**Okkhor: A Synthetic Corpus of Bangla Printed Characters**

Mridul Banik 1,, Md Jamiur Rahman Rifat 1, Jebun Nahar1 , Nazmul Hasan1 and Fuad Rahman2

1Apurba Technologies, Dhaka, Bangladesh.

2Apurba Technologies, Sunnyvale, CA, USA

{mridul,rifat,jebunnahar,nazmul,fuad}@apurbatech.com

**Abstract.** Bangla is the fifth most-spoken native language in the world. Despite having such a large number of speakers, the resources related to development of language processing solutions are very limited. To realize the full potential of Machine Learning (ML) and Artificial Intelligence (AI) solutions for computer vision and Natural Language Processing (NLP), a complete and standardized fully-annotated corpus is an essential prerequisite. Specifically, development of Optical Character Recognition systems (OCRs) for printed characters, an important resource for language automatic and digitization, requires a large corpus with high coverage and variability of fonts, representing the nuances of the language usage, which does not exist for Bangla. In this paper, we present a novel synthetic corpus of over 5 million printed Bangla characters containing 60 alphanumeric characters, 10 vowel modifiers, 159 compound characters, which corresponds to 229 different classes of both Unicode and ASCII encodings. This is entirely novel work, since there exists no such corpus currently for the Bangla language.

**Keywords:** Corpus, Synthetic Data, OCR, True Type Font Extension, Histogram, Contour, Dilation, Erosion, Unicode

1. **Introduction**

Bangla is one the most commonly spoken languages of the world. Although it is widely used, unfortunately, Bangla has not been digitized like the other languages. Development of many of the resources needed for automated processing of languages such as labeled language corpora, language modeling, Named Entity Recognition (NER), Parts of Speech (POS) tagging, Optical Character Recognition (OCR) etc. are still not mature for Bangla. There have been efforts in recent times [1-8] to build a representative Bangla corpora, but a lot of these efforts are still in their early stages. This showcases the need for building a corpus of synthetic data while a representative corpus is still being developed and adequately annotated. This paper presents the first-ever synthetic corpus built for Bangla printed characters.

1. **Literature Review**

There is no available corpus of printed Bangla characters. However, there have been sporadic and isolated attempts at building handwritten Bangla corpora. “BanglaLekha-Isolated” [1] is one such corpus. This dataset contains 84 different characters comprising 50 basic characters, 10 numerals and 24 compound characters, which is a fraction of the total number of compound characters possible in Bangla. The authors collected, digitized, and pre-processed about 2,000 handwriting samples for each of the 84 characters. After discarding mistakes and scribbles, 166,105 handwritten character images were included within the final dataset. “NumtaDB” [2] is another corpus of focusing on Bangla handwritten digits only. This corpus is a collection of more than 85,000 labelled Bangla handwritten digits, which was published by Bengali.AI Community. “Ekush” [3], on the other hand, focused on Bangla handwritten characters as a whole and not only numerals. This corpus contains 367,018 isolated handwritten characters written by 3,086 unique writers, containing Bangla modifiers, vowels, consonants, compound letters and numerical digits. Another dataset was published by the Bengali.AI community, which was on Bangla Handwritten Graphemes [4]. This dataset collected 1,000 classes of graphemes, which are basically combinations of multiple characters and their shorthands, called diatrics, a common occurrence in Bangla. Another dataset named “MatriVasha” [5] is a collection of Bangla handwritten compound characters. This dataset contains 120 unique compound characters and consists of 306,464‬ images. There has been some lesser work with smaller datasets that are either too small to mention or not publicly or commercially available [6]. In addition, there are some other research works of Bangla datasets which are designed for speech recognition, sentiment analysis, chatbot systems etc. that we have studied, but have not discussed here in detail since they are not directly related to Bangla text corpora [7, 8, 9].

