

An Efficient Handwritten Bangla Character Recognition System using Computer Vision and Natural Language Processing

by

Tanusree Das Aishi

20101012

Md. Seam Iltimas

20301036

Mirza Azwad Wakil

20101063

Pritam Barua

20101291

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
School of Data and Sciences
BRAC University
September 2023

© 2023. BRAC University
All rights reserved.

Declaration


It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at BRAC University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:



Tanusree Das Aishi
20101012



Md. Seam Iltimas
20301036



Mirza Azwad Wakil
20101063



Pritam Barua
20101291

Approval

The thesis/project titled “An Efficient Handwritten Bangla Character Recognition System using Computer Vision and Natural Language Processing” submitted by

1. Tanusree Das Aishi(20101012)
2. Md. Seam Iltimas(20301036)
3. Mirza Azwad Wakil(20101063)
4. Pritam Barua(20101291)

Of Summer, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on September 2023.

Examining Committee:

Supervisor:
(Member)



Dr. Md. Ashraful Alam
Associate Professor
Department of Computer Science and Engineering
BRAC University

Co-Supervisor:
(Member)



Dr. Farig Yousuf Sadeque
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
BRAC University

Ethics Statement

We hereby declare that this thesis is based on our own findings from our own research discoveries. All other sources used in this work have been properly acknowledged. Furthermore, we confirm that this thesis has not been submitted nor presented, either in its entirety or partially for the purpose of receiving a degree from any other educational institution or university.

Abstract

An efficient handwritten character recognition system for an alpha-syllabary language like Bangla has always been a challenging issue. Despite being in demand, the number of papers being conducted on this was very infrequent. Alongside computer vision, our paper proposes the idea of using grapheme segmentation approach to create an effective system for handwritten Bangla character recognition. The system profoundly deals with the image of handwritten Bangla characters to pre-process through Computer Vision. To achieve the efficiency, we have segregated each Bangla word into grapheme roots and diacritics, whether its simple or compound character. Through these segments, we compare the roots and diacritics individually with the given dataset to recognise the characters. Thus, this system is capable of coping up with the limitations that previous models have by recognising any handwritten Bangla characters efficiently with great accuracy of 0.98357, 0.98208 and 0.94325 for vowel diacritics, consonant diacritics and grapheme root respectively. For proper reconstructed grapheme representation as output, we have approached a reconstruction method with grapheme segmentation. Thus, the implementation of efficient handwritten character recognition was achieved by computer vision and grapheme approach from NLP.

Keywords: Character Recognition, Computer Vision, Graphemes, NLP

Dedication

We want to dedicate all of our sacrifices and educational efforts to our great parents, without whom we would be worthless. We also dedicate our thesis to Md. Ashraful Alam sir, who served as our supervisor and Farig Yousuf Sadeque sir, who served as our co-supervisor, guided us, and showed us how to develop our skills and personalities as successful professionals.

Acknowledgement

We extend our gratitude to the Great Almighty for guiding us through the completion of our thesis without any major setbacks. Our appreciation also goes to our supervisor, Md. Ashrafal Alam and to our co-supervisor Farig Yousuf Sadeque, for their invaluable support and guidance throughout the process. Lastly, we are grateful for the unwavering support of our parents, without whom this achievement would not have been possible.

Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iv
Abstract	v
Dedication	vi
Acknowledgment	vii
Table of Contents	viii
List of Figures	x
List of Tables	xii
Nomenclature	xiii
1 Introduction	1
1.1 Handwritten Bangla Character Recognition	1
1.2 Research Problem	3
1.3 Research Objective	4
2 Literature Review	5
3 Description of the dataset	8
3.1 Bangla Handwritten Grapheme Dataset	8
3.2 Data Augmentation	14
4 Description of the models	16
4.1 Overview of our proposed system:	16
4.2 Support Vector Classifier (SVC)	16
4.3 Custom CNN Model	18
4.4 ResNet-50	20
4.5 VGG19	23

5	Reconstruction	26
5.1	Reconstruction of Graphemes	26
5.2	Challenges of Reconstruction and proposed solutions	27
5.3	Features	29
6	Results and analysis	30
6.1	Evaluation Metrics	30
6.2	Results and Comparison of different models	32
6.2.1	SVC	32
6.2.2	Custom CNN model	33
6.2.3	ResNet-50	38
6.2.4	VGG19	43
6.3	Comparison	48
7	Conclusion	50
	Bibliography	52

List of Figures

1.1	Bangla grapheme	1
3.1	Image values	8
3.2	Grapheme Count	9
3.3	Labels	9
3.4	Considered roots and diacritics	10
3.5	Grayscale grapheme image example 1	10
3.6	Grayscale grapheme image example 2	10
3.7	Grapheme Frequency	11
3.8	SVC x	11
3.9	SVC y	12
3.10	Automated Batch Learning	12
3.11	Examples of Left Shift, Right Shift and Zoom	14
3.12	Examples of Horizontal Flip, Vertical Flip and Rotation	15
4.1	Overall structure of the proposed system	16
4.2	Support Vector Classifier (SVC)	17
4.3	Support Vector Classifier (SVC) model diagram	17
4.4	A basic CNN model architecture	18
4.5	Custom CNN model	19
4.6	Building block of residual learning	20
4.7	ResNet-50 Bottleneck building block	21
4.8	ResNet-50 architecture	22
4.9	VGG19 Model	24
5.1	Consonant diacritics in middle	27
5.2	Consonant diacritics in end	27
5.3	Solution for the representation problem of ‘বৈফ’	28
5.4	Out of the dictionary grapheme prediction	29
6.1	CNN vowel diacritics accuracy	33
6.2	CNN vowel diacritics loss	33
6.3	CNN Vowel Diacritics Confusion Matrix	34
6.4	CNN Consonant Diacritics accuracy	35
6.5	CNN Consonant Diacritics Loss	35
6.6	CNN Consonant Diacritics Confusion Matrix	36
6.7	CNN Grapheme Root Accuracy	37
6.8	CNN Grapheme Root Loss	37
6.9	ResNet-50 Vowel Diacritics Accuracy	38

6.10	ResNet-50 Vowel Diacritics Loss	38
6.11	ResNet-50 Vowel Diacritics Confusion Matrix	39
6.12	ResNet-50 Consonant Diacritics Accuracy	40
6.13	ResNet-50 Consonant Diacritics Loss	40
6.14	ResNet-50 Consonant Diacritics Confusion Matrix	41
6.15	ResNet-50 Grapheme Root Accuracy	42
6.16	ResNet-50 Grapheme Root Loss	42
6.17	VGG19 Vowel Diacritics Accuracy	43
6.18	VGG19 Vowel Diacritics Loss	43
6.19	VGG19 Vowel Diacritics Confusion Matrix	44
6.20	VGG19 Consonant Diacritics Accuracy	45
6.21	VGG19 Consonant Diacritics Loss	45
6.22	VGG19 Consonant Diacritics Confusion Matrix	46
6.23	VGG19 Grapheme Root Accuracy	47
6.24	VGG19 Grapheme Root Loss	47
6.25	Comparison of all the models	49

List of Tables

6.1	SVC Results	32
6.2	Accuracy comparison of all the models	48
6.3	Precision comparison of all the models	48
6.4	Recall comparison of all the models	48

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

F1 Balanced F-Score

ML Machine Learning

NLP Natural Language Processing

ResNet Residual Neural Network

SVM Support Vector Machine

Chapter 1

Introduction

1.1 Handwritten Bangla Character Recognition

Bangla, the fifth-most-spoken Indo-European language, is the official and native language of Bangladesh and is spoken by around 220 million people globally, primarily in the Indian subcontinent (West Bengal, Assam, Tripura). It is a very historically enriched language and is an essential communication medium in the aforementioned regions. The Bangla alphabet consists of 39 consonants and 11 vowels. Being a language from the alphasyllabary family which is also known as abugida, the writing system of Bangla is segmental, meaning that each word is made up of consonant-vowel units and these units are referred as Graphemes. In alphasyllabary languages, graphemes are the smallest units of writing [1]. The fundamental components of Bangla graphemes are grapheme roots, vowel diacritics, and consonant diacritics. Three types of grapheme roots can be distinguished: vowels, consonants, or conjuncts of characters. Vowel or consonant diacritics may be used with grapheme roots, or they may be used alone to create a single grapheme.

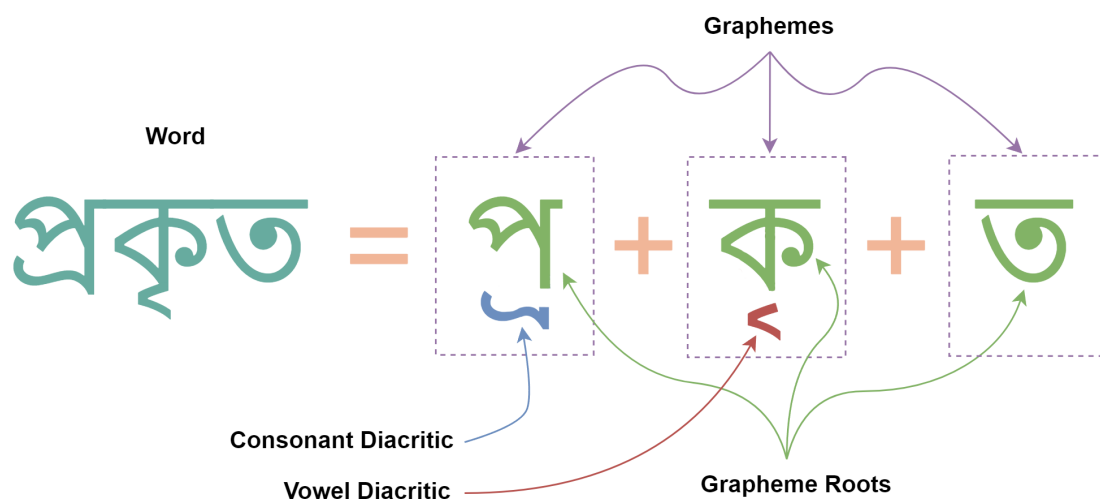


Figure 1.1: Bangla grapheme

Character recognition systems are increasingly in demand for governmental and academic uses in recent times and have been growing in popularity. There have been many studies done for handwritten English character recognition but unfortunately very few researches have been carried out for the recognition of Bangla handwritten characters. Moreover, it is easier to recognize English characters because of their linear arrangement of letters which is not the case for Bangla characters. Unlike English characters, Bangla characters are difficult to recognize due to the complex writing system and non-linear arrangement of the characters. Whereas most OCR(Optical Character Recognition) systems perform a linear pass over a written line, we need to take into account the idea of non-linear placement while developing OCR systems for alphasyllabary languages like Bangla [1]. Another challenge as mentioned by M.Z. Alom et al. [2] in their paper, handwriting of each language typically varies in size and shape from person to person. Additionally, the characters might either be isolated or written in cursive, which makes it more difficult to recognize them.

Our goal in this thesis is to address the aforementioned issues with handwritten Bangla character recognition, with a focus on conjunct grapheme recognition. We aim to construct a system with the aid of computer vision and natural language processing by researching existing methodologies and datasets. We intend to develop a feasible and more efficient method than the existing approaches for reading Bangla handwriting that will be useful for both academic, governmental and commercial purposes.

1.2 Research Problem

The progress of handwritten character recognition in English and a few other languages have come a long way. Many models have been developed through which handwritten characters can be recognized with almost 100 percent accuracy especially in English. According to [3], MNIST dataset is able to recognize English handwritten characters with 99.79% accuracy which is almost as close as a human can recognize.

However, in the case of Bangla, things get a little complicated. Firstly, the letters are complex in shape and some have similarities with each other. There are even few characters that differ from one another only by a single dot. Secondly, there is no perfect Bangla dataset. Even though there are some, those are not enough. These datasets can either identify a higher number of classes or provide high accuracy but not both at the same time. The system developed by K. Roy et al. obtained 98.42% accuracy on 10 numeric data classes and 91.13% accuracy on 50 classes of character data [4]. Next, there is also another model which is mainly machine learning based through which satisfying accuracy can be achieved [4]. Yet, the accuracy becomes questionable when it comes to identifying a new character compared to a previously faced character. Moreover, there are many other models such as CNN (Convolutional Neural Network), Autoencoder, Extended CNN, Data Augmentation, Bi-directional LSTM (Long Short-Term Memory) etc. However, most of these models fail to provide high accuracy for large numbers of classes. Now this raises a question,

Is it possible to develop such a system where huge numbers of data, especially compound characters are recognized with high accuracy?

We believe that it is possible to do so using the latest techniques of Natural Language Processing and Computer Vision combined.

In this research, NLP and Computer vision based Bangla Handwritten character recognition is investigated. The proposed system first divides the given word into separate characters and applies computer vision to identify whether the character is simple or compound. If it is a simple character then the corresponding unicode is generated. On the other hand, if it is a compound character then NLP comes into action in order to divide the compound character into separate segments based on graphemes. Finally, the corresponding unicodes of the separate segments are combined in order to convert it to the desired text regardless of the combination of characters used in the compound character. We are hopeful that this method will be able to show satisfactory results and outperform other existing methods.

1.3 Research Objective

Considering the previously used models for the purpose of handwritten character recognition in Bangla language, our paper upholds the purpose of a more efficient system. Alongside using the adequate models, usage of grapheme segmentation via NLP and computer vision for preprocessing the simple singular character, will be the part of our preferred research.

In the field of Bangla character recognition, working with neural models we often face the hardships in finding the proper dataset in correct and desired format, which results in fluctuating accuracy. In the preprocessing stage of our model, Computer Vision extracts the characters(Simple and Compound) from any word and the simple characters are converted to unicodes according to their class values. Then, for the compound character NLP (Natural Language Processing) will be of great use through graphemes.

Therefore Objectives:

1. Synchronizing the unicodes from classifier in preprocessing via Computer vision.
2. For compound characters to be recognised, efficient usage of NLP, where Graphemes(Segmentation of the words linguistically) will play a significant role.
3. Attempt to gather a proper dataset for peak accuracy.

Chapter 2

Literature Review

To establish an efficient system for Bangla handwritten character recognition is very demanding despite the number of researches being infrequent. Some recent research is based on DNN, CNN, Computer Vision and Machine Learning. But those papers do not take NLP into consideration, specifically for compound Bangla handwritten graphemes. Also, a Grapheme based approach for detecting the conjunct characters is not so usual as the number of researches is very few.

Due to the variability of individual handwriting style together with the isolated or cursive style of the characters make it more complex to recognize them [2]. This complexity enhances more in terms of Bangla handwritten characters, especially the compound characters. An approach to create a model of Bengali handwritten digits recognition has been taken through a Computer Vision challenge in 2018 for the purpose of mass public use [5]. In this challenge, the participants have submitted different models for digit recognition while using traditional machine learning techniques. However, their models only focused on Bengali digits and not on the letters.

