it easier to create pipelines
that process and analyze multimedia data, including audio and video streams.
With modules for tasks like pose estimation, object detection, face detection,
and hand tracking, Mediapipe [28] is a valuable tool for a variety of computer vision
and machine learning applications.
4. **Pillow:** PIL 9.4.0, known as Pillow [29] is a Python library for opening, ma-
nipulating, and saving various image file formats. It's used widely in digital art,
computer vision, and image processing for tasks like filtering, resizing, and for-
mat conversion. In addition to these basic operations, Pillow supports advanced
functionalities such as image enhancement, drawing shapes and text, and batch
processing of images. Its ease of use and extensive documentation make it a pop-
ular choice among developers and researchers for implementing complex image
processing pipelines efficiently.

4.3 Dataset Collection

The dataset for Bangla digit on air writing image classification was collected using a custom script that utilizes the MediaPipe [28] library for hand tracking and OpenCV [20] for image processing. The steps involved in the dataset collection are detailed below:

1. **Environment Setup:** The script begins by importing necessary libraries: OpenCV [20] for image processing, NumPy [27] for numerical operations, MediaPipe [28] for hand tracking, and additional utilities for handling files and directories.

```
74
1 import cv2
2 import numpy as np
3 import mediapipe as mp
4 import os
```

2. **Hand Tracking Initialization:** MediaPipe's [28] hand tracking solution is initialized to detect and track hand landmarks. This enables capturing the movement and positions of hand joints in real-time.

```
4
1 mp_hands = mp.solutions.hands
2 hands = mp_hands.Hands()
```

3. **Video Capture:** The script captures video from the webcam. Each frame of the video is processed to detect hand landmarks.

```
1 cap = cv2.VideoCapture(0)
```

4. **Identification of Hand Landmarks:** The core logic involves processing each video frame to detect hand landmarks. If a hand is detected, the landmarks are extracted and normalized to the image dimensions.

```
91
1 while cap.isOpened():
2     success, image = cap.read()
3     if not success:
```

```

4     continue
5
6     49
7     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
8     results = hands.process(image)
9
10    if results.multi_hand_landmarks:
11        for hand_landmarks in results.multi_hand_landmarks:
12            data = [(landmark.x * width, landmark.y *
13                     height) for landmark in hand_landmarks.
14                     landmark]
15            data = np.array(data)
16            # Further processing and mode detection

```

These landmarks are then used to perform specific actions like drawing or moving based on the distance between certain points.

5. **Drawing and Mode Handling:** The script defines various modes such as drawing, erasing, and clearing. These modes are determined by analyzing the hand landmarks and distances between specific points.

```

1 def draw(src, pt1, pt2):
2     if pt1 != (-5000, -5000) and pt2[0] != (-5000, -5000):
3         src = cv2.line(src, pt1, pt2, (0, 0, 255), 10)
4
5 def clear(src):
6     src = cv2.rectangle(src, (0, 0), (width, height), (255,
7                         255, 255), -1)
8
9 def action(digitLength, pt1, pt2, src):
10    if digitLength < some_threshold:
11        draw(src, pt1, pt2)
12    return mode

```

6. **Saving Collected Images:** Collected data is saved when the user presses specific keys. The data includes hand landmarks and the corresponding drawn images. This data is stored in directories named after the collected characters.

```

1 if k == ord('s'):

```

```

2     n_collected += 1
3     save_dir = f"./collected_data/{user_name}/{  

4         collected_char}/"
101    if not os.path.isdir(save_dir):
5         os.mkdir(save_dir)
6     write_data(save_dir, data_history, blended)
7     clear(src)
8     data_history.clear()

```

7. **Cleanup:** Windows closed for cleanup, data collected, and resources released.

```

1 hands.close()
2 cap.release()
3 cv2.destroyAllWindows()

```

This structured approach allows for systematic collection and organization of the dataset, ensuring that each character's hand movement data is accurately captured and stored for subsequent training and analysis.

4.4 Dataset Preparation

This section outlines the process of preparing the dataset for the Bangla digit classification on air writing project. The dataset consists of a total of 5314 images distributed across 10 classes. Below is the distribution of images per class as shown in Table 4.3

Table 4.3: No. of images for collected Bangla digit.

Digit	No. of Bangla digit
0	526
1	534
2	533
3	531
4	532
5	532
6	532
7	533
8	531
9	530

When collecting the dataset, each individual's name was assigned to organize the data. Initially, all data were stored in folders labeled from 0 to 9, corresponding to each digit. Subsequently, data from approximately 200 individuals were consolidated into the 'collected_data' folder, where subfolders 0 through 9 contain data classified by their respective digits. This organizational structure defines the classes used in this study.

The Bangla digit on air writing image dataset utilized in our study includes the fol-

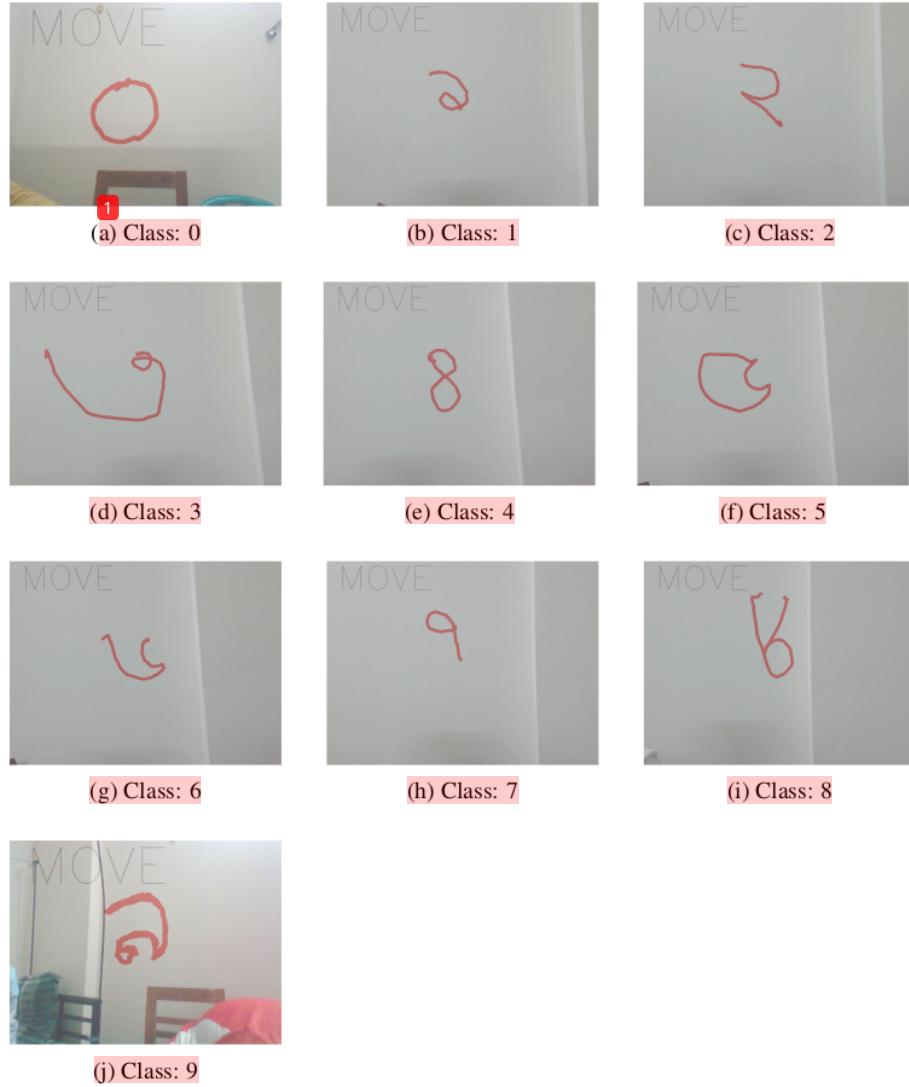


Figure 4.15: Sample images from our dataset.

lowing pictures shown in Figure 4.15. The preparation involves defining the classes,

51

preprocessing the images, augmenting the data, and dividing the dataset into training, validation, and test sets.

4.4.1 Data Preprocessing

To ensure uniformity and enhance model performance, we implemented several pre-processing procedures on the original images. These included resizing, orientation correction, systematic renaming, and organizing into a well-structured directory. These preprocessing steps are essential for maintaining consistency throughout the dataset, thereby facilitating effective model training and evaluation.

1. **Directory Setup:** This Python [19] script creates a single "destination_folder" by combining data from multiple folders arranged by class labels (0–9) and individuals. It creates "destination_folder" if it doesn't already exist after making sure it does. The script traverses through subfolders that match class labels as it iterates through each person folder in "source_folder". It ensures that every file is uniquely renamed using the individual's folder name, class label, and unique index when it copies files from each class folder to "destination_folder". The combined dataset can be easily accessed and analyzed thanks to this organized method.

```
29
1 import os
2 import shutil 33
3 def merge_data(source_folder, destination_folder):
4     if not os.path.exists(destination_folder):
5         os.makedirs(destination_folder)
6
7     for person_folder in os.listdir(source_folder):
8         person_folder_path = os.path.join(source_folder,
9             person_folder)
10        if os.path.isdir(person_folder_path):
11            for class_folder in range(10):
12                class_source_folder = os.path.join(
13                    person_folder_path, str(class_folder))
14                if os.path.exists(class_source_folder):
```

```
13
14     for root, dirs, files in os.walk(
15         class_source_folder):
16         for index, file in enumerate(files):
17             source_file = os.path.join(root,
18                 file)
19             destination_file = os.path.join(
20                 destination_folder, str(
21                     class_folder), f"{
22                     person_folder}_{class_folder}
23                     }_{index}{os.path.splitext(
24                         file)[1]}")
25             if not os.path.exists(os.path.
26                 dirname(destination_file)):
27                 os.makedirs(os.path.dirname(
28                     destination_file))
29             shutil.copy(source_file,
30                         destination_file)
31
32 # Source folder containing data for each person
33 source_folder = 'collected_data'
34
35 # Destination folder where merged data will be stored
36 destination_f
37
38 merge_data(source_folder, destination_folder)
```

2. **Renaming:** The provided Python [19] script utilizes the os module to systematically rename files within sub-folders labeled from 0 to 9 within the directory "collected_data". It iterates through each sub-folder, checks for the existence of files, and sequentially renames them using a format that combines the sub-folder name and a running index. This ensures that each file is uniquely identified and organized, facilitating better management and analysis of the dataset. The script includes error handling to manage cases where sub-folders or files might be missing, maintaining robustness throughout the renaming process.

```
1 import os  
2 def change_filenames(directory):  
3     # Iterate through each sub-folder (0-9)
```

```

4   for subdir in range(10):
5       27
6           subdir_path = os.path.join(directory, str(subdir))
7           # Check if the sub-folder exists
8           if os.path.exists(subdir_path):
9               29
10              # Iterate through each file in the sub-folder
11              for idx, filename in enumerate(os.listdir(
12                  subdir_path)):
13                  # Construct the new filename
14                  new_filename = f"{subdir}_{idx+1}.jpg" #
15                  Assuming JPG images, change extension
16                  accordingly if needed
17                  # Rename the file
18                  46
19                  os.rename(os.path.join(subdir_path,
20                      filename), os.path.join(subdir_path,
21                      new_filename))
22
23                  print(f"Renamed {filename} to {new_filename}
24                      )")
25
26      15 # Directory where collected_data folder is located
27      directory = "small_data_1"
28
29      change_filenames(directory)

```

3. **Resizing :** The Python [19] script resizes images in an input folder before saving them to an output folder. It uses the os module for file operations and PIL [29] for image processing. The resize_images function requires three arguments: "input_folder", "output_folder", and size (224x224 pixels by default). If the output folder does not already exist, it creates one, and then searches the input folder for picture files. It resizes each image and saves it to the appropriate subfolder in the output folder while preserving the original folder structure. Each scaled image generates a message, which the script prints.

```

102
1   import os
2   from PIL import Image
3
4   def resize_images(input_folder, output_folder, size=(224,
5       224)):
93
6       if not os.path.exists(output_folder):