1. **Methodology**
   1. **Bangla Language**

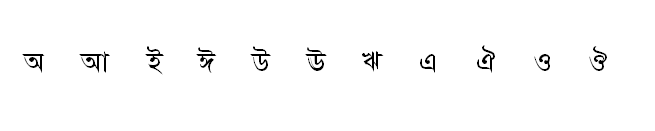
Bangla Language alphabets can be classified into vowels, which are called “Sworoborno” and consonants, which are called “Benjonborno.” Apart from these basic characters, there are vowel modifiers and consonant modifiers. These modifiers can be added with other vowels and consonants, depending on where, within a word, a specific letter combination occurs. Furthermore, two or more consonants or vowels can be combined to form completely new characters, known as compound characters, or “Jukto Borno” in Bangla.

The Bangla Language has 10 unique digits to represent numeric values, as seen in Fig. 1.



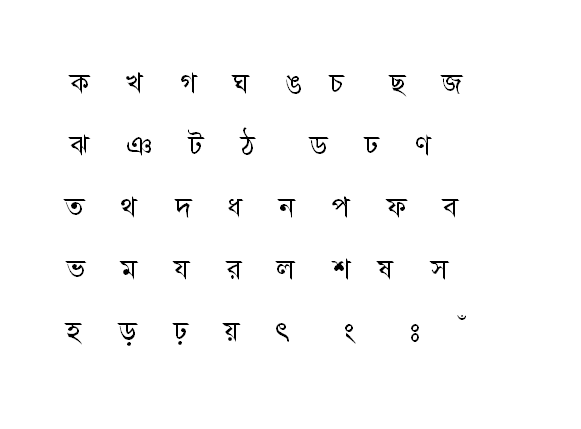
**Fig 1.** Bangla digits

There are 11 vowels (“Sworoborno”) in Bangla Language, as seen in Fig. 2.



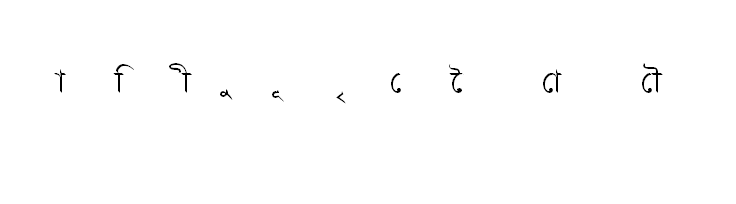
**Fig. 2.** Bangla vowels

There are 39 consonants (Byenjonborno) in Bangla, as seen in Fig. 3.



**Fig. 3.** Bangla consonants

In Bangla, there are 11 vowel modifiers, which have been illustrated in Fig. 4.



**Fig. 4.** Vowel modifiers

There are many possible compound characters in Bangla. We have documented 159widely used compound characters in our corpus, as seen in Fig. 5.



**Fig. 5.** Compound characters

* 1. **Bangla Font Encoding**

To digitally support fonts of different languages, character encoding is needed for internal representations of these characters [10]. The process of transforming a character symbol into a binary number and then using a map to read that binary number as a letter is called character encoding.

### **Unicode Encoding.** Unicode is an information technology standard which is used to encode, decode, represent and handle most of the world’s writing systems [11]. The Unicode Standard includes a set of code charts for visual citation, an encoding process and a set of standard character encodings, and a number of related items, such as character properties, a set of reference data files, rules for normalization, collation, decomposition, rendering, and bidirectional text display order (for the correct display of text containing both right-to-left scripts, such as Arabic and Hebrew, alongside the more common, left-to-right scripts, such as Bangla and English). For the Bangla Language, we have identified a set of Unicode fonts which are included in our corpus.

**ASCII Encoding.** ASCII, a character encoding that is used for computers to save and retrieve characters (letters, symbols, indentations, spaces, numbers etc.) as bit-patterns for storage. The widely used Bangla keyboard “Bijoy” supports a wide variety of ASCII encoded fonts [12]. We have included these fonts in our corpus.