To work on the characters, the authors of this paper [1] have proposed an identifying scheme dependent on linguistic fragments of words called Graphemes. The Grapheme segments consist of Grapheme roots (Vowel, Consonant and Conjunct roots) as well as Diacritics. They have used the Bengali ASR dataset [6] for graphemes selection, a large dataset based on the scheme. Also, the contrasting orthography of alpha-syllabary languages like Bangla compared to English had made it more challenging for them to create the model. In our model, we have also used Graphemes to detect simple and compound characters through CV and NLP with larger dataset. In order to perform that, the idea of reconstruction has been introduced.

Another paper [7] introduced a DNN based automated system for Bangla character recognition where they have used DNNs, for example, VGG16, ResNet50 and DenseNet121 to achieve a great accuracy of more than 90%. Their dataset is BanglaLekha-isolated dataset [8] for the research. They have also worked for visually impaired people by converting those recognised characters into Unicode and

eventually interpreting it though a Braille representation system. However, in our paper, we only focus on the implementation of detecting the handwritten characters. Also, our paper gives importance on NLP for compound characters to eventually recognise Bangla words which this paper [7] fails to achieve.

Another approach made by M.M. Rahman et al. [9] using CNN(Convolutional Neural Network) was able to generate better accuracy than DNN based model in character recognition. For this, they have gone through 20,000 handwritten samples and achieved 93.93% accuracy eventually. However, their work is only limited to basic Bangla characters. To deal with the compound Bangla handwritten characters recognition, U. Pal et al. [10] proposed the idea of using Gradient Feature by which they could obtain an accuracy of 85.90% from 20,543 samples of these conjunct characters. Nonetheless, our intent is to produce higher accuracy for this task with the help of NLP which can assist computers to read texts, recognise and interpret it.

Again, T.K. Bhowmik et al. [11] introduced another model where they introduced an MLP classifier to recognise 50 Bangla basic characters from 440 sample images based on stroke features. In this process, they tried to extract 10 distinct features from the character images by identifying the strokes present in them. On the contrary, another model [12] of SVM (Support Vector Machine) based two-stage Hierarchical Architecture has been proved to outperform the MLP and RBF network classifiers in recognising the characters. In addition, the researchers proved the fusion scheme of the three classifiers to be slightly better than the SVM classifier.

Apart from the above models, S.K. Parui et al. [13] presented an alternate approach where they worked with 24,500 handwritten isolated sample characters and grouped the different strokes of the graphemes into 54 classes based on their similarity in shape. And they used HMM (Hidden Markov Models) for the recognition of these strokes. Finally, they used a second phase classification for character recognition through the classification of strokes. Nonetheless, their model only concentrates on the basic Bangla handwritten graphemes despite having a large dataset.

Analyzing the above discussed papers, it can be concluded that an efficient recognition system for Bangla handwritten graphemes is still a requirement as the amount of research on the topic is very few. Moreover, most of the work is based on simple Bangla characters recognition. There exists some work on recognition of compound characters, yet their accuracy is not up to the expectation in every case. Therefore, the significance of compound or conjunct characters recognition cannot be denied also. In our proposed model, we are not only using Computer Vision for basic characters, but also implementing NLP(Natural Language Processing) for Compound characters through the concept of Grapheme segmentation.

Lastly, our paper deals with the recognition system with a unique approach. In this approach, each image of Bangla characters (Simple and Compound) is represented by a unique numerical value. Then the model is trained with the corresponding value of the image where the value is reconstructed to that specific image character. Thereafter, the trained model can predict the characters while testing an image by generating the corresponding numeric value as well as converting it to that character in text format. But what makes it unique is that the model can even predict a conjunct character which may have not been used in training. However, the diacritics and the roots of that conjunct character may have been used in separate different images. Basically, the reconstruction process is used in such a way that it does not merely search for an expected character as it is. Rather, it tries to recognise a character by its root and diacritics, matching them with different images (roots and diacritics of different compound characters). In this way, the proposed model is distinctive among all other models. As a result, our model is able to detect all kinds of Bangla characters (Basic and Conjunct) with brilliant accuracy.

Chapter 3

Description of the dataset

3.1 Bangla Handwritten Grapheme Dataset

The task of Bangla handwritten character recognition requires a huge amount of data. There are lots of datasets available for Bangla handwritten characters that are nearly similar. But as we are considering an approach based on graphemes, we were able to acquire a multi-target dataset for common Bengali graphemes that aligns with our requirements. Therefore, we chose to work with this pre-existing grapheme dataset [1], which is the very first bangla handwritten characters dataset with a grapheme-based labeling scheme.

Our data set is divided into two parts. One consists of the pixel values of 137×236 sized images with image Id and the other one contains the labels along with the image Id. Each image is Flattened to 1-dimensional 32,332 pixel values and saved as a row. Again the image Id section is divided into four different parquet files.

	image_id	0	1	2	3	4	5	6	7	8	...	32322	32323	32324	32325	32326	32327	32328	32329	32330	32331
0	Train_0	254	253	252	253	251	252	253	251	251	...	253	253	253	253	253	253	253	253	253	251
1	Train_1	251	244	238	245	248	246	246	247	251	...	255	255	255	255	255	255	255	255	255	254
2	Train_2	251	250	249	250	249	245	247	252	252	...	254	253	252	252	253	253	253	253	251	249
3	Train_3	247	247	249	253	253	252	251	251	250	...	254	254	254	254	254	253	253	252	251	252
4	Train_4	249	248	246	246	248	244	242	242	229	...	255	255	255	255	255	255	255	255	255	255

Figure 3.1: Image values

It is really difficult to manage four separate files simultaneously. So, while working, we had to merge them all into one table for getting proper values and we used the pandas library for this purpose. After combining, we can observe that our data consists of 1295 unique graphemes along with 200,840 images which means there are 200,840 rows in total.

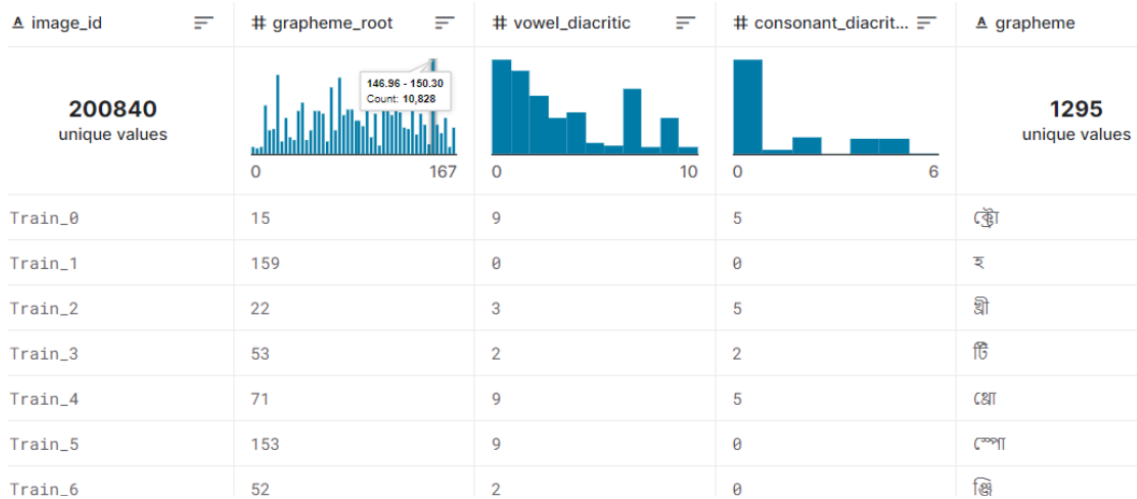


Figure 3.2: Grapheme Count

Similarly, the label file consists of 200,840 rows. However, it consists of 5 columns representing image_id, grapheme_root, vowel_diacritic, consonant_diacritic and unicode_Bangla_character_representation respectively. Grapheme root, vowel diacritic and consonant diacritic are already encoded and each are assigned with values from 0 to n (total number).

	image_id	grapheme_root	vowel_diacritic	consonant_diacritic	grapheme
0	Train_0	15	9	5	কৌ
1	Train_1	159	0	0	হ
2	Train_2	22	3	5	খী
3	Train_3	53	2	2	টি
4	Train_4	71	9	5	গো
...
200835	Train_200835	22	7	2	র্খে
200836	Train_200836	65	9	0	ন্তো
200837	Train_200837	2	1	4	অ্যা
200838	Train_200838	152	9	0	নো
200839	Train_200839	127	2	0	ল্টি

200840 rows × 5 columns

Figure 3.3: Labels

Bangla language has a large number of graphemes (approximately 341,782,677). Considering all of the possible combinations of compound characters, the number is nearly infinite. Consequently, it is impractical to deal with such a vast quantity of data. Therefore, in our case, we have restricted it to 1295 commonly employed graphemes. We have 168 grapheme roots among which 11 are vowels, 38 are consonants and 119 are conjuncts. Also, we have 11 vowel diacritics as well as 8 consonant diacritics (including null).

Target Variable	Class
Grapheme roots (168)	<u>VOWEL ROOTS</u> অ (a), আ (ā), ই (i), ঐ (ī), উ (u), ঊ (ū), ঋ (r), এ (ē), ঐ (ai), ও (ō), ঔ (au)
	<u>CONSONANT ROOTS</u> ক (ka), খ (kha), গ (ga), ঘ (gha), ঙ (ña), চ (ca), ছ (cha), জ (ja), ঝ (jha), ঞ (ña), ট (ta), ঠ (tha), ড (da), ঢ (dha), ণ (ña), ত (ta), থ (tha), দ (da), ধ (dha), ন (na), প (pa), ফ (pha), ব (ba), ভ (bha), ম (ma), য (ya), র (ra), ল (la), শ (śa), ষ (ṣa), স (sa), হ (ha), ঙ (ra), ঙ (rha), য় (ya), ং (m̐), ঃ (ḥ), ণ (ṭ)
	<u>CONJUNCT ROOTS</u> ক্ক (kka), ক্ট (kṭa), ক্ত (kta), ক্ক (kla), ক্ক (kṣa), ক্ক (kṣṇa), ক্ক (kṣma), ক্ক (ksa), ক্ক (gdha), গ্গ (gna), গ্গ (gba), গ্গ (gma), গ্গ (gla), গ্গ (ghna), ক্ক (ñka), ক্ক (ñkta), ক্ক (ñkṣa), ক্ক (ñkha), ক্ক (ñga), ক্ক (ñgha), ক্ক (cca), ক্ক (ccha), ক্ক (cchba), ক্ক (jja), ক্ক (jjba), ক্ক (jña), ক্ক (jba), ক্ক (ñca), ক্ক (ñcha), ক্ক (ñja), ক্ক (ṭṭa), ক্ক (ḍḍa), ক্ক (ṇṭa), ক্ক (ṇṭha), ক্ক (ṇḍa), ক্ক (ṇṇa), ক্ক (tta), ক্ক (ttba), ক্ক (t'tha), ক্ক (tna), ক্ক (tba), ক্ক (tma), ক্ক (dgha), ক্ক (dda), ক্ক (d'dha), ক্ক (dba), ক্ক (dbha), ক্ক (dma), ক্ক (dhba), ক্ক (nja), ক্ক (ṇṭa), ক্ক (ṇṭha), ক্ক (ṇḍa), ক্ক (nta), ক্ক (ntba), ক্ক (ntha), ক্ক (nda), ক্ক (ndba), ক্ক (ndha), ক্ক (mna), ক্ক (mba), ক্ক (nma), ক্ক (nsa), ক্ক (ṭṭa), ক্ক (pta), ক্ক (pna), ক্ক (ppa), ক্ক (pla), ক্ক (psa), ক্ক (phṭa), ক্ক (phpha), ক্ক (phla), ক্ক (bja), ক্ক (bda), ক্ক (bdha), ক্ক (bba), ক্ক (bla), ক্ক (bhla), ক্ক (mna), ক্ক (mpa), ক্ক (mba), ক্ক (mbha), ক্ক (m'ma), ক্ক (mla), ক্ক (lka), ক্ক (lga), ক্ক (lṭa), ক্ক (lḍa), ক্ক (lpa), ক্ক (lba), ক্ক (lma), ক্ক (lla), ক্ক (śca), ক্ক (śna), ক্ক (śba), ক্ক (śma), ক্ক (śla), ক্ক (śka), ক্ক (śṭa), ক্ক (śṭha), ক্ক (śṇa), ক্ক (spa), ক্ক (spha), ক্ক (sma), ক্ক (ska), ক্ক (sṭa), ক্ক (sta), ক্ক (stha), ক্ক (sna), ক্ক (spa), ক্ক (spha), ক্ক (sba), ক্ক (sma), ক্ক (sla), ক্ক (s'sa), ক্ক (hna), ক্ক (hba), ক্ক (hma), ক্ক (hla)
Vowel Diacritics (11) Null, বা (bā), বি (bi), বী (bī), বু (bu), বূ (bū), বে (bē), বৈ (bai), বৌ (bō), বৌ (bau)	
Consonant Diacritics (8) Null, ব্য (bya), ব্র (bra), বঁ (rba), বঁ (rbya), ব্র্য (brya), ব্রঁ (rbra), বঁ (ḥ)	

Figure 3.4: Considered roots and diacritics

As the pixel values of our image are represented in 1D shape, we can not visualize it directly. To solve this, we implemented the numpy library to convert the data to a numpy array from panda. After that, we reshaped it to size 137×236 through numpy.reshape. Following that, the matplotlib library is used for visualizing the images. Few examples of the grapheme grayscale images are shown below.

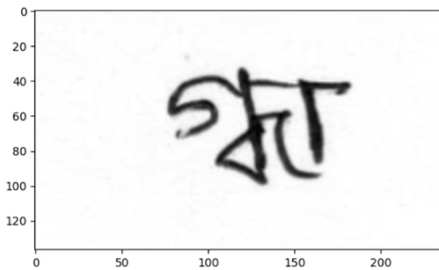


Figure 3.5: Grayscale grapheme image example 1

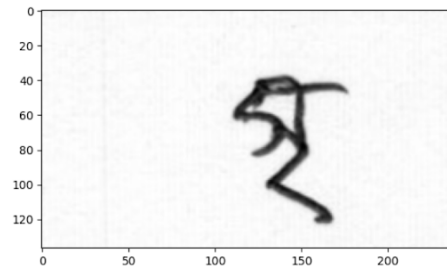


Figure 3.6: Grayscale grapheme image example 2

Subsequently, we iterated over the graphemes to determine the total number of images for each individual grapheme. We noticed that the highest frequency is 178 and the lowest is 118. We also determined the average frequency and it turned out to be around 155 images per grapheme.