```

```

6         os.makedirs(output_folder)

7
8  59   for subdir, dirs, files in os.walk(input_folder):
9      for file in files:
10         file_path = os.path.join(subdir, file)
11         82   if file_path.lower().endswith(('png', 'jpg', 'jpeg', 'bmp', 'gif')):
12             with Image.open(file_path) as img:
13                 img_resized = img.resize(size, Image.
14                                         ANTIALIAS)
15
16             # Create the corresponding output
17             50   subdirectory
18             relative_path = os.path.relpath(subdir,
19                                              input_folder)
20             output_subdir = os.path.join(
21                 output_folder, relative_path)
22             if not os.path.exists(output_subdir):
23                 os.makedirs(output_subdir)
24             41   output_path = os.path.join(
25                 output_subdir, file)
26             img_resized.save(output_path)
27             print(f"Resized and saved: {output_path}
28                  ")
29
30     input_folder = '/content/collected_data'  # Change this to
31     your input folder path
32     output_folder = '/content/resize_collected_data'  # Change
33     this to your output folder path
34     resize_images(input_folder, output_folder)

```

4.4.2 Data Augmentation

67 We used a variety of **data augmentation** techniques to improve the model's capacity to generalize and enrich the dataset. By making altered copies of the original photos, these methods increase the diversity of the dataset. By using this method, the model is able to learn to recognize signs in a greater variety of environmental and situational contexts. The techniques used for augmentation include

```

28   train_datagen = ImageDataGenerator(

```

```

2     rescale=1./255,
3     rotation_range=60,
4     width_shift_range=0.5,
5     height_shift_range=0.5,
6     shear_range=0.5,
7     zoom_range=0.5,
8     horizontal_flip=True,
9 )
10 ])

```

66

1. **Rescale:** Increases the effectiveness of training a model by normalizing pixel values to a range of 0 to 1.
2. **Rotation Range:** Increases image quality and diversifies training samples by randomly rotating them up to 60 degrees.
3. **Width Shift Range:** Broadens dataset variability by horizontally shifting images by up to half their width.
4. **Height Shift Range:** Adjusts images vertically by up to 50%, increasing the diversity of the dataset.
5. **Shear Range:** Increases the robustness of the dataset by introducing shearing transformations up to 50%.
6. **Zoom Range:** Enhances dataset robustness by introducing shearing transformations up to 50%
7. **Zoom Range:** Increases the robustness of the dataset by introducing shearing transformations up to 50%.
8. **Horizontal Flip:** Horizontal image mirroring adds diversity to dataset samples for better generalization.

4.4.3 Data Splitting

6

The provided code segment splits the dataset into training and validation sets using TensorFlow's [23] "image_dataset_from_directory" function. It loads images from the

directory specified by "folder_dir_tr", resizes them to a specified size (SIZE), likely (500x300 pixels), and organizes them into batches for efficient training.

```

 1  train_set = tf.keras.utils.image_dataset_from_directory(
 2      directory=folder_dir_tr,
 3      shuffle=True,
 4      image_size=SIZE,    # Resize images to 244x244
 5      batch_size=batch_size,
 6      label_mode='categorical',
 7      seed=22,
 8      validation_split=0.2,   # 20% of the data will be used for
 9          validation
10      subset='training',   # Specify that this is the training set
11      interpolation='bilinear'  # Interpolation method for
12          resizing
13
14
15 validation_set = tf.keras.utils.image_dataset_from_directory(
16      directory=folder_dir_tr,
17      shuffle=True,
18      image_size=SIZE,    # Resize images to 244x244
19      batch_size=batch_size,
20      label_mode='categorical',
21      seed=22,
22      validation_split=0.2,   # 20% of the data will be used for
23          validation
24      subset='validation',   # Specify that this is the validation
25          set
26      interpolation='bilinear'  # Interpolation method for
27          resizing
28 )

```

1. **Training set:** It uses 80% of the data for training. Images are shuffled "shuffle=True" to randomize the order in which batches are presented during training. Labels are encoded in the categorical format "label_mode='categorical'" .
2. **Validation set:** It uses the remaining 20% of the data "validation_split=0.2", "subset='validation'" for validation. Similar settings for image resizing, batch-

ing, shuffling, and label encoding apply as in the training set.

- 37
3. **Seed:** The seed=22 parameter is used to set a **seed** value for the random number generator in TensorFlow's [23] "image_dataset_from_directory" function. This ensures reproducibility in data shuffling and batch organization. By setting a specific seed (22 in this case), the random processes involved in shuffling images and organizing batches will produce the same sequence of outputs each time the code is run, provided all other parameters remain unchanged. This is particularly useful for debugging, testing, and ensuring consistent results in machine learning experiments.

4.5 Model Training

We selected each model (EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18]) for its complex architecture, proven effectiveness in image classification, and adaptability to Bangla Sign Language recognition. We delve into specific architectural details, data augmentation strategies, training configurations, and optimization techniques employed to fine-tune these models for accurate recognition of Bangla Sign Language.

4.5.1 Experimental Setup

1. **Learning Rate:** Learning rates for the models were set to 0.00001 for EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Xception [18] while Vgg19 [17] used a slightly lower rate of 0.000001. These choices were made to strike a balance between training stability and convergence speed, considering the complexities of each model and the characteristics of the dataset.

```

1   18
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
24   34     loss='categorical_crossentropy', metrics=['accuracy'
      ''])

```

```
1     model.compile(optimizer=tf.keras.optimizers.Adam(1e-6),
2                     loss='categorical_crossentropy', metrics=['accuracy'])
3 )
```

2. **Epoch:** The training of multiple models over 50 epochs is demonstrated by this code snippet, which includes EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18] models. In order to maximize model performance and avoid overfitting, callbacks like checkpointing, early stopping, and learning rate reduction are used to track training progress.

```
26
1   history = model.fit(
2     train_generator,
3     epochs=50,
4     validation_data=validation_generator,
5     callbacks=[checkpoint, early_stopping, reduce_lr]
6   )
```

3. **Image Size:** This configuration indicates that 224 x 224 pixel images were resized for the EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18] models.

```
1 data_dir = '/kaggle/input/saadnewdataset/dataset'
2 img_size = (224, 224)
3 batch_size = 32
```

4. **Weight:** By initializing the model with pre-trained parameters from the ImageNet dataset [30], the 'imagenet' weights improve the model's recognition and feature extraction capabilities from images. This method makes use of in-depth training on a wide variety of objects and scenes, which enables efficient transfer learning for enhanced model performance across a range of computer vision tasks.

```

1      39
base_model = EfficientNetB0(weights='imagenet',
                             include_top=False, input_shape=(224, 224, 3))

```

5. **Loss function:** The loss function being used is categorical_crossentropy. This loss function is typically used for multi-class classification problems where the labels are one-hot encoded.

```

1      18
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
               loss='categorical_crossentropy', metrics=['accuracy'])

```

6. **Batch Size:** A batch size of 32 was used for all models to ensure stable gradient updates and effective GPU utilization. This balance helps in maintaining computational efficiency and training stability across the models.

```

1      data_dir = '/kaggle/input/saadnewdataset/dataset'
2      img_size = (224, 224)
3      batch_size = 32

```

7. **Optimizer:** The Adam optimizer is a popular choice in deep learning, combining the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp.

```

1      18
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
               34
               loss='categorical_crossentropy', metrics=['accuracy'])

```

```

1      model.compile(optimizer=tf.keras.optimizers.Adam(1e-6),
                  loss='categorical_crossentropy', metrics=['accuracy'])

```

It adapts the learning rate for each parameter and uses both first-order and second-order moments of gradients to accelerate convergence. Using a learning rate of 1×10^{-5} or 1×10^{-6} , it ensures fine-tuned adjustments during training, optimizing the categorical_crossentropy loss for multi-class classification tasks.

8. Regularization:

- (a) **Early stopping:** The provided code's EarlyStopping callback keeps track of the validation loss (val_loss). To prevent the model from overfitting, it stops the training process if the validation loss does not get better for three consecutive epochs (patience=3).

```
1     early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

It maintains the model's optimal state by restoring the weights from the epoch with the best validation performance (restore_best_weights=True), effectively striking a balance between training efficiency and generalization capacity.

- (b) **Dropout:** Increasing the dropout rate to 0.5 means that during training, half of the input units to the dropout layer (x) will be randomly set to zero.

```
1     x = Dropout(0.5)(x)
```

This regularization technique helps prevent overfitting by encouraging the network to learn more robust and generalized features.

4.5.2 Load of Pre-trained Models

1. **Xception Model:** Google developed Xception [18]), which stands for "Extreme Inception," a sophisticated convolutional neural network design. By replacing the traditional Inception modules with depthwise separable convolutions, it improves the Inception design and creates a more effective model that performs well in picture classification tasks.

```
13    base_model = Xception(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

2. **InceptionV3 Model:** Google unveiled InceptionV3 [10], a complex convolutional neural network architecture with a unique modular structure that allows it

to handle multi-scale information. It achieves the highest performance in a variety of vision tasks and is very effective for large-scale image classification.

```
13
base_model = InceptionV3(weights='imagenet', include_top=
    False, input_shape=(224, 224, 3))
```

3. **MobileNetV2 Model:** Google created the effective deep learning model MobileNetV2 [13], which makes creative use of linear bottlenecks and

```
13
base_model = MobileNetV2(weights='imagenet', include_top=
    False, input_shape=(224, 224, 3))
```

42 inverted residuals for mobile and embedded vision applications. Because of its ability to balance precision and speed, it works well in contexts with limited resources.

4. **Inceptionresnetv2:** With the help of InceptionResNetV2 [11], a potent convolutional neural network architecture, deep learning can be accomplished effectively and efficiently by combining the advantages of residual connections and Inception modules. In comparison to other architectures, this hybrid approach enables the model to achieve high accuracy with comparatively fewer parameters.

```
13
base_model = Inceptionresnetv2(weights='imagenet',
    include_top=False, input_shape=(224, 224, 3))
```

Its robust performance and versatility make it a popular choice for image classification and other computer vision applications.

5. **Resnet50:** ResNet50 [14] is a 50-layer deep convolutional neural network that is well-known for addressing the vanishing gradient issue by using residual connections. The model can learn identity mappings thanks to these connections, which facilitates the training of extremely deep networks. ResNet50 offers a balance between computational efficiency and depth, making it a highly effective model for image classification and other computer vision tasks.

```
17
base_model = Resnet50(weights='imagenet', include_top=False
    , input_shape=(224, 224, 3))
```

- 6. Resnet101:** With 101 layers, ResNet101 [15] is a deep convolutional neural network that builds upon ResNet50's architecture to enhance performance. It solves the vanishing gradient problem by using residual connections to make training easier by allowing gradients to move through the network more readily. Because of its greater depth, ResNet101 provides higher accuracy in image classification
149 and other computer vision tasks. It is a widely used model.

```
30
: base_model = Resnet101(weights='imagenet', include_top=False,
    input_shape=(224, 224, 3))
```

- 7. Vgg16:** The 16-layer convolutional neural network architecture VGG16 [16] is renowned for its consistency and ease of use. It is composed of fully connected layers after a sequence of convolutional layers with tiny 3x3 filters. VGG16 is a well-liked option in the computer vision community because of its high performance in image classification tasks, which is achieved despite its relatively simple design.

```
77
: base_model = Vgg16(weights='imagenet', include_top=False,
    input_shape=(224, 224, 3))
```

- 8. Vgg19:** Adding more convolutional layers for improved feature extraction, the VGG19 [17] convolutional neural network architecture has 19 layers and builds upon the VGG16 design. At the end, it has fully connected layers and consistently applies small 3x3 filters to all convolutional layers. Thanks to its deeper architecture, which enables it to detect more intricate patterns in the data, VGG19
30 is renowned for its excellent performance in image classification tasks.

```
39
: base_model = Vgg19(weights='imagenet', include_top=False,
    input_shape=(224, 224, 3))
```

4.5.3 Model Configuration

To change the complexity of the model, dense layers with 32, 64, or 128 units are used. These dense layers are preceded by a GlobalAveragePooling2D layer, which lowers the
81 input's spatial dimensions. A dropout layer with a rate of 0.5 is used to randomly set

50% of the units to zero during training in order to prevent overfitting. Lastly, the output probabilities for each class are obtained by classifying the data into ten groups using a softmax layer.

```

54
1 # Add custom layers on top of the base model
2 x = base_model.output
3 x = GlobalAveragePooling2D()(x)
4 x = Dense(64, activation='relu')(x) # Reduce the number of
5   units
45
6 x = Dropout(0.5)(x) # Increase dropout
7 predictions = Dense(10, activation='softmax')(x) # Assuming you
8   have 10 classes
90
9 # Define the model
10
11 model = Model(inputs=base_model.input, outputs=predictions)

```

4.5.4 Training Pre-trained Models

Every pretrained model was initially set to run for 50 epochs, but some of them stopped running early because of early stopping. MobileNet, EfficientNetB0, ResNet50, and ResNet101 all finished all 50 epochs. On the other hand, Xception stopped at 34 epochs, VGG16 at 23 epochs, VGG19 at 39 epochs, MobileNetV2 at 45 epochs, and InceptionV3 at 32 epochs.

```

26
1 history = model.fit(
2   train_generator,
3   epochs=50,
4   validation_data=validation_generator,
5   callbacks=[checkpoint, early_stopping, reduce_lr]
6 )

```

4.6 Summary

This chapter illustrates a methodical approach to implementing Bangla digit classification on air writing models, focusing on practical issues and technology tools. By combining Python [19] for scripting, Visual Studio Code for development, and Google Colab for GPU-accelerated training, the chapter demonstrates a strong architecture for

fast model building and optimization. The use of several pretrained models and ensemble techniques highlights the chapter's emphasis on achieving high accuracy and reliability in recognizing Bangla digit images. This implementation builds the groundwork for the upcoming evaluation and validation in Chapter 5, providing vital insights into the topic of machine learning-driven digit recognition.