* 1. **Data collection from Available Sources**

**Character Collection**. We have collected fonts from different sources. Basic characters are very common and widely used. We studied the different existing databases such as “BanglaLekha-Isolated,” “Ekushe,” “MatriVasha,” and “NumtaDB.” Although these databases agree on the basic characters, the number of compound characters is different in different sources. We decided to take all the unique compound characters found in these different sources and included those in our corpus. The primary sources of our compound characters are “BanglaLekha-Isolated,” “Ekushe,” “MatriVasha,” and the grapheme dataset by “Bengal.AI” community. The rest of our compound characters have been collected from “Bijoy,” a widely used Bangla keyboard layout software.

**Font Collection**. We have collected 45 ASCII fonts, which were found from Bijoy Ekushe Keyboard Software. In addition, we have used 17 Unicode fonts. We have collected a total of 62 different Bangla fonts. The printed character images in this corpus have been extracted from these 62 fonts.

* 1. **Metadata**

To construct the synthetic corpus, the first step was to build a font map. We have assigned numeric values to the various classes. All the same characters having a unique value is saved under the same class name. All these characters are mapped with a unique number. We have assigned the following list as our metadata, as seen in Table 1.

Our corpus has 5 types of characters which are listed in the first column of Table 1. The second column shows how we mapped characters with class names. Here, numerals are represented as 1-10 numeric values. Vowels and consonants are mapped from 11-21 and 22-60 respectively. On the other hand, vowel modifiers are represented from 61-70 and compound characters are from 81-239. The third column is the number of characters of these 5 types of classes. The last row shows that the corpus has 229 characters in total.

**Table 1.** Metadata

|  |  |  |
| --- | --- | --- |
| **Characters** | **Numeric Class** | **Number of Character** |
| 0-9 Numerals | 1-10 | 10 |
| Vowels (Sworoborno) | 11-21 | 11 |
| Consonants (Benjonborno) | 22-60 | 39 |
| Vowel Modifiers | 61-70 | 10 |
| Compound Characters | 81-239 | 159 |
| **Total Number of Characters** |  | **229** |

* 1. **Font Image Extraction**

The second step is to extract the font images from these different font samples coming from different font families, both Unicode and ASCII.

### **ASCII Font Image Extraction.** As mentioned earlier, there are two types of encoded fonts in Bangla language, Unicode and ASCII. They differ widely in their encoding and other internal representations. Therefore, it is challenging to extract font images automatedly from these font files. This section documents the different methods employed for extracting images from these true type font (ttf) files. We ended up using both manual and automated methods to extract font images from these fonts.

In the automated process, we have adopted a widely used Python implementation of the OpenCV library [13], especially since our approach mainly depended on image segmentation. We have also used an open source online tool to transform the PDF files into image files [14]. For the generation of different font class images, we used Microsoft Word [15].

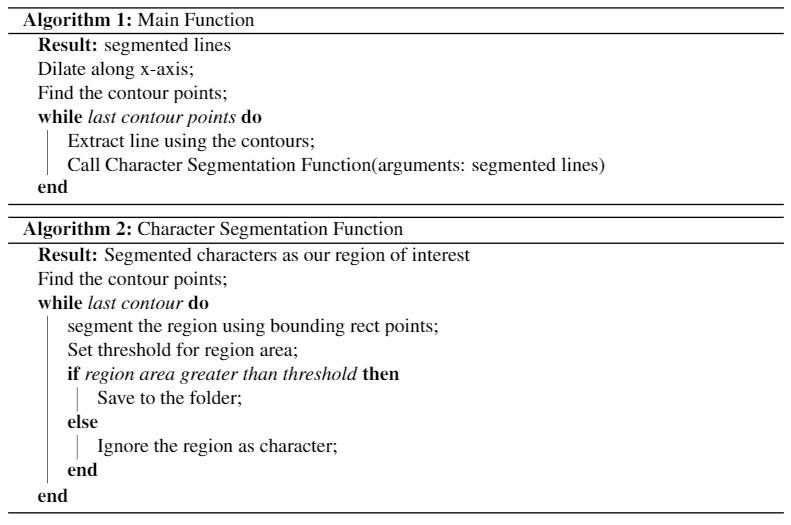
Not all the font images can be extracted by this automated process. For those, we had to resort to a completely manual approach.