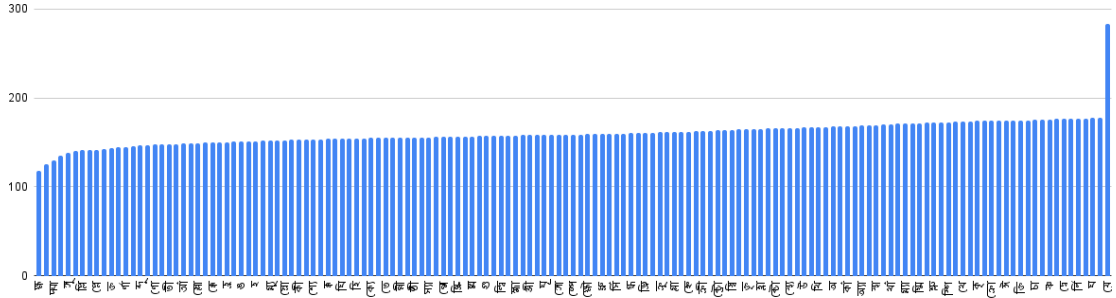


Figure 3.7: Grapheme Frequency

After that , we preprocessed our data to make it suitable for training our models. Firstly, for SVC we dropped the image_id column from image values and added each row to an array called x which is our features. Next, we normalized the x within the range 0 to 1 by dividing the values with 255. For y (target) we dropped the image_id and Bangla character representation column from the label file and appended each row in y.

image_id	0	1	2	3	4	5	6	7	8	...	32322	32323	32324	32325	32326	32327	32328	32329	32330	32331
Train_75	249	252	253	254	254	254	254	253	251	...	254	254	254	254	254	254	254	254	255	255
Train_128	247	251	248	253	252	248	250	252	249	...	253	253	253	253	253	253	254	254	253	253
Train_221	252	253	252	251	252	252	252	252	253	...	254	254	254	253	252	252	252	252	252	252
Train_262	248	248	249	248	248	249	248	246	245	...	255	255	255	255	255	255	255	255	255	255
Train_283	253	252	252	252	252	253	253	253	253	...	252	252	252	252	252	252	252	252	252	252
...
rain_199714	251	244	250	251	254	252	252	248	252	...	254	254	253	253	254	253	254	253	254	252
rain_199865	251	250	251	250	248	248	251	252	252	...	255	255	255	255	255	255	255	255	254	254
rain_200409	239	240	246	250	249	247	247	247	247	...	253	253	252	252	252	252	252	252	252	252
rain_200599	252	250	253	251	249	254	253	253	251	...	254	254	253	252	253	253	253	254	254	255
rain_200767	251	252	253	253	252	252	252	253	253	...	252	252	252	252	252	252	253	253	254	252

Figure 3.8: SVC x

grapheme_root	vowel_diacritic	consonant_diacritic
15	9	5
159	0	0
22	3	5
53	2	2
71	9	5
...
22	7	2
65	9	0
2	1	4
152	9	0
127	2	0

Figure 3.9: SVC y

On the other hand , VGG19 and ResNet-50 takes 224×224 sized images as input. Loading all data at once might cause the dead kernel issue due to the huge amount of data. To avoid these issues, we stored the numpy array as raw image of size 137×236 in a desktop folder using `Image.fromarray.save()` function using a loop for all images and giving names according to the `image_id`. After that we created a loader function with the purpose to load the image. To implement that, we used the `keras.utils.load_img` to load images from directory and convert it to `size=224 \times 224` and `color_mode=rgb`. For custom CNN , we did the similar thing, only changed the size to 200×200 and `color_mode` to grayscale. Then we normalized the `x` by dividing it by 225. After that, we used the `yield` function which plays the main role of loading our data in batches. Finally, incremented the `batch_start` and `batch_end` position. This loader function allowed us to train our model in even low configuration pc which was beneficial for us.

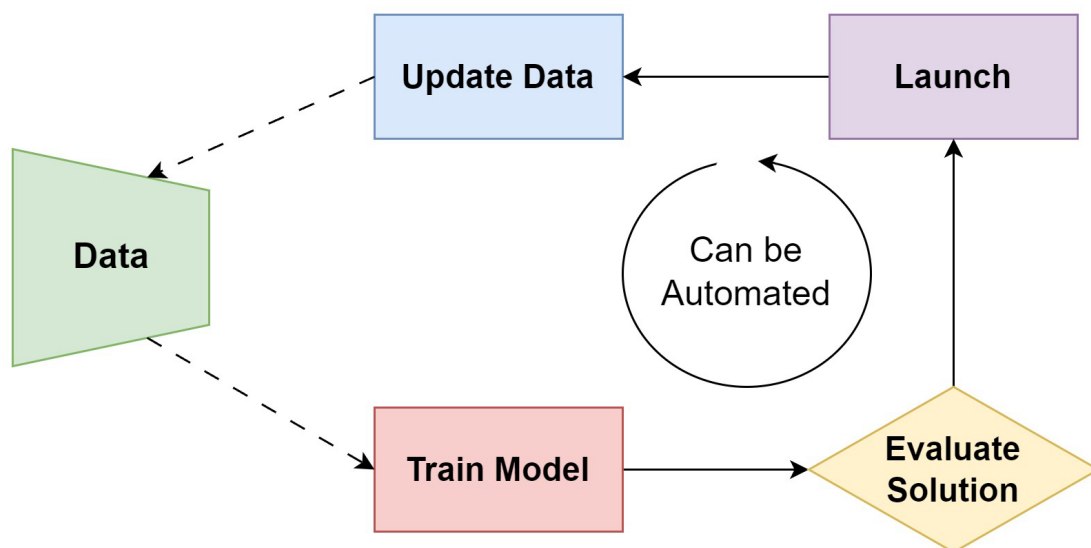
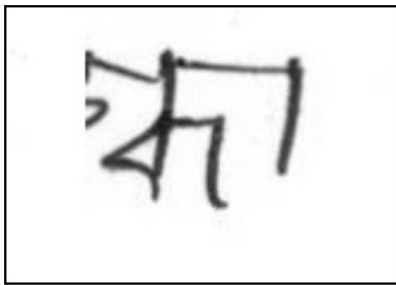


Figure 3.10: Automated Batch Learning

We randomly splitted the x and y in 90:10 ratio for train and test named x_train, y_train, x_test and y_test using train_test_split of sklearn model_selection. Then x_train ,y_train was again splitted into 90:10 ratio for actual train and validation respectively. We plan on using x_train and y_train for training the model, x_val and y_val for validation during training and x_test and y_test for testing its final accuracy. We also ensured that all the models use the same data splits using seed numbers.

3.2 Data Augmentation

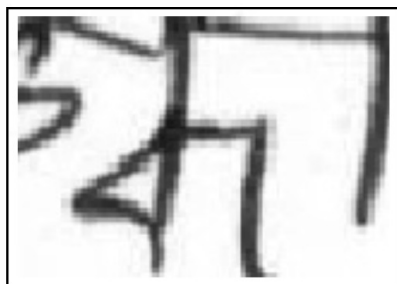
While preparing our dataset, we went for data augmentation. Due to the fact that people have different handwriting styles, data augmentation was crucial for us. It is unfeasible to compile every type of writing style. Data augmentation enables us to transform the original image into different shapes by rotating, adjusting contrast, and so on. Data augmentation additionally expands the size of the training data, which is beneficial for deep learning neural network models as they are able to learn the features better. In addition, it reduces over-fitting during model training by providing more training data to the model ensuring that it can learn from these data and it allows the model to generalize to new data. For data augmentation, we created a function named `data_augment` that uses the `ImageDataGenerator` library. This function was used for data augmentation during the training of the model. We used the `rotation_range=15` and tried to keep other things minimal as too much of other things such as zoom, shift might have resulted in wrong character.



Left Shifting ✗

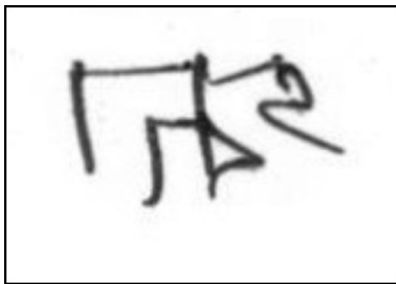


Right Shifting ✗

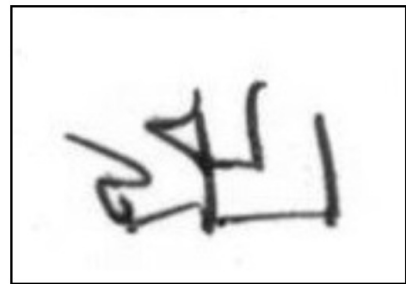


Zoom ✗

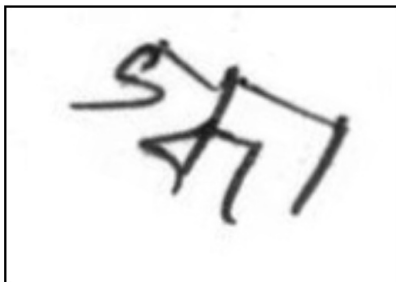
Figure 3.11: Examples of Left Shift, Right Shift and Zoom



Horizontal Flip ✗



Vertical Flip ✗



Rotation ✓



Rotation ✓

Figure 3.12: Examples of Horizontal Flip, Vertical Flip and Rotation

Chapter 4

Description of the models

4.1 Overview of our proposed system:

The figure 4.1 illustrates the overview of our proposed system where an image of handwritten grapheme would be taken as an input and it would be passed through the model which would give three labels as outputs: grapheme root, vowel diacritic and consonant diacritic. The labels would be mapped to get the actual form of the three outputs and then through the reconstruction approach, it would give a proper reconstructed computer usable grapheme as an output.

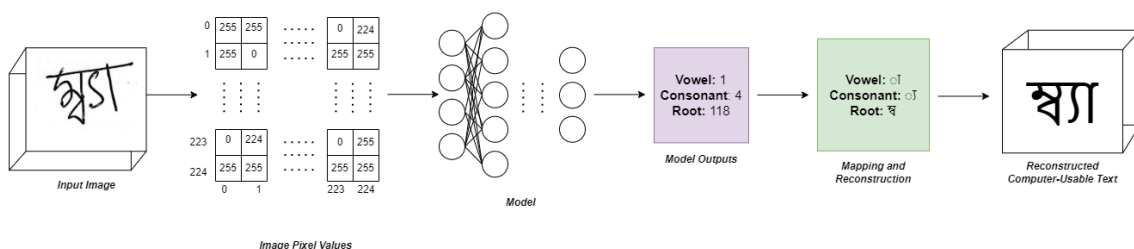


Figure 4.1: Overall structure of the proposed system

4.2 Support Vector Classifier (SVC)

In machine learning, the Support Vector Classifier (SVC), developed based on SVM, is an algorithm to be used for classification problems. In SVC, the data points of the dataset are classified distinctly in an N-dimensional space through Hyperplane (maximum margin to separate the data points). The SVC kernel transforms low dimensional input space into a higher one to linearly separate the data points. With this algorithm, two bordering parallel plates are created to draw the linearly separable categories of data away from each other. The categories that are farthest from the boundary of the plates are the best separator, and the nearest training data to the hyperplane separator plates is called the Support Vector.

In order to implement SVC we used the built in model from sklearn library. Firstly , we imported the SVC from sklearn. Then initialized the model with a linear kernel. Our project outputs three labels at once. As the SVC kernel is linear, it has been

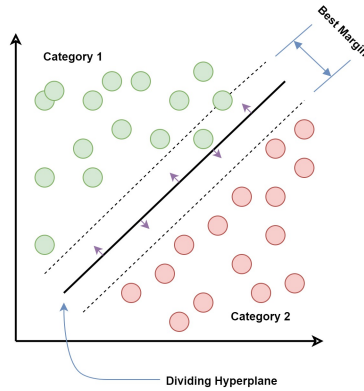


Figure 4.2: Support Vector Classifier (SVC)

combined with ClassifierChain for handling multiple labels in this proposed project. The ClassifierChain will basically arrange the binary Support Vector Classifier into a chain where the models will parallelly make predictions in a specific labels using the available features. This makes our model ready for training . Finally we trained the model with our train data and calculated the accuracy accordingly.

After the preprocessing and formatting of the image data, it is split into Train and Test data where 80% of the data are Train data and the rest of the 20% are Test data. Then the SVC parameters are defined as well as combined with ClassifierChain to train the model using the Train data. To know whether our model is accurate enough to predict, we have used the Test data in our model. The final prediction score signifies how well the model has been trained.

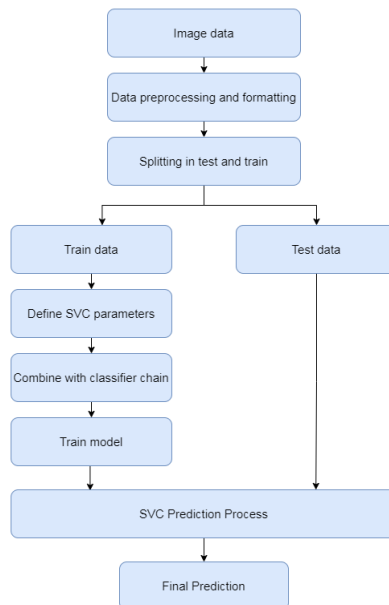


Figure 4.3: Support Vector Classifier (SVC) model diagram

4.3 Custom CNN Model

Deep learning models have developed significant popularity in the field of computer vision and recognition tasks due to their better efficacy when compared to machine learning models. As deep learning models utilize neural networks, they are significantly better at analyzing and capturing complexities in data patterns. Additionally, these models perform far more accurately when trained with large datasets.

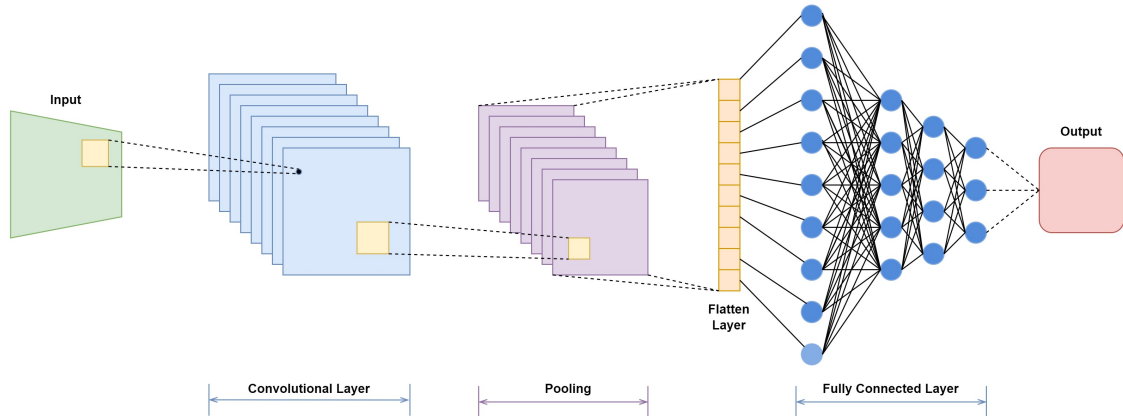


Figure 4.4: A basic CNN model architecture

Convolution Neural Network (CNN) is a neural network model that is extensively used in computer vision and image processing related tasks. A basic CNN model is composed of three layers: convolutional layer, pooling layer and fully connected layers. Convolutional layers are the fundamental building blocks of CNN architecture as they carry out most of the computations. These convolutional layers extract crucial details and features from the input image by performing dot products between two matrices, one of which is the kernel layer, also known as filters and the other being a portion of the input image data. The kernel layer which is essentially a matrix with learnable parameters and smaller than the input data, traverses iteratively throughout the entire image input and generates feature maps as final output. To introduce nonlinearity into the model, activation functions such as ReLU are applied to the produced feature maps. These outputs are then forwarded to the pooling layer, which reduces the spatial dimensions of the feature maps while preserving the most important features. Pooling layer contributes in reducing computational complexity and brings robustness to the model. Feature extraction process is mainly carried out within the convolutional and pooling layers. Following this, the feature maps are flattened to a one dimensional vector and passed through the fully connected layers for the final classification, generating the final output.