CHAPTER 5

Result and Analysis

5.1 Overview

A thorough examination of the outcomes produced from multiple deep learning models applied to the Bangla digit image classification job. This offers extensive performance information for each model: EfficientNetB0 [9], InceptionV3 [10], Inceptionresnetv2 [11], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [15], Vgg16 [16], Vgg19 [17], Xception [18]. The study also assesses the effectiveness of various models in improving categorization accuracy. By rigorously comparing different models, the study hopes to uncover the benefits and shortcomings of each strategy, providing insights about the most effective Bangla digit classification strategies.

5.2 Model Performance

Several models that performed exceptionally well in the evaluation of Bangla digit image classification were identified. In evaluating Bangla digit image classification, VGG16 and InceptionResNetV2 stood out with exceptional 99.0% accuracy, showcasing robust feature extraction. InceptionV3 followed closely at 98.53%, leveraging residual network features effectively. Conversely, ResNet50 and VGG19 showed lower accuracy at 96.77% and 97.55%, respectively. EfficientNetB0 and MobileNetV2 achieved good results at 98.02% and 98.45%, demonstrating efficiency without compromising performance. Xception also performed well with 98.45% accuracy, highlighting its suitability for complex image tasks. Overall, VGG16 and InceptionResNetV2 lead in accuracy, while EfficientNetB0, MobileNetV2, and Xception offer efficient alternatives for robust Bangla digit classification. Overall, the study emphasizes how impor-

tant model architecture is to obtaining high accuracy in Bangla digit recognition, with VGG16, InceptionResNetV2, and InceptionV3 standing out as the top candidates because of their exceptional ability to capture the fine details that are specific to Bangla digits shown in Table 5.4.

98

Table 5.4: Performance metrics of various Deep learning models.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
EfficientNetB0	98.02	98.0	98.0	98.0
InceptionV3	98.53	99.0	99.0	99.0
InceptionResNetV2	98.70	99.0	99.0	99.0
MobileNetV2	98.45	98.0	98.0	98.0
ResNet50	97.77	98.0	98.0	98.0
ResNet101	98.47	98.0	98.0	98.0
VGG16	98.90	99.0	99.0	99.0
VGG19	97.55	98.0	98.0	98.0
Xception	98.45	98.0	98.0	98.0

5.3 Loss and Accuracy Curve

1. **EfficienetnetB0:** The model was trained using more than 50 epochs. Less overfitting and effective learning are suggested by the training loss decreasing grad-

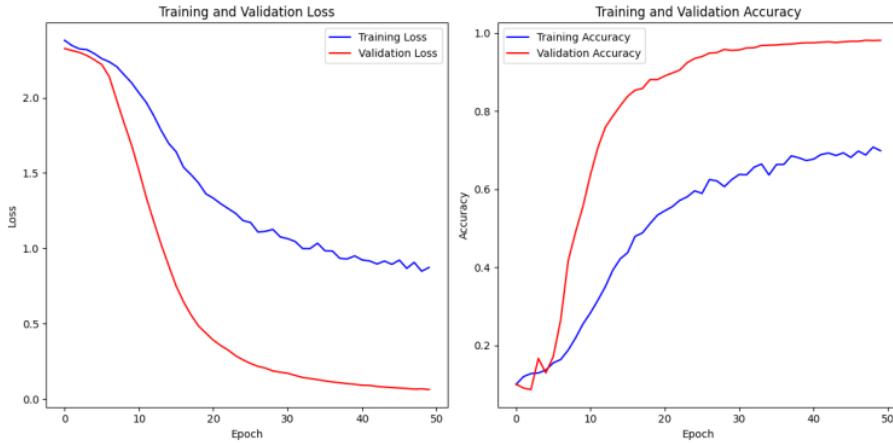


Figure 5.16: EfficientnetB0 model's loss and accuracy.

ually from 2.5 to 0.5, while the validation loss dropped quickly and stabilized at 0.1. Training accuracy increased from 10% to 85%, while validation accuracy increased sharply to stabilize at 95%. Strong and stable generalization was shown

by the convergence of the accuracy and loss curves when discerning complex patterns in the dataset. This steady performance highlights the model's capacity to identify subtle characteristics, which makes it a good fit for practical uses where dependability and accuracy are crucial. The training process has yielded consistent improvements in both metrics, indicating that the selected architecture and training strategy have been successful in producing high-performance.

2. **Inceptionv3:** The training and validation curves for the model show its performance over 30 epochs. The training loss begins around 2.5 and gradually falls to

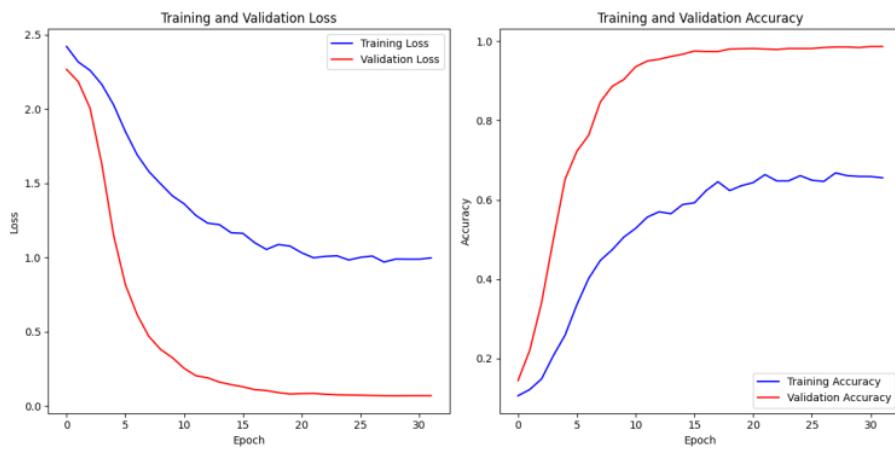


Figure 5.17: Inceptionv3 model's loss and accuracy.

around 0.5, whereas the validation loss decreases quickly and stabilizes at around 0.1, showing effective learning and minimal overfitting shown in Figure 5.17. The training accuracy starts around 10% and steadily improves to about 85%, whereas the validation accuracy starts around 10% and rapidly rises to over 95%. The convergence of the loss and accuracy curves, particularly the validation metrics, illustrates the model's durability and ability to transfer effectively to previously encountered data. The little fluctuation toward the curves' ends indicates the model's stability and competence in learning complex patterns from the dataset.

3. **Inceptionresnetv2:** The model's training and validation curves show how it performed across 25 epochs shown in Figure 5.18. The training loss begins around 2.5 and gradually falls to around 0.5, whereas the validation loss decreases quickly

and stabilizes at around 0.1, showing effective learning and minimal overfitting. While validation accuracy starts at 10% and rapidly surpasses 95%, training ac-

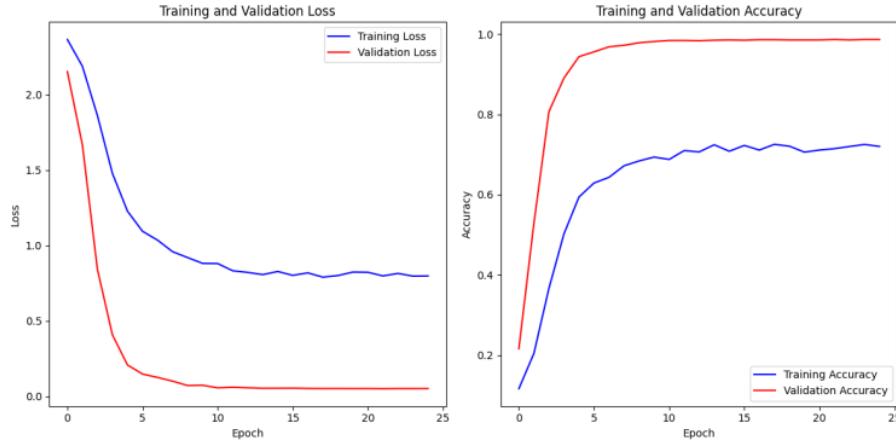


Figure 5.18: Inceptionv3 model's loss and accuracy.

curacy starts at 10% and increases gradually to 85%. The model's robustness and successful generalization to new data are demonstrated by the convergence of the accuracy and loss curves, particularly in validation metrics.

- ¹¹²
4. **MobilenetV2:** The Training and Validation curves of the MobileNetV2 [13] model show its performance over 45 epochs. The training loss begins around 2.5

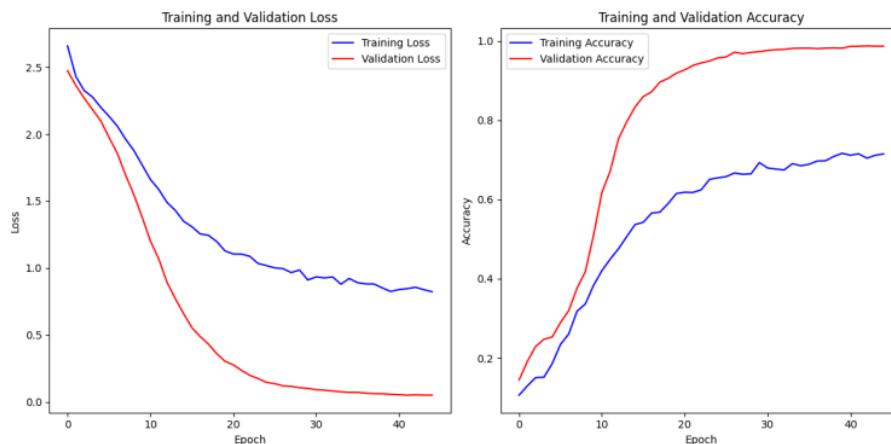


Figure 5.19: MobilenetV2 model's loss and accuracy.

and gradually falls to around 0.5, whereas the validation loss decreases quickly

and stabilizes at around 0.1, showing effective learning and minimal overfitting. The training accuracy starts around 10% and steadily improves to about 85%, whereas the validation accuracy starts around 10% and rapidly rises to over 95%.
11

The convergence of the loss and accuracy curves, particularly the validation metrics, illustrates the model's durability and ability to transfer effectively to previously encountered data. The little fluctuation toward the curves' ends indicates the model's stability and competence in learning complex patterns from the dataset.