*Extraction Process.* First of all, we have identified the list of targeted characters and font classes. As discussed already, we have identified 229 characters to be extracted of these three clases: vowels, consonants and compound characters. The compound character dataset has been built by combining the classes from Ekush, Matrivasha and Bijoy. We have also incorporated 45 Bijoy font classes into our dataset. The full list has been presented in Table 2. In our processing pipeline, the characters are stored in a word file. Then, these files are exported to PDF files, which in turn. are converted into two image files. The second page was assigned for all the compound characters and the rest of the characters were in the 1st page. So when it was converted down to images, we had two images per font for segmentation. These image files were then fed into our segmentation method. The extracted image files for each character are then stored in specific folders labeled as specific font classes.

*Segmentation Function*.The input image was first preprocessed using a binarization technique. The characters are separated from each other, and then we draw distinctive contours around the characters. In our processing pipeline, we used the OpenCV contours-identifying methods. To segment the characters, we first sliced the lines. We deployed the ‘draw contours’ method to identify our region of interest. This line was then fed into our character segmentation process. In the character segmentation process, the input image was first dilated and then using the ‘findContours’ method of OpenCV, the contour area of each character was calculated and sorted from left-to-right to maintain the ordering of the characters. These contours may identify a dot (.) like sign as a distinct character, therefore we have set a value of 100 bytes as threshold value: if the segmented image is greater than that threshold value, we consider that a validated character. There were some compound characters which were found to be split into two parts. To avoid this error, we have merged the adjacent contours found within a threshold. The threshold value for this process was set at 25% of the average distance of all the characters of that line. The pseudocode for the whole segmentation process has been shown in Fig 6.

**Table 2.** ASCII font families

|  |  |  |
| --- | --- | --- |
| AnandaMJ | ChitraMJ | HooglyMJ |
| ArhialkhanMJ | ChondanaMJ | IchamotiMJ |
| AtraiMJ | DhakarchithiMJ | IchamotiSreeMJ |
| BhagirathiMJ | DholeshwariMJ | IchamotiSushreeMJ |
| BhairabMJ | DhonooMJ | IrabotiMJ |
| BongshaiMJ | DhorolaMJ | JaJaDiMJ |
| BorakMJ | DorhatanaMJ | JomunaMJ |
| BorhalMJ | GangaMJ | JugantorMJ |
| BrahmaputraMJ | GangaSagarMJ | KalindiMJ |
| BurigangaMJ | GangaSushreeMJ | KanchanMJ |
| BurigangaSushreeMJ | GhorautraMJ | KarnaphuliMJ |
| ChandrabatiMatraMJ | GoomtiMJ | KashalongMJ |
| ChandrabatiMJ | GoraiMJ | KeertankhulaMJ |
| ChandrabatiSushreeMJ | HaldaMJ | KhooaiMatraMJ |
| ChengiMJ | HelenMJ | SutonnyMJ |



**Fig. 6.** Whole segmentation algorithm

**Unicode Font Image Extraction.** Unicode fonts are processed in a different way than ASCII fonts. These fonts are handled with Python programming language with the use of some font libraries. A total of 17 unicode font classes have been used in our corpus. The names are listed in Table 3.

#### The Python Imaging Library [16] is extensively used in this process. The Python interpreter got the image processing competence through the Python Imaging Library.

Various file formats are supported by this library with fairly strong image processing capabilities and coherent internal representation. The image library is structured for fast access to data stored in a variety of basic pixel formats.

* Image Module: A same name class is used for the representational purpose of PIL image[16]. The module gives a number of functions which includes functions to load images , files, and create new images.
* Image Draw Module: Simple 2D graphics for Image objects have been provided by the ‘ImageDraw’ module. This module is generally used to create new images, annotate or retouch existing images.
* ImageFont Module: The ‘ImageFont’ module is a class which is used to store bitmap fonts, and is used with the PIL.ImageDraw.Draw.text() method. PIL has its own font file format to store bitmap fonts.