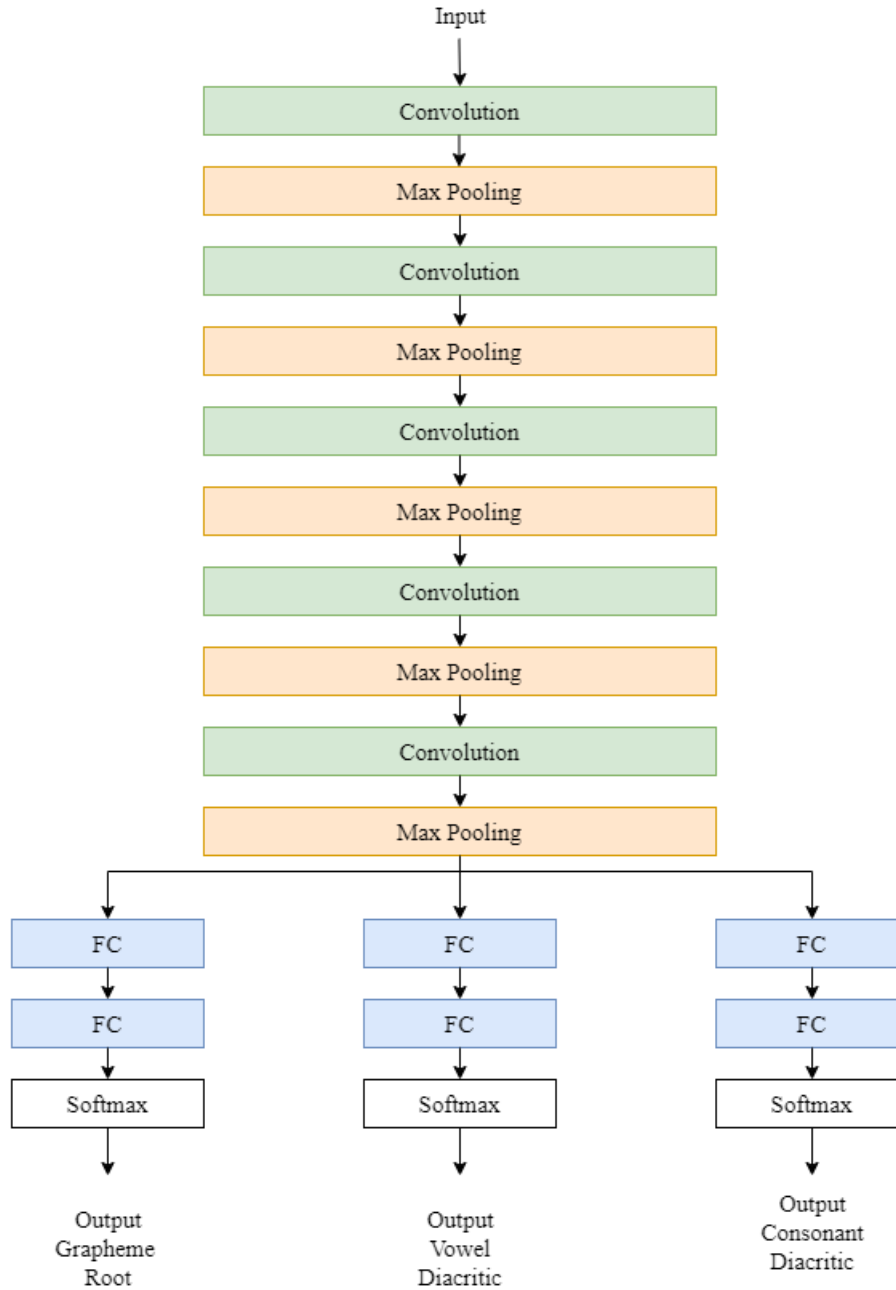


Figure 4.5: Custom CNN model

We have built a custom CNN model with 8 layers in total for our task. It takes an image of size 200*200 as input and the input is then passed through the convolutional layers. We have stacked 5 convolutional layers to enable the model to learn more intricate features and patterns, each of these convolutional layers is followed by a max pooling layer. Afterwards, the feature maps produced by these layers are passed through 2 fully connected layers and 1 softmax layer, finally producing the output. We used ADAM optimizer and sparse categorical cross-entropy for loss function.

4.4 ResNet-50

ResNet-50 is a very deep convolutional neural network architecture that was first introduced in the paper titled 'Deep Residual Learning for Image Recognition' [14], published at the 2016 IEEE Conference on Computer Vision and Pattern Recognition and has since received widespread acclaim and popularity. It is comprised of 50 layers and is based on the concept of residual learning and layering residual blocks to create a potent deeper neural network. This neural network model is a pre-trained model and it was initially pretrained using the ImageNet dataset consisting of millions of images, which helped the model acquire various features, shapes and patterns. The pre-training phase provided the model with a deeper and enriched comprehension of image features, which has been shown to be beneficial for different tasks related to computer vision.

The vanishing gradient problem is an often encountered issue while training a deeper neural network. This has been an issue for multilayered neural networks because as the network gets deeper, the gradient decreases. In the backpropagation technique, gradients are computed by moving from the output layers to the input layers and multiplying the derivatives of the activation functions of each layer. Now the problem arises because most activation functions limit their output value between 0 and 1. When these small values are multiplied, the gradient continues to decrease, which would have an impact on the initial layers because they would be unable to update their weights and biases. Consequently, it would result in an overall drop in the neural network model.

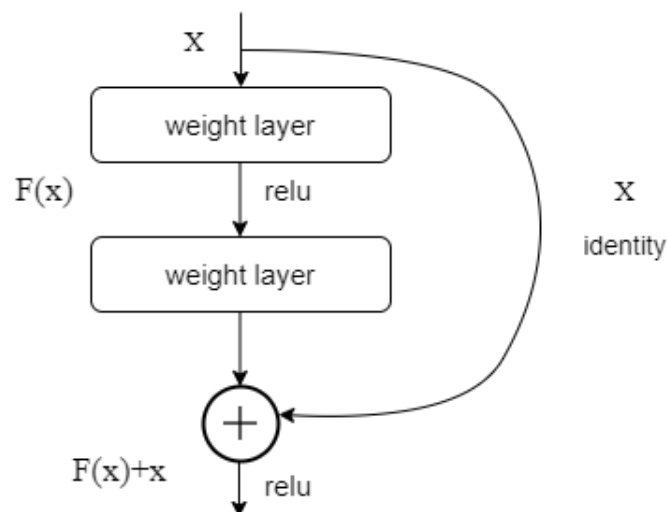


Figure 4.6: Building block of residual learning

The ResNet-50 model addresses this issue by implementing the concept of skip connections and residual blocks. In this approach, the output of one layer serves as the input for the following layers, skipping one or more layers in between. It allows to create a shortcut path for information to flow and to bypass certain intermediary layers, facilitating gradients to find a path to flow without minimizing their value. The skip connections prevent the value from passing through the activation functions that were causing the vanishing gradient in the first place. In other words, if any layer ends up having a reduced value of gradients, it skips that layer bypassing through the skip connections. Therefore, this technique resolves and mitigates the issue of vanishing gradient.

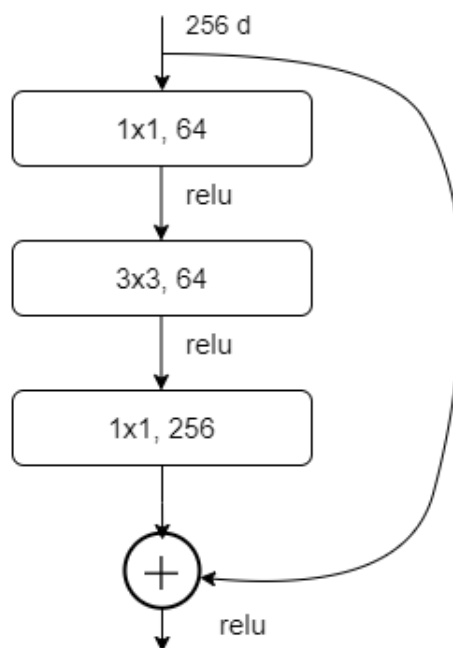


Figure 4.7: ResNet-50 Bottleneck building block

As stated previously, ResNet-50 consists of 50 layers where the network is essentially composed of residual building blocks stacked on top of one another. It takes an RGB image of 224×224 size as input which is then passed through an initial 7×7 convolutional layer. Following that, batch normalization and ReLU activation function is implemented. After that, it passes through a 3×3 kernel sized max pooling layer, with stride size of 2. Following this, it gets passed through multiple ResNet blocks where each block consists of 3 layers : the first layer of 1×1 kernel, the second layer of 3×3 kernel and the last layer of 1×1 kernel. These blocks are also known as Conv block and Identity block . Each Conv block is followed by an Identity block. In the ResNet-50 architecture, the skip connections bypass 3 layers which is illustrated in the bottleneck building block. After getting passed through the last residual block, the output then passes through the average pooling layer. Lastly the output goes through a fully connected layer and softmax function.

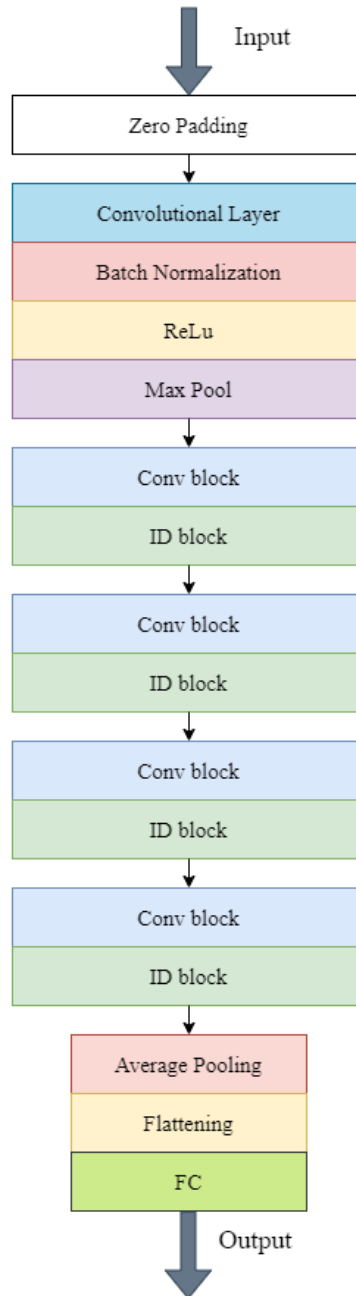


Figure 4.8: ResNet-50 architecture

In order to implement ResNet50 built-in ResNet50 of tensorflow.keras.applications, we set the parameters `include_top = False` , `weights='imagenet'` and set the `input_tensor` and `pooling` to `None`. The input shape was $(224 \times 224 \times 3)$ which is the recommended shape for resnet50. As our program outputs three separate things such as grapheme root, vowel diacritic and consonant diacritic, we had to modify the model and add extra layers. In order to do so we added three new branches where each branch consists of two dense layers with ReLU activation and one dropout layer in between. Finally, we concatenated the output of each branch and added the final classification layer which makes our model ready for training.

4.5 VGG19

VGG19 is an advanced and deep CNN model that was first introduced in the paper ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’ which was published at the ICLR 2015 [15]. This is a pre-trained model that has been trained with millions of diverse images. The performance of VGG19 was appreciated and brought to popularization in the field of computer vision and image classification because of its depth, capability of learning complex features, working efficiently and effortlessly with a wider range of dataset and having simpler architecture compared to other contemporary CNN models.

Three layers: convolutional layers, pooling layers, and fully connected layers — forms a basic CNN model. Image dataset is given as input and the first convolutional layer extracts features from the input images. It produces a feature map as an output, which is then used as input by the additional layers to learn more about the features of the images. In convolutional layers, an activation function is used to introduce nonlinearity into the network for simpler computational tasks for the network, primarily its ReLU (Rectified linear activation unit). Convolutional layers are typically followed by pooling layers after that. The core purpose of the pooling layer is to minimize the feature map size for the ease of computation. Finally it generates the desired output in image format in the output layer where another activation function is used, softmax is used commonly.

VGG19 is a more advanced Convolutional Neural Network architecture known for its 19 layers deep architecture as well as a large number of parameters. The depth of this architecture allows it to recognise more detailed patterns in images and learn more complex features about the input images. These 19 layers are made of 16 Convolutional layers along with 5 MaxPool layers, 2 Fully Connected layers and 1 final layer which is softmax. The input image is given as RGB image, pixels fixed to size 224×224 . This method is used for image classification where there are multiple 3×3 filters used in each convolutional layer. These multiple small 3×3 filters are useful for the network as it can learn more fine grained features for a better result. An activation function is implemented to introduce nonlinearity into the network after each convolutional layer. Rectified linear activation unit (ReLU) is commonly used as the activation function in the VGG19 model. Between the convolutional layers, max pooling layers are also included with the purpose of reducing the size of the feature map and acquiring important smooth and sharp features from the images. After 16 convolutional layers along with 5 MaxPool layers, fully connected layers are added along with 1 final softmax layer, which is connected with the output layer that generates the output.

With the VGG19 network, large datasets can be easily trained which eventually will not hamper the performance of the model. Rather than using a classical CNN, this complex CNN architecture can be of great use in terms of achieving better accurate results and more robustness to the alteration in the dataset.