5. **Resnet50:** The model's training and validation curves demonstrate its performance across 50 epochs. The training loss begins around 2.5 and gradually falls to

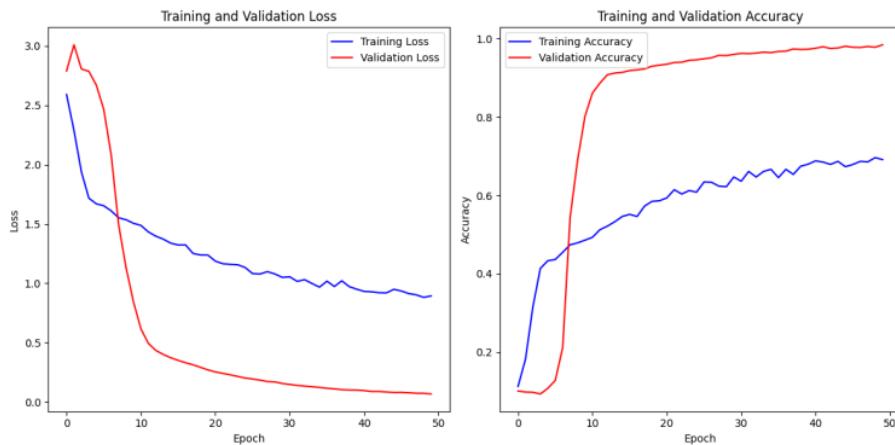


Figure 5.20: Resnet50 model's loss and accuracy.

around 0.5, whereas the validation loss decreases quickly and stabilizes at around 0.1, showing effective learning and minimal overfitting in Figure 5.20. The training accuracy starts around 10% and steadily improves to about 85%, whereas the validation accuracy starts around 10% and rapidly rises to over 95%.
11
The convergence of the loss and accuracy curves, particularly the validation metrics,
14 illustrates the model's durability and ability to transfer effectively to previously encountered data.

6. **Resnet101:** The training and validation curves for the model show its performance over 50 epochs shown in Figure 5.21. The training loss begins around 2.5 and gradually decreases to around 0.9, whereas the validation loss rapidly

decreases and stabilizes at around 0.1, showing effective learning and minimal overfitting. The training accuracy starts at 0% and steadily improves to around

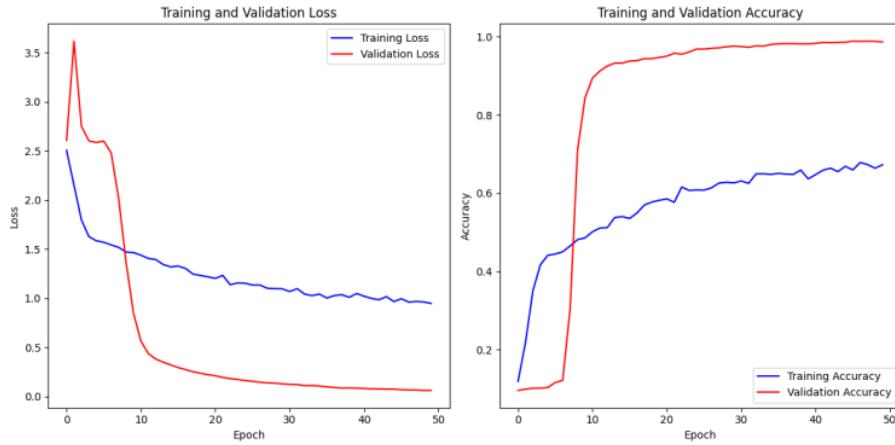


Figure 5.21: Resnet50 model's loss and accuracy.

65%, but the validation accuracy increases rapidly to approximately 97% from 0% at first. The model's robustness and successful transfer to known data are demonstrated by the convergence of the loss and accuracy curves, particularly in validation metrics. The model's steady learning of intricate dataset patterns is reflected in the low volatility near the ends of the curves.

7. **Vgg16:** The performance of the model over 23 epochs is shown in Figure 5.22. The training loss begins around 2.5 and gradually falls to around 0.5, whereas the validation loss decreases quickly and stabilizes at around 0.1, showing effective learning and minimal overfitting. The training accuracy starts around 10% and steadily improves to about 85%, whereas the validation accuracy starts around 10% and rapidly rises to over 95%. The convergence of the loss and accuracy curves, particularly the validation metrics, illustrates the model's durability and ability to transfer effectively to previously encountered data.

8. **Vgg19:** The model's training and validation curves show how it performed across 39 epochs shown in Figure 5.23. The training loss begins around 2.5 and gradually falls to around 1.0, whereas the validation loss decreases quickly and stabilizes at around 0.1, showing effective learning and minimal overfitting. The training accuracy begins around 10% and steadily improves to about 55%, whereas

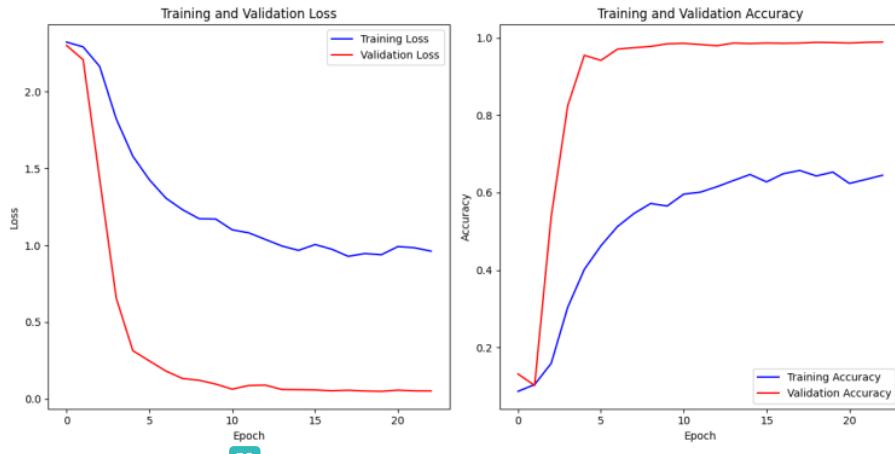


Figure 5.22: Vgg16 model's loss and accuracy.

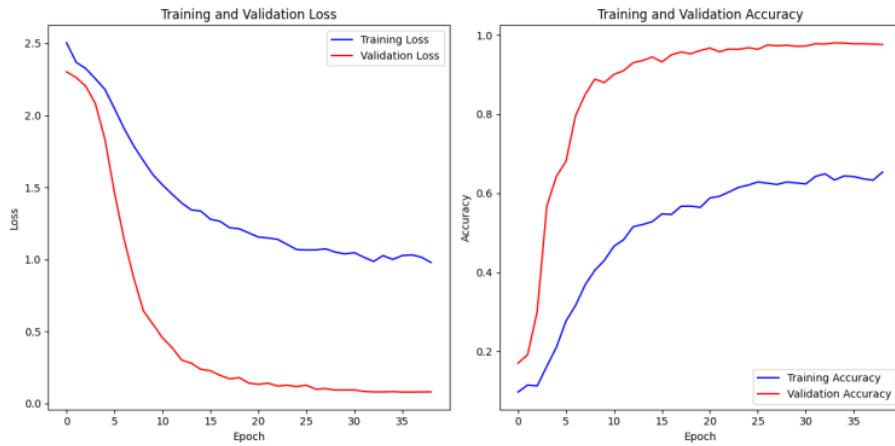


Figure 5.23: Vgg19 model's loss and accuracy.

11

the validation accuracy begins around 10% and rises sharply to around 97%. The convergence of the loss and accuracy curves, particularly the validation metrics,
14 illustrates the model's durability and ability to transfer effectively to previously encountered data. This low volatility toward the conclusion of the curves represents the model's stability and competence in learning complicated patterns from the dataset.

9. **Xception:** The training and validation curves for the model show its performance over 34 epochs. The training loss begins around 2.2 and gradually falls to around 1.0, whereas the validation loss rapidly decreases and stabilizes at around 0.1,

showing effective learning and minimal overfitting. The training accuracy begins at 5% and gradually increases to around 70%, while the validation accuracy starts at 7% and quickly exceeds 97%, as shown in Figure 5.24. This convergence

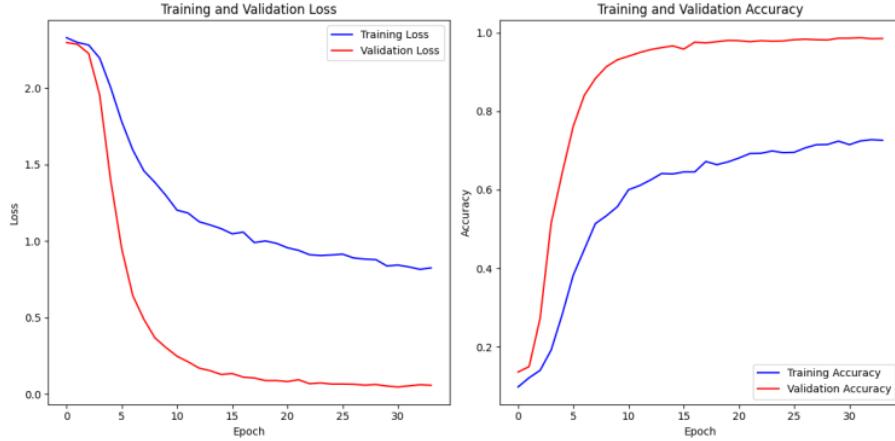


Figure 5.24: Xception model's loss and accuracy.

indicates the model's robustness and effective transfer learning. The stable, low-volatility curves towards the end highlight the model's stability and ability to learn intricate patterns from the dataset.

5.4 Result Analysis

5.4.1 Performance comparison

The accuracy of several deep learning models is compared in the bar graph shown in Figure 5.25. With an accuracy of 98.91%, VGG16 [16] stands out the most, closely followed by InceptionV3 [10] at 98.53% and InceptionResNetV2 [11] at 98.70%. Both ResNet101 [15] and MobileNetV2 [13] exhibit high accuracy levels, ranging from 98.46 to 98.48%. With competitive accuracy levels above 97%, EfficientNetB0 [9], Xception [18], and VGG19 [17] continue to outperform MobileNet [12] and ResNet50 [14], which have slightly lower accuracy levels of 97.18% and 97.78%, respectively. The superior accuracy performance of the VGG16 [16], InceptionResNetV2 [11], and InceptionV3 [10] models is demonstrated by this visualization.

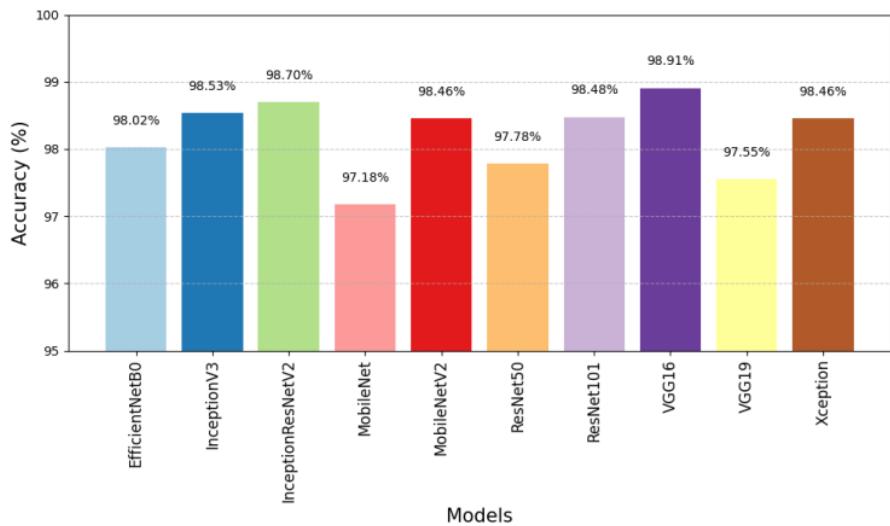


Figure 5.25: Comparison between different models

5.4.2 AUC-ROC Analysis

Receiver Operating Characteristic Area Under the Curve is referred to as ROC AUC.

The performance of the classifier across all potential classification thresholds is summed up by a single figure called the ROC AUC score. You must calculate the area under the ROC curve in order to obtain the score. The classifier's ability to discern between positive and negative classes is indicated by the ROC AUC score. It accepts numbers between 0 and 1 [31]. Plotting TPR against FPR allows AUC-ROC to evaluate classifier performance. True Positive Rate, or TPR, shows accurate positive predictions.

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negative}}$$

False Positive Rate, or FPR, quantifies inaccurately positive predictions.