**Table 3.** Unicode font families

|  |  |
| --- | --- |
| AdorshoLipi | Nikosh |
| Akaashnormal | NikoshGrameen |
| BenSenHandwriting | Rupali |
| JAJADO | Sagarnormal |
| Kalpurush | Siyamrupali |
| Lohit | SolaimanLipi |
| Mitra | Sornaly |
| Muktinarrow | SutonnyOMJ |
| Mukti |  |

#### *Extraction Process.* This process starts by importing libraries through outputting images in a certain order. At first, we imported the above mentioned libraries. The next step was to read the “True Type Font” extension files by ImageFont.truetype() function of ImageFont class. Then, we defined all the alphabets in a list which will be drawn individually. Next, we apply a context by ImageDraw.draw() method which creates an object that can be used to draw in the given image. Then, we draw images of the alphabets in the list by passing individual characters as parameters in text() method. Finally, we saved those images and organized in the font name folder with the corresponding number of metadata.

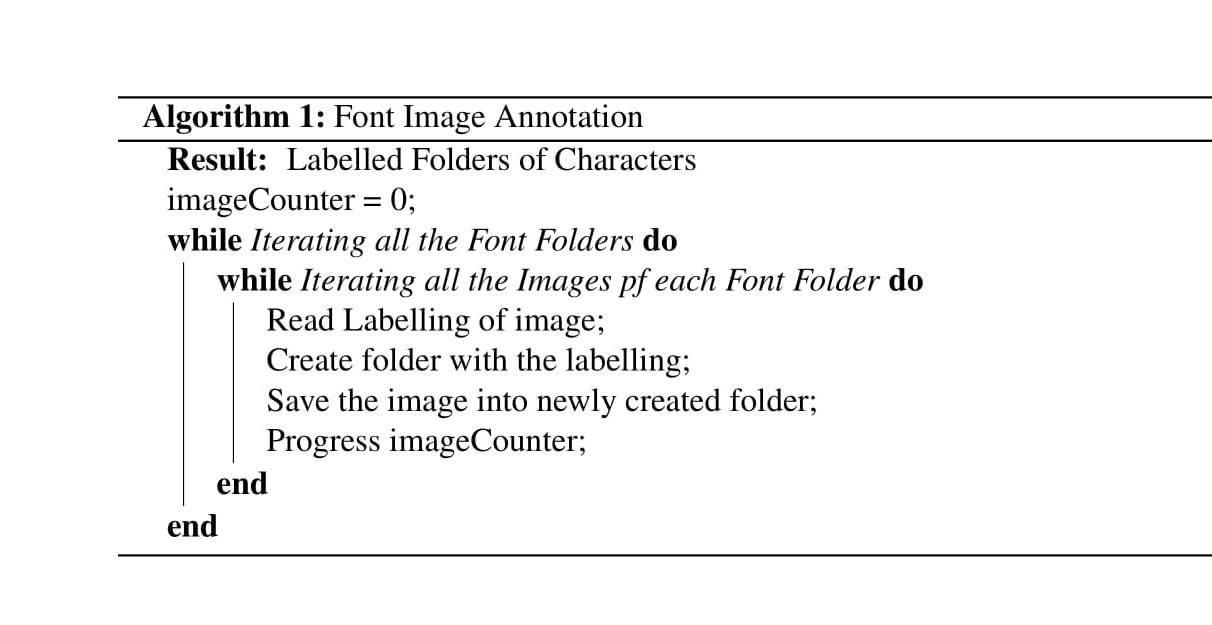
* 1. **Font Image Annotation**

Font image annotation refers to the process of data labeling. In the Image extraction process, we built a folder architecture with the font names. Inside the font folders, we had our character images labeled with a unique identifier defined in the metadata list. As seen in Fig. 7, for example, the first folder name is “AnandaMJ” which is a name of the font. Inside the folder, the images are labelled with a numeric value which is defined in the metadata descriptor.

|  |  |
| --- | --- |
|  |  |

**Fig. 7.**  Before Annotation

We then changed the structure with specific class names which was a numeric value. We iterate through all the folders with font names, and images inside these folders are written to a newly created folder with the corresponding image labeling. Then we had the labeling of the image folder corresponding to its unique classification. Inside this folder, we had all the different font images. The pseudocode for this algorithm is presented in Fig. 8.