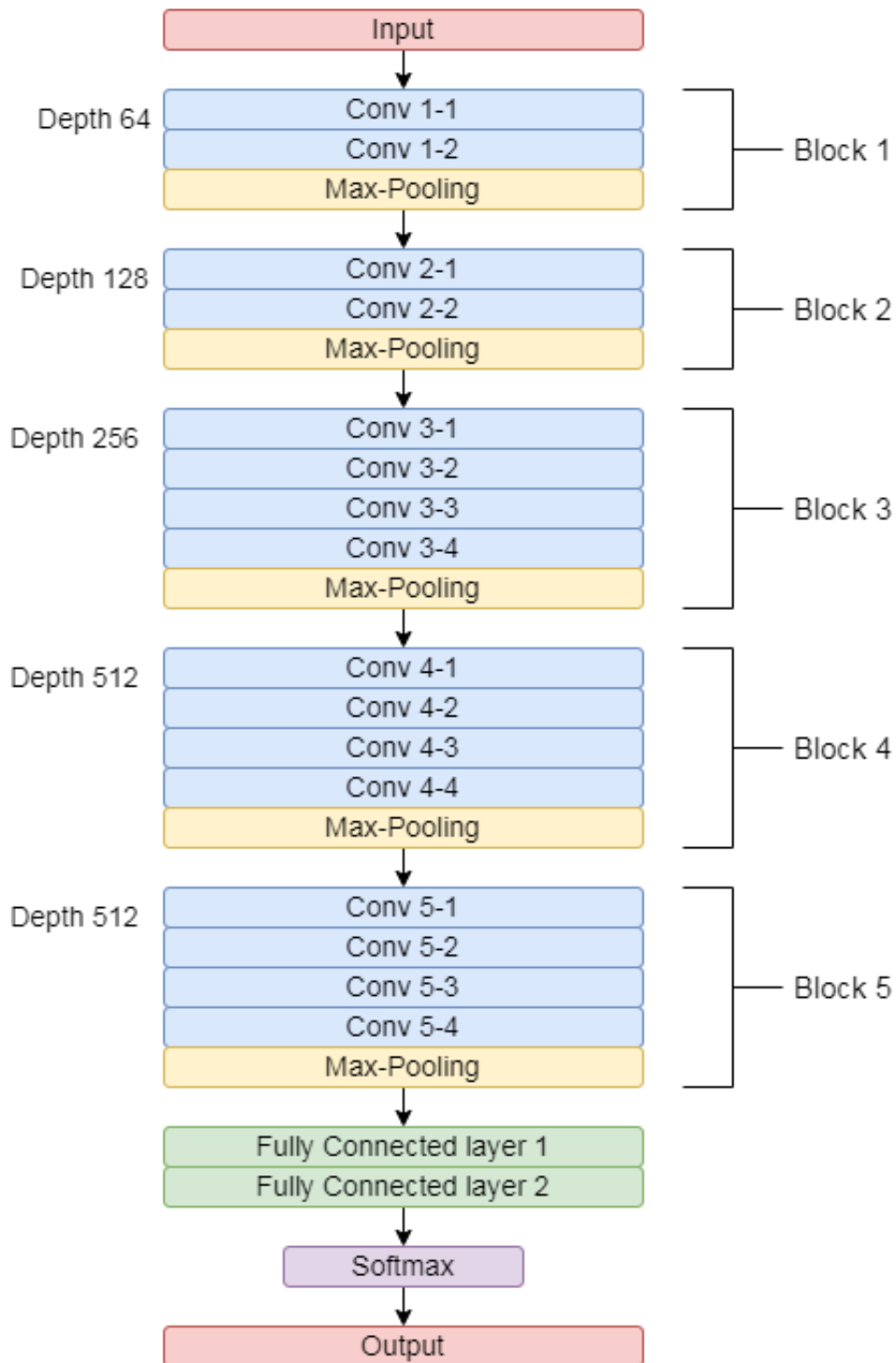


Figure 4.9: VGG19 Model

For implementing VGG-19 we used the built-in vgg19 model of keras. However, we used the `include_top=False` in order to remove the previously existing fully connected layers and to use our own data to train the model. The input size is set to $224 \times 224 \times 3$. Next, a GlobalAveragePooling layer is added. After that we created 3 different branches which will return the class values of 168 grapheme roots, 11 vowel diacritics and 8 consonant diacritics respectively. Each output layer consists of one dense layer of unit 512, 1 Dropout layer and finally another Dense layer of unit 256. Accuracy metrics have been used along with `Sparse_categorical_crossentropy` as we have not used any one hot encoding . We have also set the learning rate to be .001 initially and .0001 after 10th epoch in order to get better model training. During prediction, the model returns the class values in sorted order based on prediction value. The first values of the arrays of each output branch are considered to be predicted grapheme root, vowel diacritic and consonant diacritic.

Chapter 5

Reconstruction

5.1 Reconstruction of Graphemes

Reconstruction refers to the process of converting a graphical, numeric or any other representation of characters to a computer understandable and editable text format. Throughout the years, reconstruction of Bangla characters has been quite challenging because of vast variations and combinations of characters. So we tried to solve this problem by using the grapheme approach. In our required dataset, the images are labeled with numeric values that are later called in as input for our model training. We mapped those values to the corresponding grapheme root, vowel diacritic and consonant diacritic. However, the main challenge was to merge them into a single grapheme.

5.2 Challenges of Reconstruction and proposed solutions

During building the output, the diacritics did not seem to be adjacent as required for proper character presentation. In some cases, the root was in first position then the consonant diacritic and finally the vowel diacritic. Again, in some cases, the consonant was in first position followed by grapheme root and vowel diacritic. In order to solve this issue, we tried to find some patterns of positioning and noticed that the vowel diacritics are always to be placed after the grapheme root for proper concatenation to build the required character. We also noticed that the main thing that was changing position was the consonant diacritic. The consonant diacritics seem to be following some sort of rules.

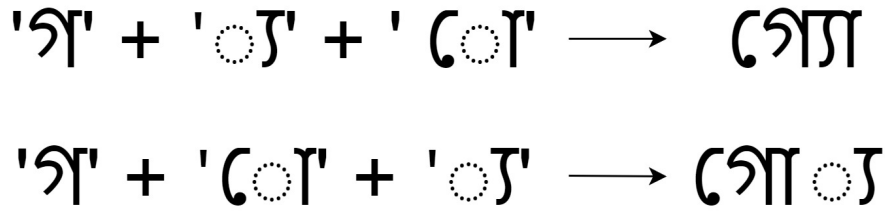


Figure 5.1: Consonant diacritics in middle

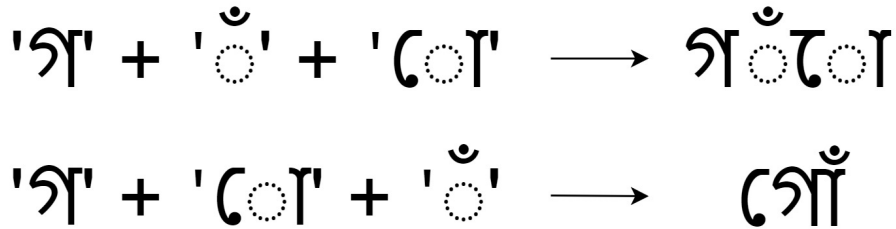


Figure 5.2: Consonant diacritics in end

The issue consonant diacritics are causing can be called unicode syntax error where the diacritics are not placed according to their adjacent placement. As mentioned above, the diacritics are labeled alongside the grapheme roots. For this reconstruction methodology we have come up with an approach of creating separate arrays that consist of the numerics in order to call for identifying as their respective positions. The positions for vowel diacritics are recalled after the roots. However, the consonants are likely to be placed before the roots or in the end or sometimes before vowel diacritics as in the middle. So, we created some arrays named as:

- consonant_middle
- consonant_after
- consonant_before
- consonant_combined

These arrays are checked while calling a function named `get_grapheme` that takes the numeric values of grapheme root, vowel diacritics and consonant diacritics as parameters and returns the text formatted corresponding grapheme. In order to do so, it first checks which array does the consonant diacritic belong to. If it is in `consonant_middle`, then it is concatenated between grapheme root and vowel diacritic. Similarly, in case of `consonant_before` and `consonant_after` it is concatenated before root and after vowel respectively. The `consonant_combined` represents the combination of two different consonant diacritics. In such cases we noticed that the first one is added before root and the second one is added before vowel diacritic. So, we constructed our character accordingly in order to get the proper result.

Next, we faced another problem which was finding the representation form of the bangla consonant diacritics ‘ৰেফ’. For other diacritics we were easily able to find the separated form of each diacritics. For example, for য-ফলা we got ্য, for ব-ফলা we found ঞ. However, there is no such representation for the consonant diacritics ‘ৰেফ’. In order to solve this problem, we tried to find out a pattern for ‘ৰেফ’ in unicode and found that the unicode of ‘ৰেফ’ is `\u09b0\u09cd`. So we used the unicode directly instead of character representation in order to get the proper reconstructed result.

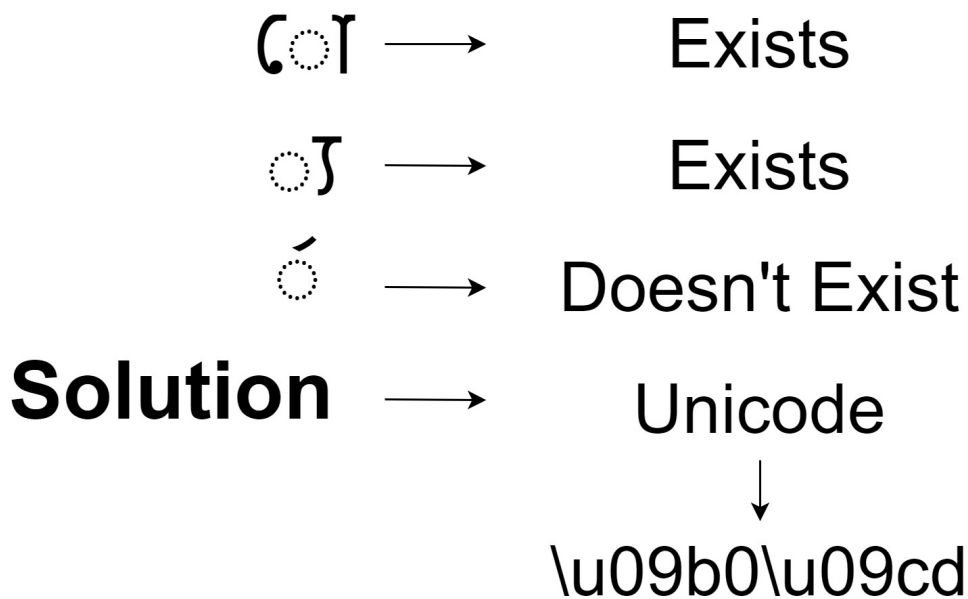


Figure 5.3: Solution for the representation problem of ‘ৰেফ’

5.3 Features

In traditional one input one output classification method applied in paper [4], the model can predict only the characters that are used to train the model. It can not predict any out of dictionary character as there is no common pattern. As a result, a huge amount of data is required to train the model. However in our case, the program can recognize out of dictionary characters.

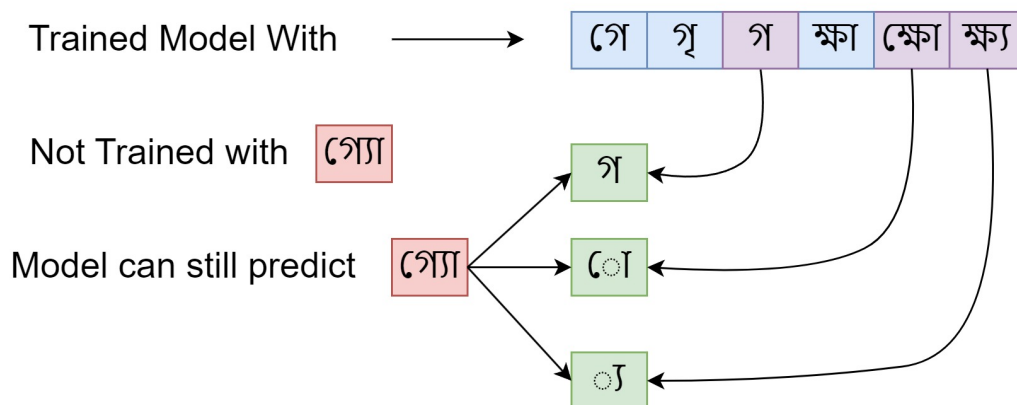


Figure 5.4: Out of the dictionary grapheme prediction

For example, let us assume that we train our model with ক্ষা, ক্ষো, ক্ষ্য, গৃ, গ and গো and we want to predict গো. One thing to note here is that, we didn't train our model with গো. Still our model would be able to predict it. As shown in the figure, গ directly exists, So, model will detect the grapheme root from there. Next, ো would come from ক্ষা and ্য would be detected from ক্ষ্য. This is how our model will give grapheme root, vowel diacritic and consonant diacritic value separately. Finally, we would combine them and get our desired result. Because of this approach, the number of classes is getting reduced which is ultimately reducing the size of our dataset.

Finally, in paper [4], they used direct searching in the dataset method to find the reconstructed character. This method might work perfectly if the dataset is huge and considers all variation of characters. The drawback of this method is that it won't be able to reconstruct any out of dictionary character. As a result, if we had implemented this approach in our program, then out of dictionary characters would have been recognized but reconstruction would not have been possible. As we are using a concatenation method applying proper sequence, our program can not only identify out of dictionary characters properly, but also reconstruct it and give us our desired character in editable text format.

Chapter 6

Results and analysis

6.1 Evaluation Metrics

For analysing a model's capability, evaluation metrics have a significant role through which we can help us evaluate the performance of the model more precisely and accurately. After training the on a labeled dataset, we can obtain different performance metrics, for instance, Accuracy, Precision, Recall and F1-score. Rather than only using any definite metric like accuracy, we evaluate our machine model even better by other metrics as well. In this proposed machine model, we have dealt with Accuracy, Precision and Recall and F1-Score.

Accuracy determines the overall correctness of the model. For measuring accuracy in both models, the ratio of total number of correct predictions is to be calculated in perspective to total instances. The total instances contain true positives, true negatives, false positives and false negatives. The correct predictions are the combination of true positives and true negatives.

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FalseNegative}$$

The precision reflects the ability to predict the positive instances correctly. It initiates the number of true positiveness in ratio to the total number of predicted positiveness.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Precision is used to decrease the incorrect positive predictions due to high false positives.

Recall is also known as sensitivity which also determines the model's ability to identify the positive instances correctly. It calculates the true positiveness in ratio to the actual positiveness.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

6.2 Results and Comparison of different models

This section evaluates and presents the results of our experiment for the handwritten Bangla character recognition task. We have experimented with both machine learning and deep learning models in order to determine the best performing model. We have placed more emphasis on deep learning models than machine learning models, as we have observed in numerous studies that deep learning models are more accurate for character recognition tasks and have better performance on large dataset. Moreover, neural network models tend to perform better when learning detailed features and patterns in the data. Therefore, we have opted for SVC as our machine learning approach. We chose and experimented with Custom CNN model, ResNet-50, and VGG-19 as our deep learning models. Here, we have employed accuracy, precision, and recall as evaluation metrics for analyzing and comparing the performance of our models.

6.2.1 SVC

We have opted for SVC as our machine learning approach. However, the accuracy result turned out to be to 0.771 , with precision of 0.7635 and recall of 0.7501 . The result was not satisfactory. It is to be expected because SVC is not meant for such complex image classification task as Character recognition. In fact, it is designed for more simple problems. So, we did not proceed with further experiments on this model.

Model	Accuracy	Precision	Recall
SVC	0.771	0.7635	0.7501

Table 6.1: SVC Results

6.2.2 Custom CNN model

Vowel Diacritics

We ran the Custom CNN model for 30 epochs. For the vowel diacritics, we noticed that the training accuracy drastically increased till 12th epochs. After that it kept increasing at a slower but constant rate. However, the validation accuracy did not improve that much after 10 epochs. It became consistent after 10 epochs. At the peak, the training vowel accuracy was 0.997 and validation accuracy was 0.965.