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negative}}$$

The percentage of real negative cases that the model correctly classifies as negative is known as specificity. It displays the model's capacity to recognize negative examples

with accuracy[32].

$$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positives}}$$

Given discrete points $(\text{FPR}_i, \text{TPR}_i)$, the AUC can be calculated as:

$$\text{AUC} = \sum_{i=1}^{n-1} (\text{FPR}_{i+1} - \text{FPR}_i) \times \frac{\text{TPR}_i + \text{TPR}_{i+1}}{2}$$

The ROC curves and AUC for the ten models we trained will be discussed in this section : EfficientNetB0 achieved an outstanding AUC-ROC score, demonstrating its exceptional accuracy. Similarly, InceptionV3 exhibited high performance, slightly surpassing EfficientNetB0. InceptionResNetV2, while still effective, presented a comparatively lower score. MobileNetV2 maintained competitive performance. Additionally, ResNet50 and ResNet101 achieved remarkable AUC-ROC scores. VGG16 and VGG19 also performed well. Xception excelled with a strong performance. These results un-

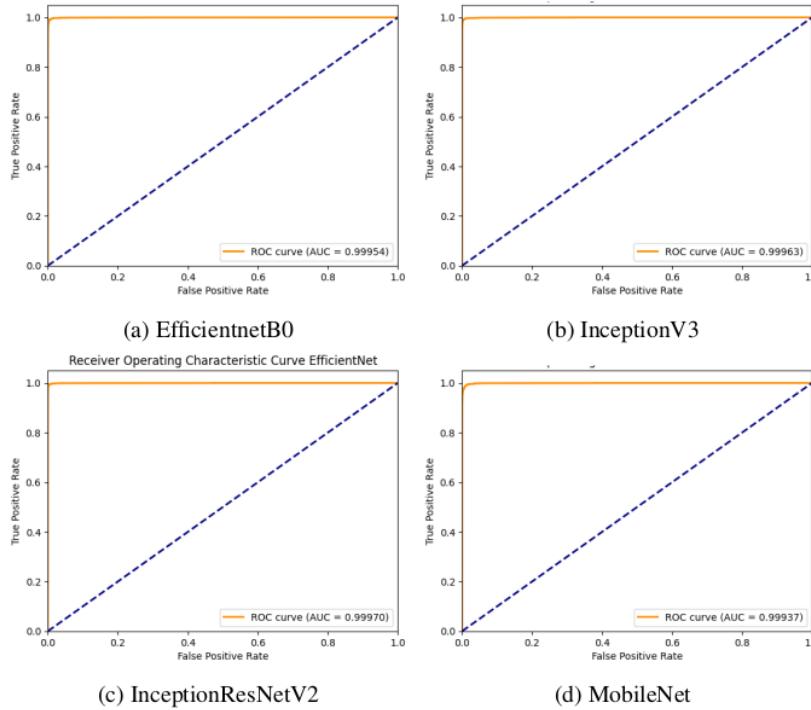


Figure 5.26: AUC-ROC curves for EfficientnetB0, InceptionV3 , InceptionResNetV2 , MobileNet models.

derscore the superior accuracy of EfficientNetB0, InceptionV3, ResNet101, and Xception in recognizing Bangla digits via air writing.

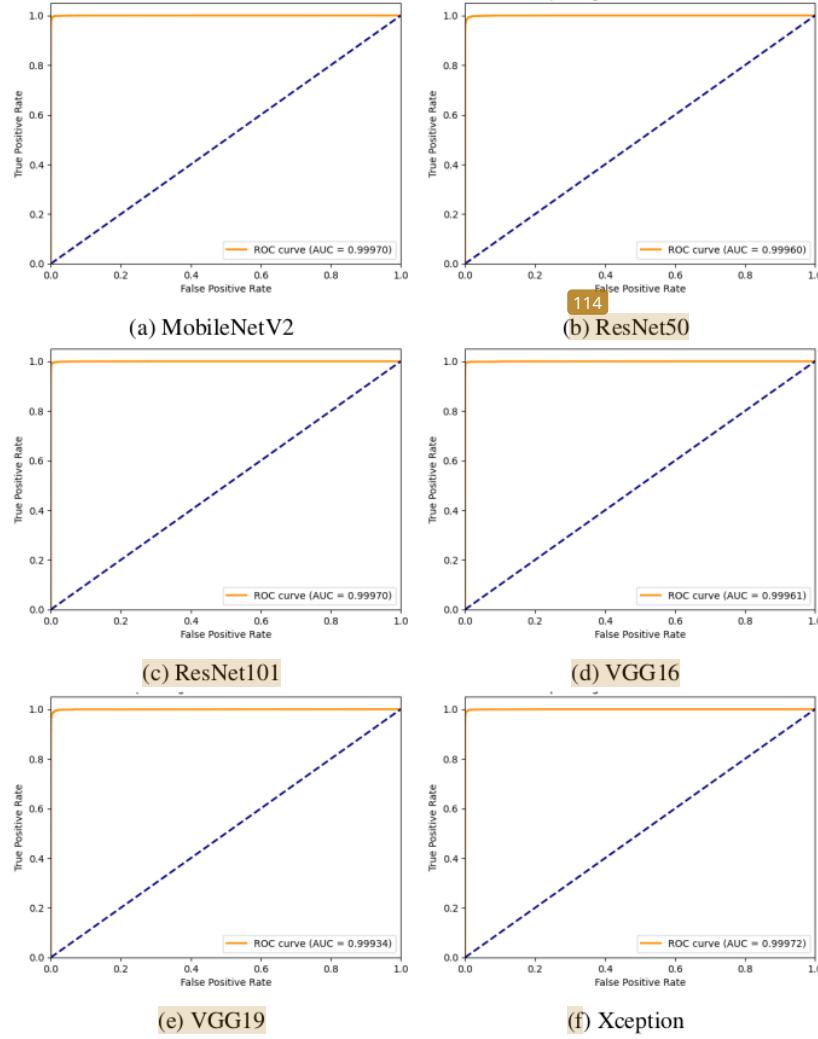


Figure 5.27: AUC-ROC curves for Mobilenet, Resnet50, Resnet101, Vgg16, Vgg19, Xception models.

5.5 Summary

The findings and analysis of the implemented Bangla digit on air-writing recognition models demonstrate their performance measures, such as accuracy and loss patterns throughout epochs. Detailed assessments such as loss and accuracy graphs, AUC-ROC

analysis, and confusion matrices are provided. The performance of multiple pretrained models in recognizing Bangla digits is compared, indicating their usefulness and superiority over current datasets. This evaluation demonstrates the effectiveness of custom-designed models in enhancing air-writing digit recognition technology.

99
CHAPTER 6

Conclusions and Future Works

6.1 Conclusions

This study creates datasets using a creative method writing in the air. This thesis aimed to address the significant gap in existing research by creating a comprehensive and diverse dataset of 5314 Bangla digit images contributed by 200 participants. Previous studies lacked a robust dataset, which limited the effectiveness and accuracy of Bangla digit recognition systems. Python [19] scripts automated data collection and formatted the collected data into a standard 224x224 format, increasing efficiency and repeatability. EfficientNetB0 [9], InceptionV3 [10], Mobilenet [12], Mobilenetv2 [13], Resnet50 [14], Resnet101 [16], Vgg19 [17], Xception [18] were the ten deep learning models that were evaluated. VGG16 [16], and InceptionResNetV2 [11] displayed the best accuracy among them. This study demonstrates how air writing may be used to build datasets and enhance Bangla script recognition. It has a useful benefit over conventional techniques. The findings support the creation of educational materials and assistive technology, as well as linguistic and cultural variety in digital applications. Future computer vision research can build on this work, which also emphasizes the value of inclusive datasets in achieving robust model performance in practical scenarios.

6.2 Future Recommendations

In the future, the following tasks can be performed:

- 42
1. Increasing the number of various classes.
 2. Increasing the number of images per class.

3. Building lightweight models suitable for our dataset.
 4. We can integrate this technology into various applications and real-time tools to broaden its utility and effectiveness
-

REFERENCES

- [1] New World Encyclopedia. (n.d.) Bengali language. Accessed: 2024-07-08. [Online]. Available: <https://www.newworldencyclopedia.org/entry/Bengali-language>
- [2] C. Saha, F. Masuma, K. Ahammad, C. S. Muzammel, and M. Mohibullah, “Real time bangla digit recognition through hand gestures on air using deep learning and opencv,” *Int. J. Curr. Sci. Res. Rev*, vol. 5, 2022.
- [3] K. M. Nahar, I. Alsmadi, R. E. Al Mamlook, A. Nasayreh, H. Gharaibeh, A. S. Almuflah, and F. Alasim, “Recognition of arabic air-written letters: Machine learning, convolutional neural networks, and optical character recognition (ocr) techniques,” *Sensors*, vol. 23, no. 23, p. 9475, 2023.
- [4] M. S. Alam, K.-C. Kwon, M. A. Alam, M. Y. Abbass, S. M. Imtiaz, and N. Kim, “Trajectory-based air-writing recognition using deep neural network and depth sensor,” *Sensors*, vol. 20, no. 2, p. 376, 2020.
- [5] H. Mahmud, R. Islam, and M. K. Hasan, “On-air english capital alphabet (eca) recognition using depth information,” *The Visual Computer*, vol. 38, no. 3, pp. 1015–1025, 2022.
- [6] M. Chen, G. AlRegib, and B.-H. Juang, “Air-writing recognition—part ii: Detection and recognition of writing activity in continuous stream of motion data,” *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 3, pp. 436–444, 2015.
- [7] ———, “Air-writing recognition—part i: Modeling and recognition of characters, words, and connecting motions,” *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 3, pp. 403–413, 2015.
- [8] S. Mukherjee, S. A. Ahmed, D. P. Dogra, S. Kar, and P. P. Roy, “Fingertip detection and tracking for recognition of air-writing in videos,” *Expert Systems with Applications*, vol. 136, pp. 217–229, 2019.

- [9] V.-T. Hoang and K. Jo, “Practical analysis on architecture of efficientnet,” *2021 14th International Conference on Human System Interaction (HSI)*, pp. 1–4, 2021.
- [10] J. Cao, M. Yan, Y. Jia, X. Tian, and Z. Zhang, “Application of a modified inception-v3 model in the dynasty-based classification of ancient murals,” *EURASIP Journal on Advances in Signal Processing*, vol. 2021, pp. 1–25, 2021.
- [11] C. Peng, Y. Liu, X. Yuan, and Q. Chen, “Research of image recognition method based on enhanced inception-resnet-v2,” *Multimedia Tools and Applications*, vol. 81, no. 24, pp. 34 345–34 365, 2022.
- [12] D. Sinha and M. El-Sharkawy, “Thin mobilenet: An enhanced mobilenet architecture,” in *2019 IEEE 10th annual ubiquitous computing, electronics & mobile communication conference (UEMCON)*. IEEE, 2019, pp. 0280–0285.
- [13] K. Dong, C. Zhou, Y. Ruan, and Y. Li, “Mobilenetv2 model for image classification,” in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*. IEEE, 2020, pp. 476–480.
- [14] B. Koonce and B. Koonce, “Resnet 50,” *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization*, pp. 63–72, 2021.
- [15] Q. Zhang, “A novel resnet101 model based on dense dilated convolution for image classification,” *SN Applied Sciences*, vol. 4, pp. 1–13, 2022.
- [16] S. Tammina, “Transfer learning using vgg-16 with deep convolutional neural network for classifying images,” *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, no. 10, pp. 143–150, 2019.
- [17] M. Shaha and M. Pawar, “Transfer learning for image classification,” in *2018 second international conference on electronics, communication and aerospace technology (ICECA)*. IEEE, 2018, pp. 656–660.
- [18] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.
- [19] W. Python, “Python,” *Python releases for windows*, vol. 24, 2021.

- [20] G. Bradski, “The opencv library.” *Dr. Dobb’s Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.
- [21] N. Ketkar and N. Ketkar, “Introduction to keras,” *Deep learning with python: a hands-on introduction*, pp. 97–111, 2017.
- [22] TensorFlow. (n.d.) Imagedatagenerator. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
- [23] M. Abadi, “Tensorflow: learning functions at scale,” in *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, 2016, pp. 1–1.
- [24] E. Bisong and E. Bisong, “Google colaboratory,” *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pp. 59–64, 2019.
- [25] P. S. Foundation, “Python 3.10.12 release,” 2023, accessed: 2024-07-09. [Online]. Available: <https://www.python.org/downloads/release/python-31012/>
- [26] B. M. Randles, I. V. Pasquetto, M. S. Golshan, and C. L. Borgman, “Using the jupyter notebook as a tool for open science: An empirical study,” in *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE, 2017, pp. 1–2.
- [27] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: a structure for efficient numerical computation,” *Computing in science & engineering*, vol. 13, no. 2, pp. 22–30, 2011.
- [28] C. Lugaressi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee *et al.*, “Mediapipe: A framework for building perception pipelines,” *arXiv preprint arXiv:1906.08172*, 2019.
- [29] A. Clark *et al.*, “Pillow (pil fork) documentation,” *readthedocs*, 2015.
- [30] ImageNet, “ImageNet large scale visual recognition challenge,” <https://www.image-net.org/>, 2009, accessed: 2024-07-09.