**Fig. 8.** Pseudocode of labeling dataset

As seen in Fig. 9, the image represents the final structure of our dataset. Here all the folders seen in the first image are named with the numeric values which represent unique characters. Characters, basic and compound, all remain in the corresponding folder with all the variations of fonts, noise, and augmentation (described in detail in later part of this paper). The right-side image represents one highlighted folder “11” which corresponds to the character “অ.” Inside the folder “11,” there are all variations of “ অ” for this image.

The final folder structure of our corpus consists of 70 alphanumeric characters and 159 compound characters. Inside this folder, we have all the font images of the corresponding characters.

* 1. **Image Augmentation**

Augmentation is often used to increase the variability in the data within the chosen domain, which is a widely used approach in the development process of a corpus. We also adopted this approach. The two techniques that we have used for image augmentation are: (a) varying the slant of the character images and (b) introducing various types of noise.

|  |  |
| --- | --- |
|  |  |

**Fig. 9.** After annotation

### **Augmentation by varying the slant.** In real data, the images of the character have varieties in terms of the slant. Therefore, we used a +30 degree to a -30 degree variations of the slant of each image. Fig. 10 shows how this process has resulted in variations of a given character for a specific font class.



**Fig. 10.** Samples of characters generated after applying slat augmentation

### **Augmentation with Noise.** We have adopted a number of approaches of noise addition in our corpus.

### Dilation: Dilation is an operation of morphological transformation. In the dilation process, a pixel element is ‘1’ if at least one pixel under the ‘kernel’ is ‘1.’ So it increases the white region in the image or size of the foreground object increases. Normally, in noise removal techniques, erosion is followed by dilation. This way the noise is removed, but our object area does not shrink. It is also useful in joining broken parts of an object. In this process the output will depend upon the kernel size. We used a 2x2 kernel for dilation. Fig. 11 shows an image before and after dilation.



**Fig. 11.** A sample image before (left) and after (right) dilation.

Salt-and-Pepper. Salt-and-pepper noise is a common form of noise sometimes also known as ‘impulse’ noise. Sharp and sudden disturbances of the image signal can create such noise. The noise presents itself in image as sparsely occurring white and black pixels. Fig. 12 shows an image before and after addition of salt-and-pepper noise.



**Fig. 12.** A sample image before (left) and after (right) addition of salt and pepper noise

Blur. The image is usually convolved with a normalized box filter to add Blur noise in a particular image. Firstly, It calculates the average of all the pixels under the kernel area. Then the central element is replaced with the computed average. In our processing pipeline, we achieved this by the function cv2.blur() [17]. Fig. 13 shows an image before and after blurring.



**Fig. 13.** A sample image before (left) and after (right) blurring

Poisson noise. The poisson noises are basically related to random fluctuation of photons, which in turn can be spatial and temporal randomness. This noise is also called ‘quantum noise’ or ‘shot noise.’ The noise can be introduced by manipulating the value of peak. We used the Poisson method of random class of Numpy library to generate Poisson noise [18]. In our implementation, the value of peak was 3. Fig. 14 shows an image before and after addition of Poisson noise.



**Fig. 14.** A sample image before (left) and after (right) addition of Poisson noise

Erosion. Erosion is almost like soil erosion, the differentiating factor here is that it erodes boundaries of foreground objects while always trying to keep the foreground white. Firestly, a kernel is defined which slides through the image as in 2D convolution [19]. Original image pixel which is basically 1 or 0 is considered 1 only if all the pixels under the kernel is 1, otherwise it is set to zero. So all the pixels near the boundary area are discarded depending upon the defined kernel size. As a result, white region decreases in the image. It is widely used for removing small white noises, or disconnecting two attached components. We used a 3x3 kernel for erosion. Fig. 15 shows an image before and after erosion.