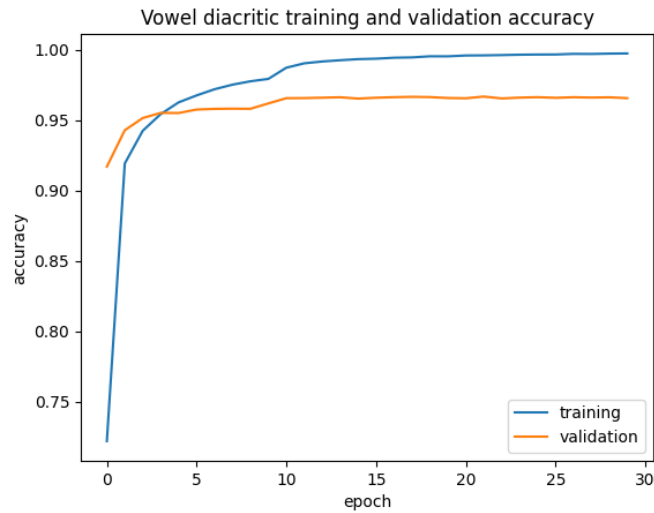


Figure 6.1: CNN vowel diacritics accuracy

However, in the case of loss, we saw that even though the training loss was decreasing at every epoch, the validation loss kept increasing after 10 epochs. So, it started to overfit at that stage.

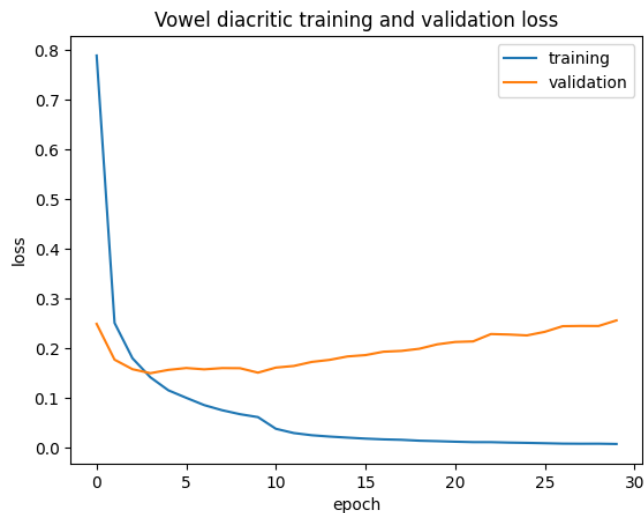


Figure 6.2: CNN vowel diacritics loss

From the confusion matrix we can see that even though majority classes were correctly predicted, still some of the classes were being wrongly predicted by this model. As we can see, the model tried to predict almost all of the labels with some data as label 0. This resembles that there is still room for improvements.

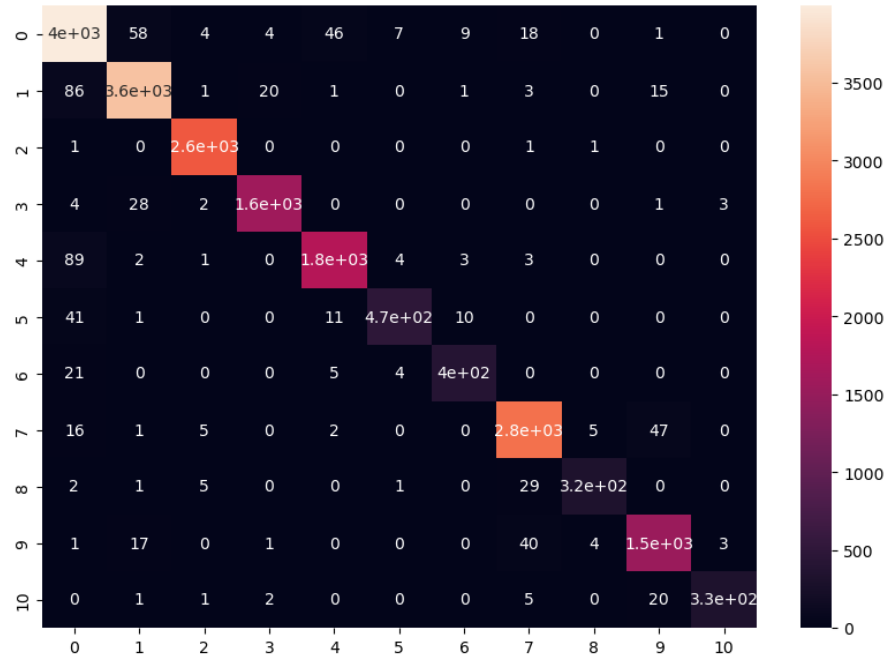


Figure 6.3: CNN Vowel Diacritics Confusion Matrix

Consonant Diacritics

The changes in consonant diacritics is also similar to vowel diacritics. Here it rose significantly till 11th epoch and finally after 30th epoch it reached 0.9979 with validation accuracy of 0.9618. In the case of losses, similar to that of vowel diacritics, the validation for consonant diacritics loss kept increasing. This shows there was overfit.

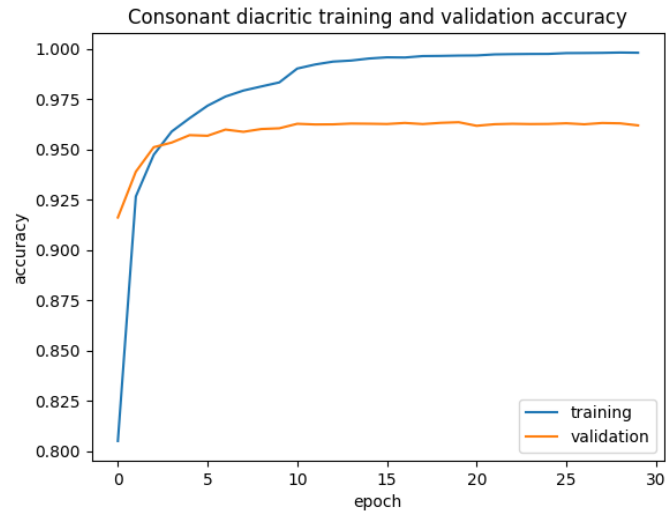


Figure 6.4: CNN Consonant Diacritics accuracy

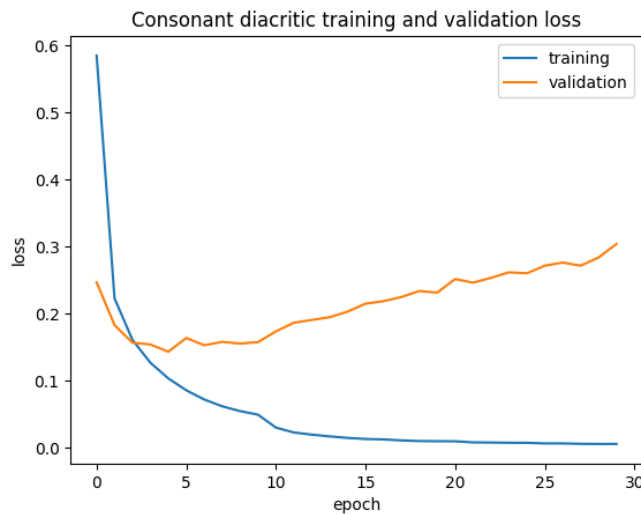


Figure 6.5: CNN Consonant Diacritics Loss

In the case of consonants we can see that there are 3 huge errors. This model predicted images of Class 2,4,5 as class 0. This indicates that the model is predicting too much class 0 and the model fails to perform in case of the consonant diacritics.

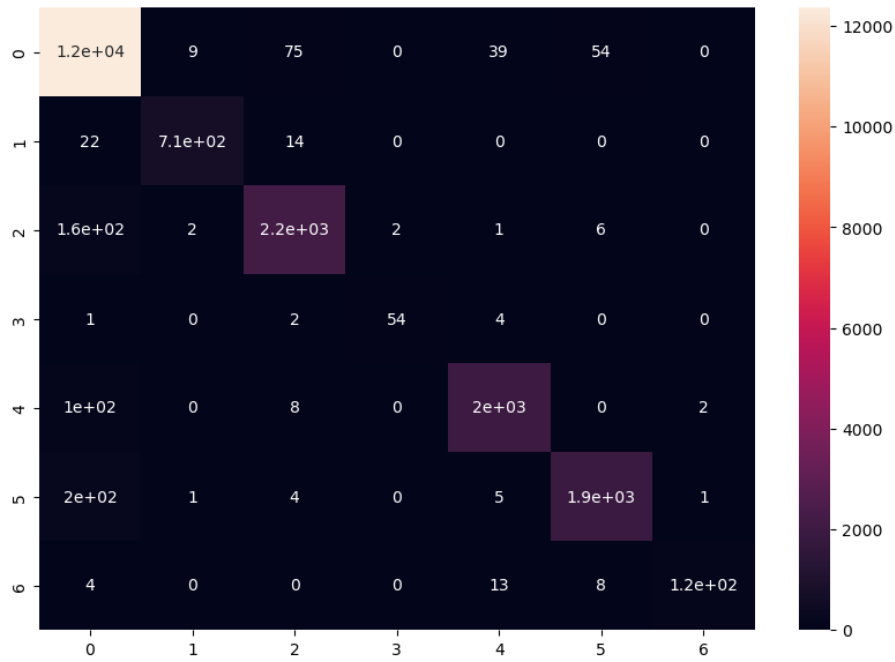


Figure 6.6: CNN Consonant Diacritics Confusion Matrix

Grapheme Root

Finally, the grapheme root's training accuracy kept improving. However, the validation accuracy became 0.87 after the 12th epoch and remained almost constant after that. This might be the reason of getting low accuracy in grapheme root. Similar to vowel and consonant diacritics, the validation loss kept increasing at a slow rate even though the training loss was decreasing.

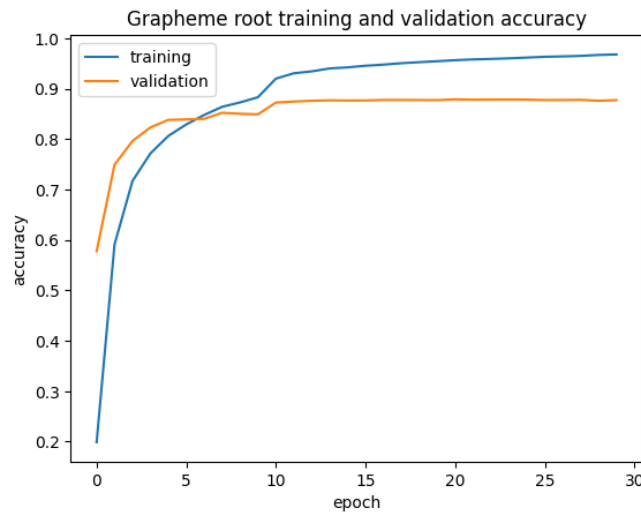


Figure 6.7: CNN Grapheme Root Accuracy

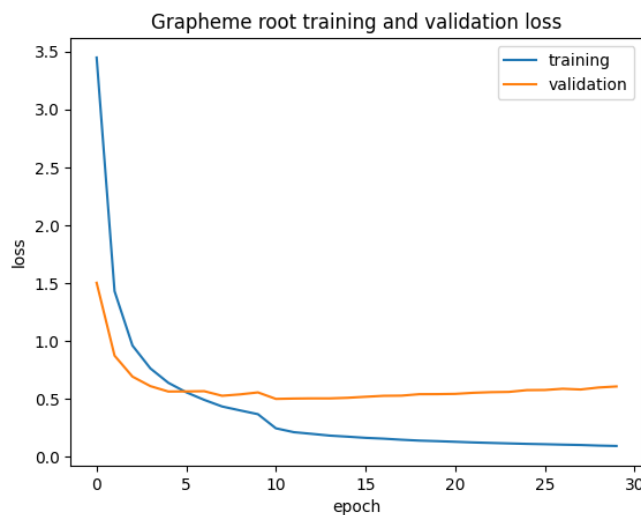


Figure 6.8: CNN Grapheme Root Loss

We can clearly see that the Custom CNN did quite well compared to the SVC. However, the results were still average and not up to the mark all the time. Though vowel diacritics and root accuracy is good, the grapheme root accuracy is pretty low. Also, the consonant diacritics confusion matrix shows it was not able to predict class 0 images properly. So, it denotes that in order to achieve better results, we felt the need of more better and complex models.

6.2.3 ResNet-50

Vowel Diacritics

We ran the ResNet-50 for 20 epochs as there was not that much change . In case of vowel diacritics, the model initially seemed to be performing well. However, at the 11th epoch, there was a huge deviation in validation accuracy and huge increase in validation loss of vowel diacritics. The accuracy almost decreased by 45 percent. Such deviations can also be observed at 16th and 19th epoch which indicates that the model might not be able to detect vowel diacritics properly which ultimately resulted in low accuracy in vowel diacritics.

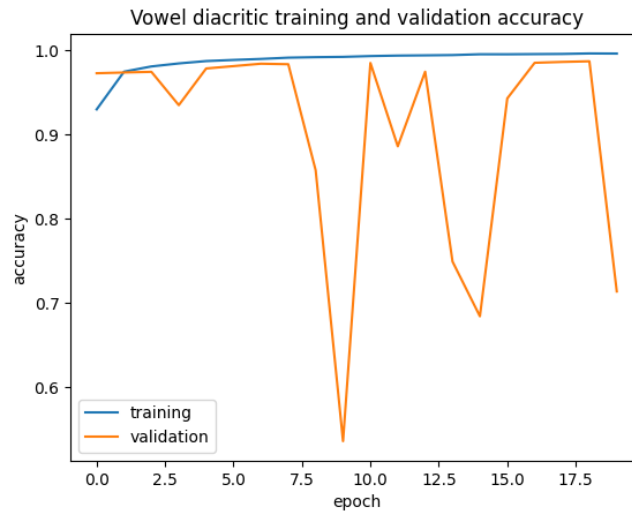


Figure 6.9: ResNet-50 Vowel Diacritics Accuracy

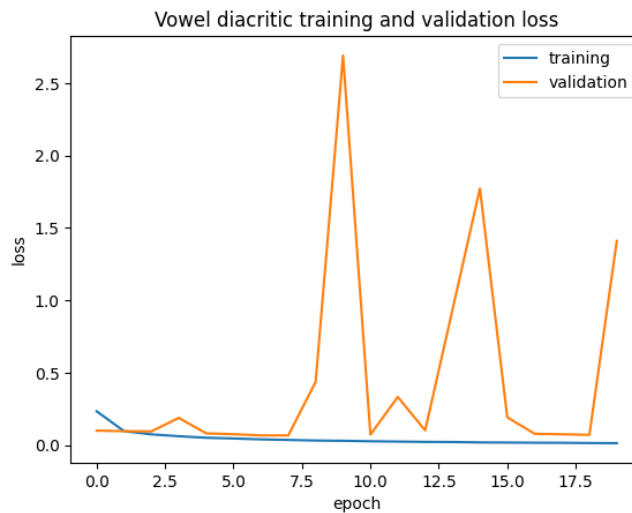


Figure 6.10: ResNet-50 Vowel Diacritics Loss

Lots of errors can be seen from the confusion matrix of vowel diacritics of Resnet50. Especially class 0,1,4 and 7 is not being predicted properly.