- [31] E. AI, “Roc curve and auc explained,” <https://www.evidentlyai.com/classification-metrics/explain-roc-curve>, Accessed: 2024-07-10.
- [32] GeeksforGeeks. (n.d.) Auc-roc curve. [Online; accessed July 10, 2024]. [Online]. Available: <https://www.geeksforgeeks.org/auc-roc-curve/>

APPENDIX A

Data Collection :

```
1 import cv2
2 import numpy as np
3 import mediapipe as mp
4 from enum import Enum, auto
5 import csv
6 import datetime
7 import glob
8 import copy
9 import os
10 import sys
11 
12 mp_drawing = mp.solutions.drawing_utils
13 mp_hands = mp.solutions.hands
14
15 n_collected = 0
16
17 class Mode(Enum):
18     """
19     """
20     DRAW = auto()
21     #ERASE = auto()
22     #CLEAR = auto()
23     MOVE = auto()
24
25
26 def draw(src, pt1, pt2):
27     """ This is a function for drawing on an image.
28     Args:
29         src (numpy.array): Image for drawing
30         pt1 (tuple): The starting point of the line to draw
31         pt2 (tuple): The ending point of the line to draw
32     """
33     if pt1 != (-5000, -5000) and pt2[0] != (-5000, -5000):
34         src = cv2.line(src, pt1, pt2, (0, 0, 255), 10)
35
36
37 def erase(src, point):
38     """ This is a function to erase the lines in the image.
```

```

39     Args:
40         src (numpy.array): Image for erase the line
41         point (tuple): The center point of eraser
42     """
43     src = cv2.circle(src, point, 100, (255, 255, 255), -1)
44
45
46 def clear(src):
47     """ This is a function to make the image blank.
48     Args:
49         src (numpy.array): Image for clear
50     """
51     pt1 = (0, 0)
52     pt2 = (width, height)
53     color = (255, 255, 255)
54     src = cv2.rectangle(src, pt1, pt2, color, -1)
55
56
57 def printMode(mode, src):
58     """ This is a function to print current mode.
59     Args:
60         mode (Mode): mode for print
61         src (numpy.array): Image for print mode
62     """
63     #Fill in the text printed in the previous frame.
64     src = cv2.rectangle(src, (0, 0), (430, 110), (255, 255, 255), -1)
65     #put text in src
66     src = cv2.putText(src, mode.name, (40, 100), cv2.FONT_HERSHEY_SIMPLEX, 4, (0,
67                         0, 0), 1, cv2.LINE_AA) #97
68
69 def action(digitLength, pt1, pt2, src):
70     """ This is a function to select a mode and execute it.
71     Args:
72         digitLength (numpy.array): Used for select a mode
73         pt1 (tuple): Used for draw function
74         pt2 (tuple): Used for draw function or erase function
75         src (numpy.array): Used for all function to draw something.
76
77     Return:
78         mode (Mode): return the executed mode
79     """
80     if digitLength[2] // digitLength[5] < 3 and digitLength[1] // digitLength[5] >
81             3:
82         mode = Mode.DRAW
83         draw(src, pt1, pt2)
84     # elif (digitLength[:5] // digitLength[5] < 3).all():
85     #     mode = Mode.ERASE
86     #     erase(src, pt2)

```

```

86     # elif (digitLength[:5] // digitLength[5] > 2).all():
87     #     mode = Mode.CLEAR
88     #     clear(src)
89     else:
90         mode = Mode.MOVE
91     printMode(mode, src)
92
93     return mode
94
95 def write_data(dir_path, data, img):
96     with open(dir_path + f'{datetime.datetime.now():%Y%m%d%H%M%S}.csv', 'w',
97               newline='') as f: 4
98         writer = csv.writer(f)
99         writer.writerow(['x', 'y'])
100        for d in data:
101            writer.writerow([d[0], d[1]])
102        #cv2.imwrite(dir_path + 'test.jpg', img)
103        print(str(n_collected) + " time : " + "complete writing data")
104
105 # create save directories
106 print("please input your name")
107 user_name = input()
108 print("What character do you collect?")
109 collected_char = input()
110
111 if not os.path.isdir("./collected_data/" + user_name):
112     os.mkdir("./collected_data/" + user_name)
113 else:
114     print("[warning] same user name directory exists")
115
116 if not os.path.isdir("./collected_data/" + user_name + '/' + collected_char):
117     os.mkdir("./collected_data/" + user_name + '/' + collected_char)
118 else:
119     print("[warning] same character name directory exists")
120
121 save_dir = "./collected_data/" + user_name + '/' + collected_char + '/'
122
123 # For webcam input:
124 cap = cv2.VideoCapture(0)
125 WIDTH: int = 960
126 HEIGHT: int = 720 18
127 cap.set(cv2.CAP_PROP_FRAME_WIDTH, WIDTH)
128 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, HEIGHT)
129
130 data_history = []
131 pre_mode = None
132 height, width = 0, 0
133 # src = cv2.imread('whitel.png', -1)
134 pt1, pt2 = (-5000, -5000), (-5000, -5000)

```

```

134
135
136 #with mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5) as
137     hands:
138         hands = mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5)
139         while cap.isOpened():
140             success, image = cap.read()
141             #print(image.shape)
142             if not success:
143                 print("Ignoring empty camera frame.")
144                 continue
145
146             # 4
147             # Flip the image horizontally for a later selfie-view display, and convert
148             # the BGR image to RGB.
149             image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
150             # To improve performance, optionally mark the image as not writeable to
151             # pass by reference.
152             image.flags.writeable = False
153             results = hands.process(image)
154
155             # Draw the hand annotations on the image.
156             image.flags.writeable = True
157             image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
158             if height != image.shape[0] or width != image.shape[1]:
159                 height, width = image.shape[:2]
160                 print(height, width)
161                 src = np.ones((height, width, 3), np.uint8) * 255
162
163             # 48
164             if results.multi_hand_landmarks:
165                 for hand_landmarks in results.multi_hand_landmarks:
166                     mp_drawing.draw_landmarks(
167                         image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
168
169                     # collect the data
170                     data = []
171                     for landmark in hand_landmarks.landmark:
172                         data.append((landmark.x * width, landmark.y * height))
173                     data = np.array(data)
174
175                     digitLength = np.array([[np.linalg.norm(data[i*4][:2] - data[0][:2]) for
176                                         i in range(1,6)]])
177                     digitLength = np.append(digitLength, np.linalg.norm(data[1][:2] - data
178                                         [0][:2]))
179                     pt2 = (int(data[8, 0]), int(data[8, 1]))

```

```

180         data_history.append(pt1)
181     #elif mode is not Mode.DRAW and pre_mode is Mode.DRAW: #For double stroke
182     #     data_history.append([-1,-1])
183
184     # if mode is Mode.CLEAR:
185     #     data_history.clear()
186
187     pre_mode = mode
188
189
83
190     blended = cv2.addWeighted(src1=image, alpha=0.7, src2=src, beta=0.3, gamma=0)
191     #src[:, :, 3] = np.where(np.all(src == 255, axis=-1), 0, 255)
192     cv2.imshow('MediaPipe Hands', blended)
193
92
194     k = cv2.waitKey(5)
195     if k == 27:           # wait for ESC key to exit
196         break
197     elif k == ord('p'):
198         print("What character do you collect?")
199         collected_char = input()
200         if not os.path.isdir("./collected_data/" + user_name + "/" +
201             collected_char):
202             os.mkdir("./collected_data/" + user_name + "/" + collected_char)
203         else:
204             print("[warning] same character name directory exists")
205             save_dir = "./collected_data/" + user_name + "/" + collected_char + "/"
206             n_collected = 0
207             print()
208         elif k == ord('c'):
209             clear(src)
210             data_history.clear()
211             elif k == ord('s'): # wait for 's' key to save
212                 n_collected += 1
213                 write_data(save_dir, data_history, blended)
214                 clear(src)
215             data_history.clear()
216 hands.close()
217 cap.release()
218 cv2.destroyAllWindows()

```

Collected Data processing:

```

29
1 import os
2 import shutil
3
33
4 def merge_data(source_folder, destination_folder):
5     if not os.path.exists(destination_folder):
6         os.makedirs(destination_folder)
7

```

```

19
8   for person_folder in os.listdir(source_folder):
9       person_folder_path = os.path.join(source_folder, person_folder)
10      if os.path.isdir(person_folder_path):
11          for class_folder in range(10):
12              class_source_folder = os.path.join(person_folder_path, str(
13                  class_folder))
14              if os.path.exists(class_source_folder):
15                  for root, dirs, files in os.walk(class_source_folder):
16                      for index, file in enumerate(files):
17                          source_file = os.path.join(root, file)
18                          destination_file = os.path.join(destination_folder,
19                                str(class_folder), f"{person_folder}_{class_folder}"
20                                )_{index}{os.path.splitext(file)[1]}")
21                          if not os.path.exists(os.path.dirname(destination_file
22 )):
23                              os.makedirs(os.path.dirname(destination_file))
24                          shutil.copy(source_file, destination_file)
25
26 # Source folder containing data for each person
27 source_folder = 'collected_data'
28 # Destination folder where merged data will be stored
29 destination_folder = 'merged_data'
30 merge_data(source_folder, destination_folder)

```

Image Rename:

```

1     import os
2 def change_filenames(directory):
3     # Iterate through each sub-folder (0-9)
4     for subdir in range(10):
5         subdir_path = os.path.join(directory, str(subdir))
6         # Check if the sub-folder exists
7         if os.path.exists(subdir_path): [29]
8             # Iterate through each file in the sub-folder
9             for idx, filename in enumerate(os.listdir(subdir_path)):
10                 # Construct the new filename
11                 new_filename = f"{subdir}_{idx+1}.jpg" # Assuming JPG images,
12                     change extension accordingly if needed
13                 [46] rename the file
14                 os.rename(os.path.join(subdir_path, filename), os.path.join(
15                     subdir_path, new_filename))
16                 print(f"Renamed {filename} to {new_filename}")
17 # Directory where collected_data folder is located
18 directory = "small_data_1"
19 change_filenames(directory)

```

Data Loading Function:

```

24 !pip install -U -q PyDrive
25 from pydrive.auth import GoogleAuth
26 from pydrive.drive import GoogleDrive
27 from google.colab import auth
28 from oauth2client.client import GoogleCredentials
29 auth.authenticate_user()
30 gauth = GoogleAuth()
31 gauth.credentials = GoogleCredentials.get_application_default()
32 drive = GoogleDrive(gauth)
33 fid = drive.ListFile({'q': "title='collected_data.zip'"}).GetList()[0]['id']
34 f = drive.CreateFile({'id': fid})
35 f.GetContentFile('collected_data.zip')
36 f.keys()
37 !unzip collected_data.zip
38
39 import os
40 from PIL import Image
41 import matplotlib.pyplot as plt
42
43 folder_dir = '/content/collected_data'
44
45 # Get the list of folders in the directory
46 folders = os.listdir(folder_dir)
47
48 # Iterate through each folder
49 for folder in folders:
50     # Get the list of images in the current folder
51     images = os.listdir(os.path.join(folder_dir, folder))
52
53     # Select the first image from the folder
54     if images:
55         image_path = os.path.join(folder_dir, folder, images[0])
56
57         # Open the image using PIL
58         img = Image.open(image_path)
59
60         # Display the image using matplotlib
61         plt.imshow(img)
62         plt.title(f'Image from folder {folder}')
63         plt.axis('off')    # Hide axis
64         plt.show()

```

Data Augmentation:

```

1 train_datagen = ImageDataGenerator(
2     rescale=1./255,
3     rotation_range=60,
4     width_shift_range=0.5,
5     height_shift_range=0.5,
6     shear_range=0.5,
7     zoom_range=0.5,
8     horizontal_flip=True,
9     validation_split=0.3 # Increase validation split to 30%
10 )

```

Model Training:

```

1 # Define callbacks
2 checkpoint = ModelCheckpoint('best_model.keras', monitor='val_accuracy',
3     save_best_only=True, mode='max')
4 early_stopping = EarlyStopping(monitor='val_loss', patience=3,
5     restore_best_weights=True)
6 reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2, min_lr=1
7     e-7)

```

Model Evaluation:

```

1 import matplotlib.pyplot as plt
2 import seaborn as sns
3 from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
4     , roc_curve, auc, accuracy_score
5
6 # Check Data Generators
7 test_generator = test_datagen.flow_from_directory(
8     'D:/apple/input/saadnewdataset/dataset', # Make sure this directory is correct
9     target_size=img_size,
10    batch_size=batch_size,
11    class_mode='categorical',
12    shuffle=False # Ensure no shuffling for consistency in predictions
13 )
14 # 1. Evaluate model on the test set
15 test_loss, test_acc = model.evaluate(test_generator)
16 print(f'Test accuracy from evaluate: {test_acc * 100:.2f}%')
17
18 # Evaluate the model on test set
19

```

```

65
20 predictions = model.predict(test_generator)
21 y_true = test_generator.classes
22 y_pred = np.argmax(predictions, axis=1)
23
24 # Calculate accuracy score
25 accuracy = accuracy_score(y_true, y_pred)
26 print("Accuracy:", accuracy)
27
28 # Generate confusion matrix
29 conf_matrix = confusion_matrix(y_true, y_pred)
30
31 # Plot confusion matrix using Seaborn
32 plt.figure(figsize=(15, 15))
33 sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues')
34 plt.xlabel('Predicted labels')
35 plt.ylabel('True labels')
36 plt.title('Confusion Matrix')
37 plt.show()
38
39 # Classification report
40 print("Classification Report:")
41 print(classification_report(y_true, y_pred))
42
43 # Calculate overall ROC AUC score
44 roc_auc = roc_auc_score(tf.keras.utils.to_categorical(y_true), predictions,
    average='macro')
45 print("Overall ROC AUC Score:", roc_auc)
46
47 # Plot ROC curve
48 fpr, tpr, _ = roc_curve(tf.keras.utils.to_categorical(y_true).ravel(), predictions
    .ravel())
49 roc_auc = auc(fpr, tpr)
50
51 plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.5f)' %
    roc_auc)
52 plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
53 plt.xlim([0.0, 1.0])
54 plt.ylim([0.0, 1.05])
55 plt.xlabel('False Positive Rate')
56 plt.ylabel('True Positive Rate')
57 plt.title('Receiver Operating Characteristic Curve EfficientNet')
58 plt.legend(loc="lower right")
59 plt.show()
60
61 # Plot loss and accuracy per epoch
62 plt.figure(figsize=(12, 6))
63
64 # Plot training loss
65 plt.subplot(1, 2, 1)

```

```
66 plt.plot(history.history['loss'], label='Training Loss', color='blue')
67 plt.plot(history.history['val_loss'], label='Validation Loss', color='red')
68 plt.title('Training and Validation Loss')
69 plt.xlabel('Epoch')
70 plt.ylabel('Loss')
71 plt.legend()
72
73 # Plot training accuracy
74 plt.subplot(1, 2, 2)
75 plt.plot(history.history['accuracy'], label='Training Accuracy', color='blue')
76 plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='red'
    )
77 plt.title('Training and Validation Accuracy')
78 plt.xlabel('Epoch')
79 plt.ylabel('Accuracy')
80 plt.legend()
81
82 plt.tight_layout()
83 plt.show()
```

APPENDIX B

Complex Engineering Problems (CP) and Complex Engineering Activities (CA)

Analysis

Title: Bangla Digit Recognition on Air Writing Using Deep Learning Techniques

Attainment of Complex Engineering Problem (CP)

S.L.	CP No.	Attai-nment	Remarks
1.	P1: Depth of Knowledge Required	Yes	K3 (Engineering Fundamentals): Python described in Chapter 4 Art No: 4.2.2.
			K4 (Engineering Specialization): Knowledge about Deep Learning described in Chapter 3 Art No: 3.4.
			K5 (Design): Methodology flowchart described in Chapter 3 Art No: 3.5.
			K6 (Technology): TensorFlow,image processing, keras,OpenCV etc. described in Chapter 4 Art No: 4.2.
			K8 (Research): Is studying to find the current limitations of existing research in Chapter 2.
2.	P2: Range of Conflicting Requirements	Yes	Dataset scarcity addressed through creation efforts. described in Chapter 4 Art No: 4.3

3.	P3: Depth of Analysis Required	Yes	deeply studied recent research to identify its limitations and developed a comprehensive dataset to address them described in Chapter 2 and Chapter 4 Art No: 4.3 .
4.	P4: Familiarity of Issues	Yes	Writing on the air is not very common. As CSE students, we've worked with this technology described in Chapter 1 .
5.	P5: Extent of Applicable Codes	Yes	applied the standard process of deep learning in Chapter 3.
6.	P6: Extent of Stakeholder Involvement and Conflicting Requirements	Yes	Researchers, individuals with disabilities, and the education sector are involved as stakeholders described in Chapter 1 Art No: 1.7.
7.	P7: Interdependence	Yes	applied standard process of deep learning described in Chapter 3 Art No: 3.3 .

Mapping of Complex Engineering Activities (CA)

S.L.	CA No.	Attainment	Remarks
1.	A1: Range of resources	Yes	Technology such as Keras, TensorFlow, deep learning models, and image processing has been used for this describe in Chapter 3 Art No: 3.2 and 3.4
2.	A2: Level of interaction	Yes	Various challenges encountered during this thesis work described in Chapter 1 Art No: 1.6.
3.	A3: Innovation	Yes	Building a new dataset and model for recognizing air written Bangla digits described in Chapter 3 and Chapter 4.

4.	A4: Consequences for Society and the Environment	Yes	enhances accessibility, promotes interactive learning, and supports Bangla language preservation described in Chapter 1 Art No: 1.5.
5.	A5: Familiarity	Yes	The current tasks involve limitations in datasets, prompting the creation of a comprehensive dataset described in Chapter 2 and Chapter 4.

BIOGRAPHY

of
Md. Yeamin Islam Sakib



Md. Yeamin Islam Sakib has a strong desire to learn and achieve academic excellence, which drives his intense passion for computer science and engineering. Born into a family that places a high emphasis on education, Sakib has always strived for academic achievement. Sakib has an excellent academic record and is currently enrolled in the ⁶⁸ Bangladesh Army University of Science and Technology, Saidpur, Nilphamari, where she is pursuing a B.Sc. in Computer Science and Engineering. He received flawless GPAs of 5.00 for both his HSC at Rajuk Uttara Model College and his SSC at Safiuddin Sarker Academy and College. Sakib's academic journey is complemented by his active participation in various technical workshops, seminars, and competitions. Sakib has developed a keen interest in Software development, Cybersecurity, and Machine learning. Sakib's leadership and practical skills highlight his potential as a future leader in the field. Sakib's dedication and hard work are making a meaningful impact in Computer Science and Engineering.

Research and Publications:

Thesis:

- **Title:** Implementing Real-Time Bangla Digit Recognition in Air Using Deep Learning and OpenCV.

Md. Yeamin Islam Sakib

+8801705118122

baust.cse.200201015@gmail.com

Tongi ,Gazipur

BIOGRAPHY

of



Harishankar Barman

Harishankar Barman's journey in Computer Science and Engineering is marked by his passion for learning, academic excellence, and a drive to make a positive impact. Born to Karuna Kanto Barman and Chandana Rani, he grew up in a family that values education and curiosity.

Currently pursuing a B.Sc. in Computer Science and Engineering at Bangladesh Army University of Science and Technology, Saidpur, Nilphamary. ³⁵ He laid a strong educational foundation with perfect GPAs of 5.00 in both his SSC from Afan Ullah High School, Rangpur, and his HSC from Police Lines School and College, Rangpur.

⁴² Harishankar is actively engaged in research in Parallel Computing, Machine Learning, Deep Learning, Natural Language Processing, and Image Processing. His work showcases his commitment to advancing technology and solving complex problems.

Beyond academics, Harishankar is a dynamic individual contributing to research and extracurricular activities. His leadership skills and practical application of knowledge highlight his role as a future leader in the field. Harishankar's journey reflects his dedication, hard work, and a commitment to excellence, making a meaningful impact in Computer Science and Engineering.

Research and Publications:

Thesis:

- **Title:** Implementing Real-Time Bangla Digit Recognition in Air Using Deep Learning and OpenCV.

Harishankar Barman

+8801878250186

baust.cse.200201001@gmail.com

Mahigonj, Rangpur

BIOGRAPHY

of

Musharrat Binte Alam Mithy



Musharrat Binte Alam Mithy, born in Rangpur, daughter of Md. Mohashin Alam and Selina Akhter, who is a dedicated Computer Science enthusiast and researcher. She is pursuing her B.Sc. Engineering in Computer Science and Engineering from Bangladesh Army University of Science and Technology, Saidpur, Nilphamari, achieving CGPA of 2.50. She completed her SSC from Cantonment Public School and College, Rangpur with GPA 5.00 and HSC from Cantonment Public School and College, Rangpur with GPA 5.00.

In academics, Mithy has demonstrated her commitment to advancing knowledge through her research endeavors. She is actively engaged in research in the field of Parallel Computing, Machine Learning and Deep learning.

Mithy played an important work in her thesis research Enhancing Search Efficiency for Large Linked Lists with Multithreading and Caching. Her other works through academics showed her proficiency in Python, Google Colab, Keras, TensorFlow, architectures to built websites and web applications.

Research and Publications:

Thesis:

- **Title:** Implementing Real-Time Bangla Digit Recognition in Air Using Deep Learning and OpenCV..

Musharrat Binte Alam Mithy

+8801723012605

baust.cse.200201006@gmail.com

Islambug, Rangpur



PRIMARY SOURCES

1	www.researchgate.net Internet Source	1 %
2	fr.overleaf.com Internet Source	1 %
3	www.mdpi.com Internet Source	1 %
4	qiita.com Internet Source	1 %
5	Submitted to Daffodil International University Student Paper	1 %
6	"Proceedings of 4th International Conference on Artificial Intelligence and Smart Energy", Springer Science and Business Media LLC, 2024 Publication	<1 %
7	Submitted to Toronto Business College Student Paper	<1 %
8	Submitted to University of London External System Student Paper	<1 %

9	core.ac.uk Internet Source	<1 %
10	doaj.org Internet Source	<1 %
11	www2.mdpi.com Internet Source	<1 %
12	www.ncbi.nlm.nih.gov Internet Source	<1 %
13	Submitted to Associatie K.U.Leuven Student Paper	<1 %
14	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1 %
15	Submitted to Liverpool John Moores University Student Paper	<1 %
16	www.slideshare.net Internet Source	<1 %
17	Submitted to Faculty of Organization and Informatics Student Paper	<1 %
18	github.com Internet Source	<1 %
19	Submitted to University of Glamorgan Student Paper	<1 %

20	programtalk.com Internet Source	<1 %
21	Sohom Mukherjee, Sk. Arif Ahmed, Debi Prosad Dogra, Samarjit Kar, Partha Pratim Roy. "Fingertip detection and tracking for recognition of air-writing in videos", Expert Systems with Applications, 2019 Publication	<1 %
22	Submitted to De Montfort University Student Paper	<1 %
23	link.springer.com Internet Source	<1 %
24	Submitted to Birla Institute of Technology and Science Pilani Student Paper	<1 %
25	repozitorij.etfos.hr Internet Source	<1 %
26	Submitted to UPN Veteran Jawa Timur Student Paper	<1 %
27	osuva.uwasa.fi Internet Source	<1 %
28	www.cnblogs.com Internet Source	<1 %
29	www.coursehero.com Internet Source	<1 %

30	Submitted to University of Westminster Student Paper	<1 %
31	Submitted to UOW Malaysia KDU University College Sdn. Bhd Student Paper	<1 %
32	Submitted to University of Hull Student Paper	<1 %
33	Submitted to University Politehnica of Bucharest Student Paper	<1 %
34	hooshio.com Internet Source	<1 %
35	www.ijaceeonline.com Internet Source	<1 %
36	Submitted to University of San Diego Student Paper	<1 %
37	Abdelaziz Testas. "Distributed Machine Learning with PySpark", Springer Science and Business Media LLC, 2023 Publication	<1 %
38	Submitted to University of Sydney Student Paper	<1 %
39	Babeş-Bolyai University Publication	<1 %
Submitted to Indian School of Business		