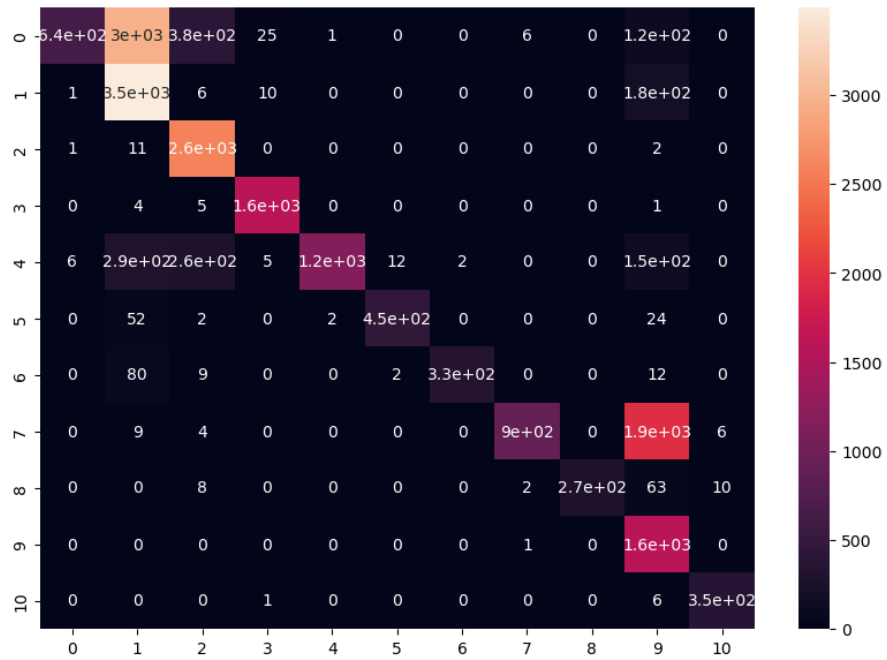


Figure 6.11: ResNet-50 Vowel Diacritics Confusion Matrix

Consonant Diacritics

In case of consonant diacritics there is also a huge change in accuracy and loss in 11th epoch. However, it is not that significant. It is just around 10 percent which can be considered as normal.

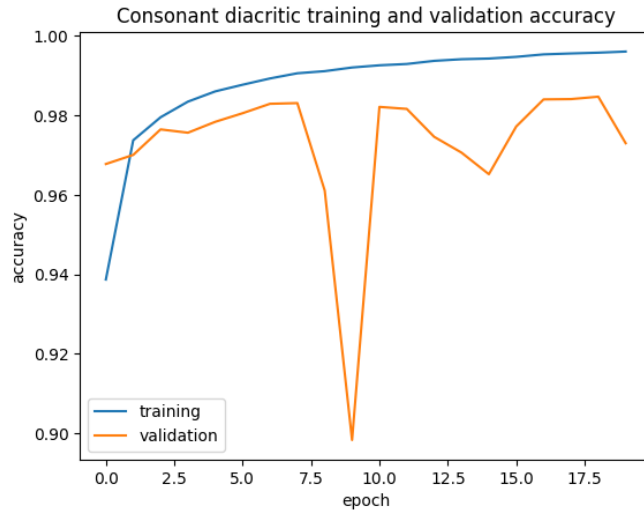


Figure 6.12: ResNet-50 Consonant Diacritics Accuracy

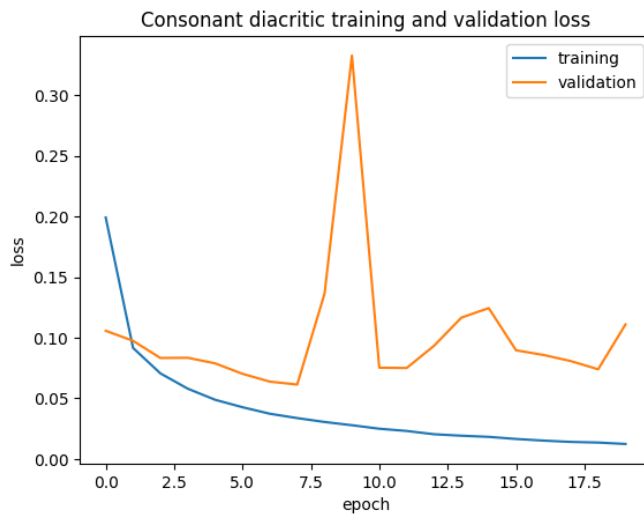


Figure 6.13: ResNet-50 Consonant Diacritics Loss

The matrix of consonant diacritics class seems normal apart from few errors in class 0.

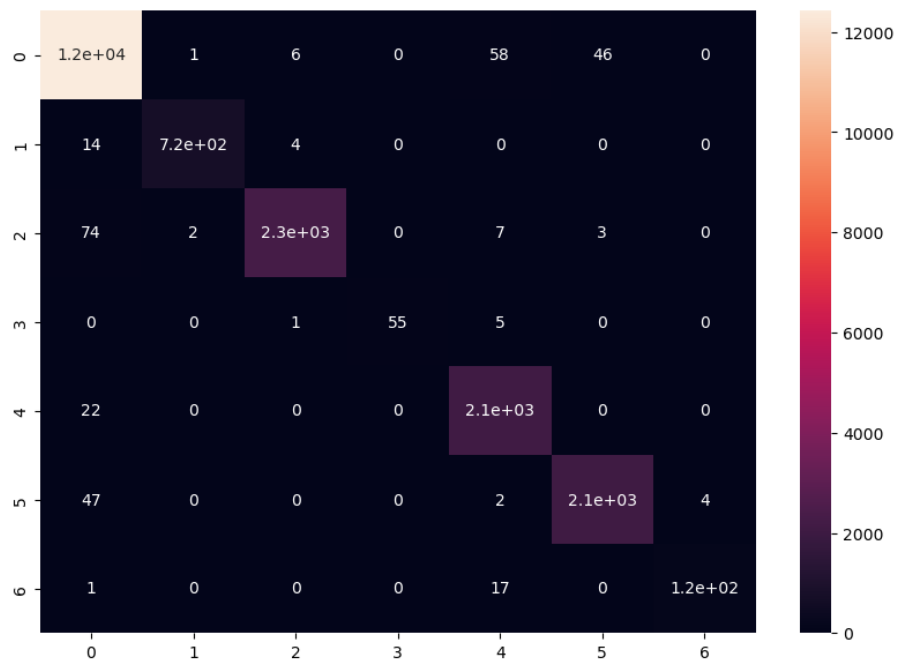


Figure 6.14: ResNet-50 Consonant Diacritics Confusion Matrix

Grapheme Root

Just like consonant diacritics , there is a decrease in accuracy in case of grapheme root too in the 11th epoch. However, in the case of 16 th and 19th epoch, the model was able to recover from it which led to a good final accuracy in grapheme root.

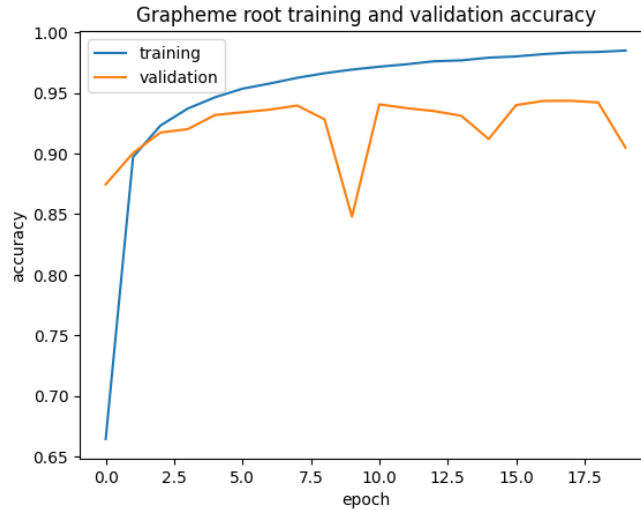


Figure 6.15: ResNet-50 Grapheme Root Accuracy

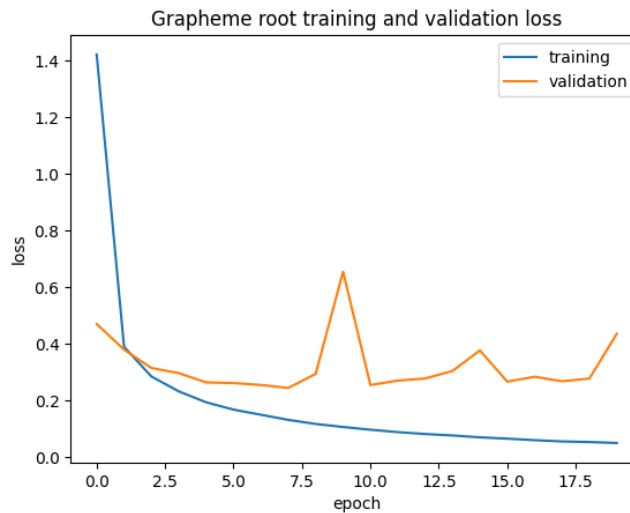


Figure 6.16: ResNet-50 Grapheme Root Loss

The ResNet-50 performed extremely well in case of consonant diacritics and grapheme roots. However, in the case of vowel diacritics, it had a huge downfall. It almost dropped around 40 percent which is not acceptable. So, we moved on to the next model which is VGG19.

6.2.4 VGG19

Vowel Diacritics

Just like ResNet50, we ran the Vgg19 for 20 epochs too. In case of vowel diacritics, the validation accuracy jumped within the range of .97 to .99. No significant difference was noticed here. The validation loss was also near to constant.

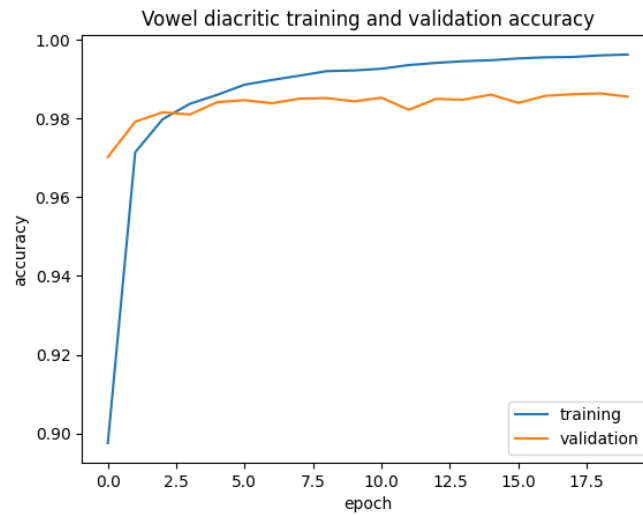


Figure 6.17: VGG19 Vowel Diacritics Accuracy

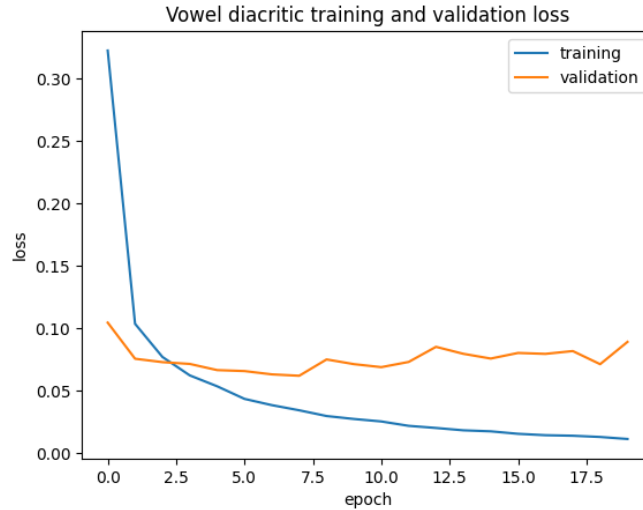


Figure 6.18: VGG19 Vowel Diacritics Loss

From the matrix we can see almost every class is being predicted properly.

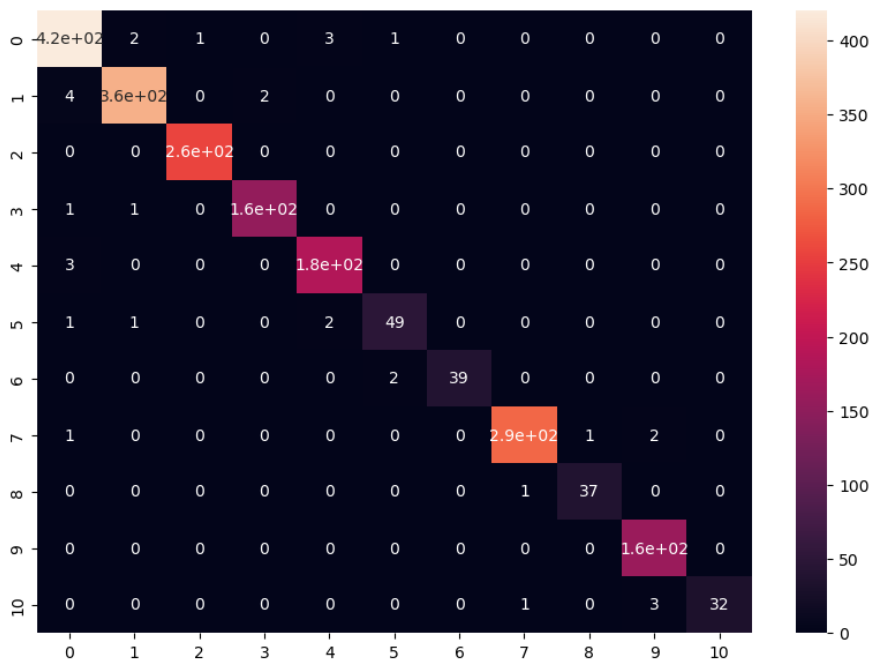


Figure 6.19: VGG19 Vowel Diacritics Confusion Matrix

Consonant Diacritics

The consonant validation accuracy was also good and no huge deviation was noticed.