40

<1 %

41

[dadun.unav.edu](#)

Internet Source

<1 %

42

[dokumen.pub](#)

Internet Source

<1 %

43

Submitted to Aligarh Muslim University,
Aligarh

Student Paper

<1 %

44

[gist.github.com](#)

Internet Source

<1 %

45

Submitted to Coventry University

Student Paper

<1 %

46

Submitted to Wright State University

Student Paper

<1 %

47

[itgit.cu.ac.kr](#)

Internet Source

<1 %

48

[www.mekinfo.net](#)

Internet Source

<1 %

49

Submitted to University of Liverpool

Student Paper

<1 %

50

Submitted to University of New Haven

Student Paper

<1 %

51

[wmi.amu.edu.pl](#)

Internet Source

<1 %

-
- 52 Aura-Loredana Popescu, Nirvana Popescu, Ciprian Dobre, Elena-Simona Apostol, Decebal Popescu. "IoT and AI-Based Application for Automatic Interpretation of the Affective State of Children Diagnosed with Autism", Sensors, 2022 <1 %
Publication
-
- 53 Hasan Mahmud, Robiul Islam, Md. Kamrul Hasan. "On-air English Capital Alphabet (ECA) recognition using depth information", The Visual Computer, 2021 <1 %
Publication
-
- 54 Submitted to University of Lagos <1 %
Student Paper
-
- 55 Submitted to Babes-Bolyai University <1 %
Student Paper
-
- 56 Submitted to Deakin University <1 %
Student Paper
-
- 57 digitalscholarship.unlv.edu <1 %
Internet Source
-
- 58 www.nature.com <1 %
Internet Source
-
- 59 Submitted to KDU College Sdn Bhd <1 %
Student Paper

60	Submitted to Katholieke Universiteit Leuven Student Paper	<1 %
61	Submitted to Kingston University Student Paper	<1 %
62	Submitted to universitetechnologimara Student Paper	<1 %
63	Submitted to American International University - Bangladesh Student Paper	<1 %
64	www.analyticsvidhya.com Internet Source	<1 %
65	Submitted to University of Kent at Canterbury Student Paper	<1 %
66	dspace.bracu.ac.bd:8080 Internet Source	<1 %
67	oparu.uni-ulm.de Internet Source	<1 %
68	Muhammad Sheikh Sadi, Mohammed Alotaibi, Prottoy Saha, Fahamida Yeasmin Nishat et al. "COV-CTX: A Deep Learning Approach to Detect COVID-19 from Lung CT and X-Ray Images", International Journal of Online and Biomedical Engineering (ijOE), 2023 Publication	<1 %

69	Submitted to Technological University Dublin Student Paper	<1 %
70	Submitted to University of East London Student Paper	<1 %
71	corescholar.libraries.wright.edu Internet Source	<1 %
72	www.theseus.fi Internet Source	<1 %
73	Chen, Mingyu, Ghassan AlRegib, and Biing-Hwang Juang. "Air-Writing Recognition—Part II: Detection and Recognition of Writing Activity in Continuous Stream of Motion Data", <i>IEEE Transactions on Human-Machine Systems</i> , 2016. Publication	<1 %
74	Submitted to Institut Teknologi Kalimantan Student Paper	<1 %
75	elibrary.buse.ac.zw:8080 Internet Source	<1 %
76	5dok.org Internet Source	<1 %
77	Submitted to Berlin School of Business and Innovation Student Paper	<1 %

- 78 Chaur-Heh Hsieh, You-Shen Lo, Jen-Yang Chen, Sheng-Kai Tang. "Air-writing Recognition Based on Deep Convolutional Neural Networks", IEEE Access, 2021
Publication
-
- 79 Submitted to Kazakh-British Technical University <1 %
Student Paper
-
- 80 Submitted to AUT University <1 %
Student Paper
-
- 81 Calimanut-Ionut Cira, Alberto Díaz-Álvarez, Francisco Serradilla, Miguel-Ángel Manso-Callejo. "Convolutional Neural Networks Adapted for Regression Tasks: Predicting the Orientation of Straight Arrows on Marked Road Pavement Using Deep Learning and Rectified Orthophotography", Electronics, 2023
Publication
-
- 82 Submitted to Columbia University <1 %
Student Paper
-
- 83 Submitted to Nanyang Polytechnic <1 %
Student Paper
-
- 84 Submitted to Oakridge International School, Bengaluru <1 %
Student Paper
-

85	Submitted to The Robert Gordon University Student Paper	<1 %
86	eprints.iain-surakarta.ac.id Internet Source	<1 %
87	vdoc.pub Internet Source	<1 %
88	Submitted to Student Paper	<1 %
89	Gypsy Nandi. "Principles of Soft Computing Using Python Programming", Wiley, 2023 Publication	<1 %
90	Submitted to Loughborough University Student Paper	<1 %
91	Submitted to Pontificia Universidad Catolica del Ecuador - PUCE Student Paper	<1 %
92	Submitted to UC, Boulder Student Paper	<1 %
93	Submitted to University of Lincoln Student Paper	<1 %
94	Submitted to University of Suffolk Student Paper	<1 %
95	Submitted to University of Wollongong Student Paper	<1 %

- 96 Wahyu Maulana Hassanudin, Victor Gayuh Utomo, Riski Apriyanto. "Fine-Grained Analysis of Coral Instance Segmentation using YOLOv8 Models", sinkron, 2024
Publication
-
- 97 bbs.huaweicloud.com <1 %
Internet Source
-
- 98 iieta.org <1 %
Internet Source
-
- 99 nagoya.repo.nii.ac.jp <1 %
Internet Source
-
- 100 Md. Shahinur Alam, Ki-Chul Kwon, Nam Kim. "Trajectory-Based Air-Writing Character Recognition Using Convolutional Neural Network", 2019 4th International Conference on Control, Robotics and Cybernetics (CRC), 2019
Publication
-
- 101 Submitted to University of Nottingham <1 %
Student Paper
-
- 102 khuhub.khu.ac.kr <1 %
Internet Source
-
- 103 pureadmin.qub.ac.uk <1 %
Internet Source
-
- 104 tech.hindustantimes.com <1 %
Internet Source

- 105 www.oknews.in <1 %
Internet Source
-
- 106 Submitted to University of Huddersfield <1 %
Student Paper
-
- 107 deepai.org <1 %
Internet Source
-
- 108 infai.org <1 %
Internet Source
-
- 109 ir.library.dc-uoit.ca <1 %
Internet Source
-
- 110 noexperiencenecessarybook.com <1 %
Internet Source
-
- 111 www.research-collection.ethz.ch <1 %
Internet Source
-
- 112 Dostdar Hussain, Muhammad Ismail, Israr
Hussain, Roobaea Alroobaea, Saddam
Hussain, Syed Sajid Ullah. "Face Mask
Detection Using Deep Convolutional Neural
Network and MobileNetV2-Based Transfer
Learning", Wireless Communications and
Mobile Computing, 2022
Publication
-
- 113 Mubarak Taiwo Mustapha, Ilker Ozsahin,
Dilber Uzun Ozsahin. "Convolution neural
network and deep learning", Elsevier BV, 2024
Publication

- 114 Nurullainy Mat Rashid, Liyana Shuib, Christopher Ifeanyi Eke, Nor A. Azmi. "Detecting Covid-19 in Chest X-Ray Images With Convolutional Neural Network", 2021 International Conference on Computer Science and Engineering (IC2SE), 2021
Publication <1 %
- 115 Prameeladevi Chillakuru, K. Ananthajothi, D. Divya. "Three stage classification framework with ranking scheme for distracted driver detection using heuristic-assisted strategy", Knowledge-Based Systems, 2024
Publication <1 %
- 116 dr.ntu.edu.sg <1 %
Internet Source
- 117 dspace.iiuc.ac.bd <1 %
Internet Source
- 118 ebin.pub <1 %
Internet Source
- 119 ijisae.org <1 %
Internet Source
- 120 lib.buet.ac.bd:8080 <1 %
Internet Source
- 121 peerj.com <1 %
Internet Source

- 122 "Advanced Computer Architecture", Springer Science and Business Media LLC, 2018 **<1 %**
Publication
-
- 123 "Pattern Recognition and Machine Intelligence", Springer Science and Business Media LLC, 2019 **<1 %**
Publication
-
- 124 Fu Zhangjie, Jiashuang Xu, Zhuangdi Zhu, Alex X. Liu, Xingming Sun. "Writing in the Air with WiFi Signals for Virtual Reality Devices", IEEE Transactions on Mobile Computing, 2018 **<1 %**
Publication
-
- 125 Hakan Kayan, Yasar Majib, Wael Alsafery, Mahmoud Barhamgi, Charith Perera. "AnoML-IoT: An end to end re-configurable multi-protocol anomaly detection pipeline for Internet of Things", Internet of Things, 2021 **<1 %**
Publication
-
- 126 Purnendu Mishra, Kishor Sarawadekar. "Fingertips Detection with Nearest-Neighbor Pose Particles from a Single RGB Image", IEEE Transactions on Circuits and Systems for Video Technology, 2021 **<1 %**
Publication
-
- 127 Sheikh Shamshad Ahmed, Afrin Akter Mim, Dipto Saha, Kamrun Nahar, Md. Aynul Hasan Nahid. "A CNN-Based Novel Approach for The **<1 %**

Detection of Compound Bangla Handwritten Characters", 2023 11th International Symposium on Digital Forensics and Security (ISDFS), 2023

Publication

-
- 128 **Shriyank Somvanshi, Gian Antariksa, Subasish Das. "Enhanced Balanced-Generative Adversarial Networks to Predict Pedestrian Injury Types", Research Square Platform LLC, 2024** <1 %
- Publication
-
- 129 Submitted to University of Central Lancashire <1 %
- Student Paper
-
- 130 **dspace.cvut.cz** <1 %
- Internet Source
-
- 131 **elib.uni-stuttgart.de** <1 %
- Internet Source
-
- 132 **elibrary.tucl.edu.np** <1 %
- Internet Source
-
- 133 **file.techscience.com** <1 %
- Internet Source
-
- 134 **gitlab.sliit.lk** <1 %
- Internet Source
-
- 135 **iris.uniroma1.it** <1 %
- Internet Source
-
- jurnal.polibatam.ac.id**

136	Internet Source	<1 %
137	library.samdu.uz	<1 %
138	rocs.ici.ro	<1 %
139	scholarworks.utep.edu	<1 %
140	trepo.tuni.fi	<1 %
141	vdocuments.mx	<1 %
142	wininspace.uwinnipeg.ca	<1 %
143	www.frontiersin.org	<1 %
144	www.ijisae.org	<1 %
145	www.w3computing.com	<1 %
146	"A Guide to Applied Machine Learning for Biologists", Springer Science and Business Media LLC, 2023	<1 %

- 147 "Advances in Smart Healthcare Paradigms and Applications", Springer Science and Business Media LLC, 2023 **<1 %**
Publication
-
- 148 "Artificial Intelligence Methods in Intelligent Algorithms", Springer Science and Business Media LLC, 2019 **<1 %**
Publication
-
- 149 "Image and Graphics Technologies and Applications", Springer Science and Business Media LLC, 2019 **<1 %**
Publication
-
- 150 digitalcommons.calpoly.edu **<1 %**
Internet Source
-
- 151 "Advances in Swarm Intelligence", Springer Science and Business Media LLC, 2018 **<1 %**
Publication
-
- 152 Pensiri Akkajit, Arsanchai Sukkuea, Boonnisa Thongnonghin. "Comparative analysis of five convolutional neural networks and transfer learning classification approach for microplastics in wastewater treatment plants", Ecological Informatics, 2023 **<1 %**
Publication
-
- 153 Sridhar Alla, Suman Kalyan Adari. "Beginning MLOps with MLFlow", Springer Science and Business Media LLC, 2021 **<1 %**

154 hdl.handle.net <1 %
Internet Source

Exclude quotes On Exclude matches Off
Exclude bibliography On