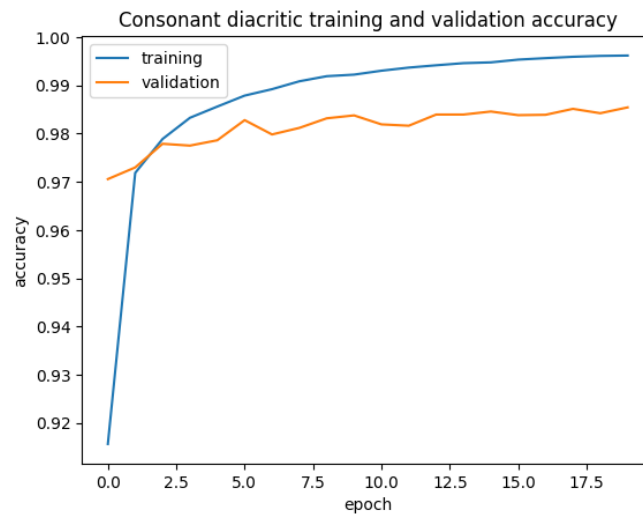


Figure 6.20: VGG19 Consonant Diacritics Accuracy

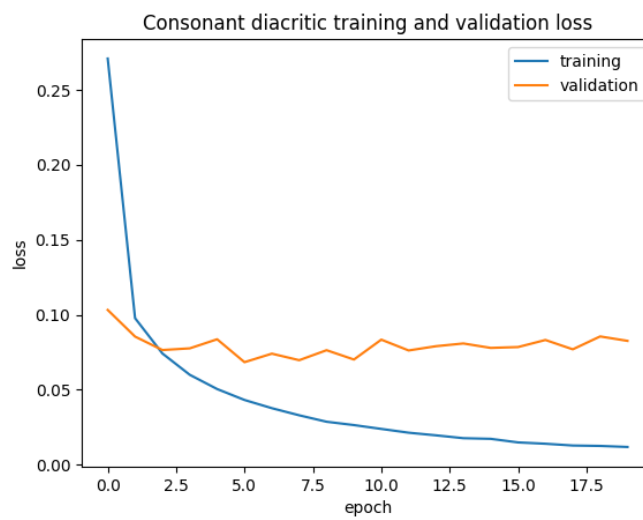


Figure 6.21: VGG19 Consonant Diacritics Loss

Just like vowel diacritics, Vgg19 is also able to predict almost all consonant classes properly.

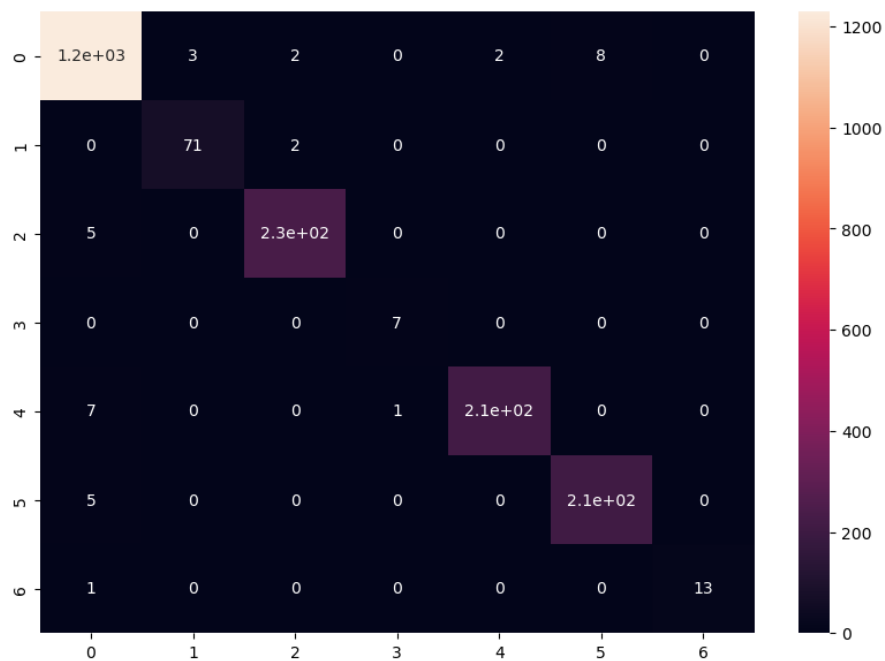


Figure 6.22: VGG19 Consonant Diacritics Confusion Matrix

Grapheme Root

Finally, the grapheme root accuracy kept changing between .944 to .949 after the 12th epoch. Similarly the training accuracy hardly got more than .98. So, stopped the epoch at 20 as there was no change in neither accuracy nor loss.

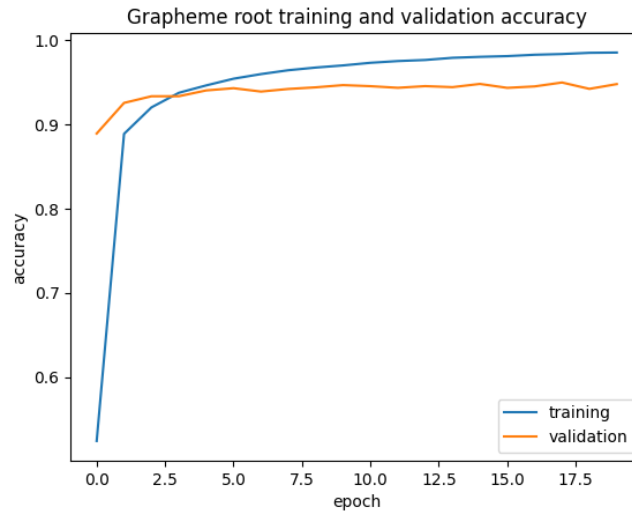


Figure 6.23: VGG19 Grapheme Root Accuracy

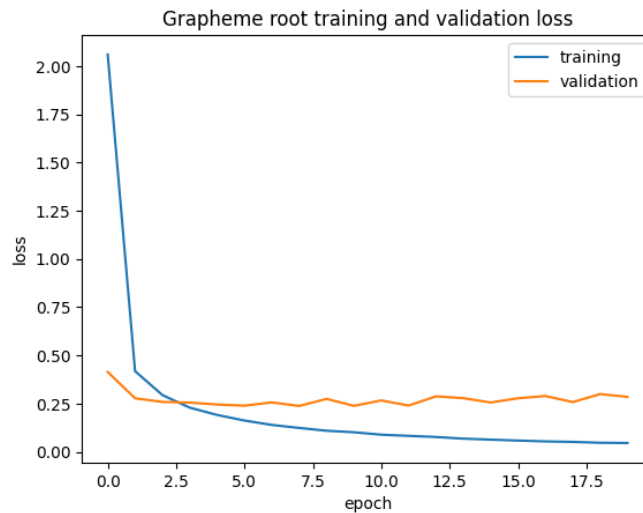


Figure 6.24: VGG19 Grapheme Root Loss

Here we can see that VGG19 was able to perform all three of the grapheme root, vowel diacritic and consonant diacritic with high accuracy. Though the accuracy of grapheme root is slightly lower than ResNet50, the overall accuracy was very satisfactory. So we decided to use VGG19 as our final model.

6.3 Comparison

	Accuracy		
	Custom CNN	ResNet50	VGG19
vowel diacritics	0.9642	0.66669	0.98357
consonant diacritics	0.9635	0.98436	0.98208
grapheme root	0.88	0.94809	0.94325

Table 6.2: Accuracy comparison of all the models

	Precision		
	Custom CNN	ResNet50	VGG19
vowel diacritics	0.96438	0.82557	0.98367
consonant diacritics	0.96356	0.98448	0.9822
grapheme root	0.88131	0.94649	0.94689

Table 6.3: Precision comparison of all the models

	Recall		
	Custom CNN	ResNet50	VGG19
vowel diacritics	0.9642	0.6667	0.98357
consonant diacritics	0.9635	0.98437	0.98208
grapheme root	0.88	0.94209	0.94326

Table 6.4: Recall comparison of all the models

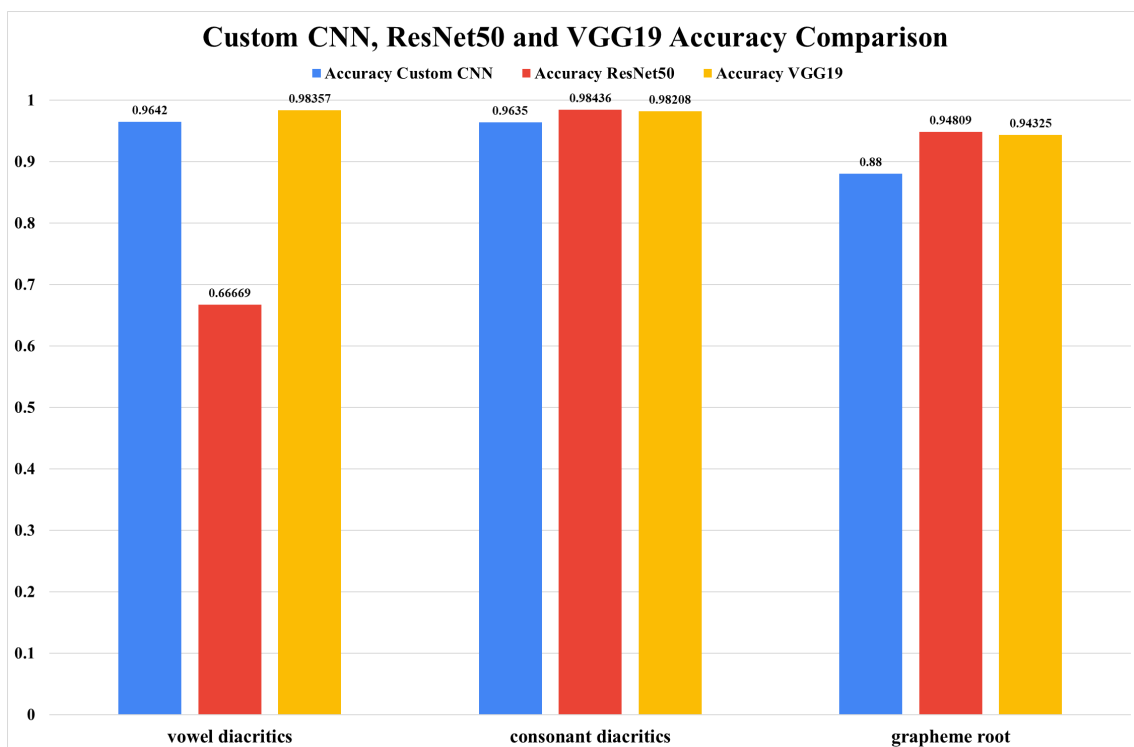


Figure 6.25: Comparison of all the models

Chapter 7

Conclusion

The very purpose of this project started with the intention of making an efficient Bangla handwritten character recognition system. As stated earlier, this requirement was in demand despite the number of researches on this topic being infrequent. Isolated or cursive styles within individual handwriting was making it complex for recognising alpha syllabary languages like Bangla. We defined an efficient system within goals of recognising the basic Bangla characters as well as the conjunct characters. Deep neural networks and machine learning models have been approached in previous works, but couldn't result in great accuracy for recognizing both the simple and compound Bangla handwritten characters. We approached NLP for applying grapheme segmentation. Grapheme, a linguistic fragment of words, helped us to train the model with grapheme roots and diacritics. Through NLP and Computer Vision we trained our models with larger dataset. The very intention of approaching grapheme segmentation was to avoid the extra data considering all the compound characters which are almost infinite. Training models in the grapheme approach not only reduced the number of data but also resulted in successful detection of compound characters that were not included in the dictionary while training. SVM as a machine learning model and DNN models like CNN, Resnet50, VGG19 were used for our approach. We also used a reconstruction approach, which not only generates the corresponding values correctly but also solves the complex ordering issue of the traditional reconstruction approach. After model implementation we trained the model with our train data and calculated the accuracy accordingly. Starting with SVC as a machine learning approach the result was not satisfactory, considering it not suitable for such complex image classification. Custom CNN gave average results with low grapheme root accuracy. Also couldn't predict the class 0 images properly. Then, Resnet50 resulted in a huge downfall in case of vowel diacritics even though it had high accuracy in root and consonant diacritics. Therefore, we moved to the VGG19 model. It performed with high accuracy. Despite being slightly lower in grapheme root accuracy, the overall result was satisfactory. Thus, VGG19 was chosen as the final model. This has been challenging but we overcame it and solved most of the research problems stated before. To conclude, we have successfully been able to build a Bangla handwritten character recognition system which will contribute to future research in this field.

Bibliography

- [1] Samiul Alam et al. “A Large Multi-target Dataset of Common Bengali Handwritten Graphemes.” In: *Document Analysis and Recognition – ICDAR 2021*. Springer International Publishing, 2021, pp. 383–398. DOI: 10.1007/978-3-030-86337-1_26. URL: https://doi.org/10.1007%2F978-3-030-86337-1_26.
- [2] Md Zahangir Alom et al. *Handwritten Bangla Digit Recognition Using Deep Learning*. 2017. arXiv: 1705.02680 [cs.CV].
- [3] Md Shopon, Nabeel Mohammed, and Md Anowarul Abedin. “Bangla handwritten digit recognition using autoencoder and deep convolutional neural network.” In: *2016 International Workshop on Computational Intelligence (IWCI)*. IEEE. 2016, pp. 64–68.
- [4] Rumman Rashid Chowdhury et al. “Bangla handwritten character recognition using convolutional neural network with data augmentation.” In: *2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE. 2019, pp. 318–323.
- [5] Sharif Amit Kamran et al. *AI Learns to Recognize Bengali Handwritten Digits: Bengali.AI Computer Vision Challenge 2018*. 2018. arXiv: 1810.04452 [cs.CV].
- [6] Oddur Kjartansson et al. “Crowd-Sourced Speech Corpora for Javanese, Sundanese, Sinhala, Nepali, and Bangladeshi Bengali.” In: *Proc. The 6th Intl. Workshop on Spoken Language Technologies for Under-Resourced Languages*. 2018, pp. 52–55. URL: <http://dx.doi.org/10.21437/SLTU.2018-11>.
- [7] Md Hussain et al. “Deep Learning based Bangla Voice to Braille Character Conversion System.” In: Oct. 2022, pp. 0262–0267. DOI: 10.1109/IEMCON56893.2022.9946619.
- [8] Mithun Biswas et al. “BanglaLekha-Isolated: A multi-purpose comprehensive dataset of Handwritten Bangla Isolated characters.” In: *Data in Brief* 12 (June 2017), pp. 103–107. DOI: 10.1016/j.dib.2017.03.035.
- [9] Md Mahbubar Rahman et al. “Bangla handwritten character recognition using convolutional neural network.” In: *International Journal of Image, Graphics and Signal Processing (IJIGSP)* 7.8 (2015), pp. 42–49.
- [10] Umapada Pal, Tetsushi Wakabayashi, and Fumitaka Kimura. “Handwritten Bangla compound character recognition using gradient feature.” In: *10th international conference on information technology (ICIT 2007)*. IEEE. 2007, pp. 208–213.

- [11] Tapan Kumar Bhowmik, Ujjwal Bhattacharya, and Swapan K Parui. “Recognition of Bangla handwritten characters using an MLP classifier based on stroke features.” In: *Neural Information Processing: 11th International Conference, ICONIP 2004, Calcutta, India, November 22-25, 2004. Proceedings 11*. Springer. 2004, pp. 814–819.
- [12] Tapan Kumar Bhowmik et al. “SVM-based hierarchical architectures for handwritten Bangla character recognition.” In: *International Journal on Document Analysis and Recognition (IJDAR)* 12.2 (2009), pp. 97–108.
- [13] Swapan K Parui et al. “Online handwritten Bangla character recognition using HMM.” In: *2008 19th International Conference on Pattern Recognition*. IEEE. 2008, pp. 1–4.
- [14] Kaiming He et al. “Deep Residual Learning for Image Recognition.” In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.
- [15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].