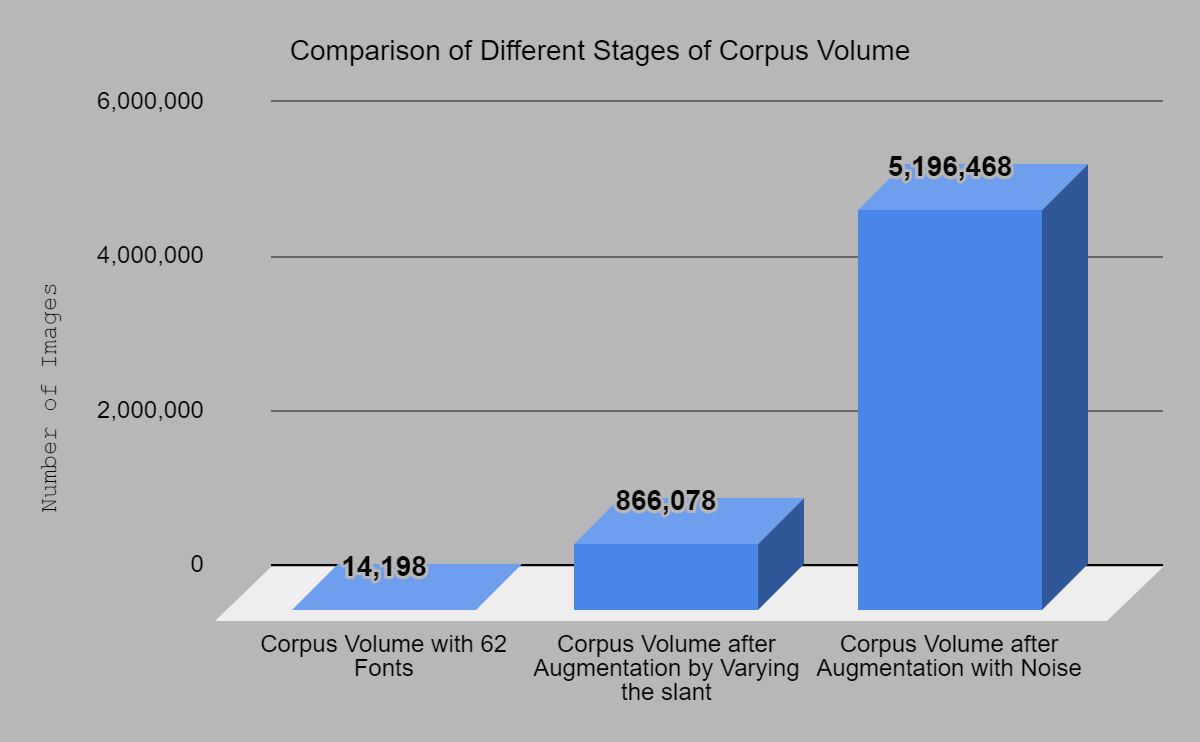
**Fig. 15.** A sample image before (left) and after (right) erosion

**4 Corpus Overview**

In our corpus we have 62 different font files: 45 of them are ASCII fonts and 17 of them are of Unicode Fonts. Therefore, each character has 62 different base forms in our Dataset. As described earlier, we have 70 basic characters denoting classes from 1 to 70 and 159 compound characters, which are denoted by 81 to 239 numeric values as class names. After having the basic dataset formed, we used augmentation to introduce variability to our corpus. We converted each image into 60 different images by varying the degree of the slant from -30 degree to +30 degree. This augmentation means that our corpus has 62 classes, and each class contains 3,782 images (62 class by 61 slant angles). Therefore our corpus contains 866,078 base images. Then we added different kinds of noise which are widely used in image processing. Each of these 3,782 images were transformed by adding 5 types of noises, which are Dilation, Salt-and-Pepper, Blur, Poisson Noise, and Erosion Noise. Every individual noise creates a separate copy of 3,782 images. This adds 18,910 images (3,782x5) in each class. Therefore, after noise addition, there are 22,692 images in each class. We have 229 classes. Altogether, the overall corpus has 5,196,468 images, which is roughly over 5 million images.

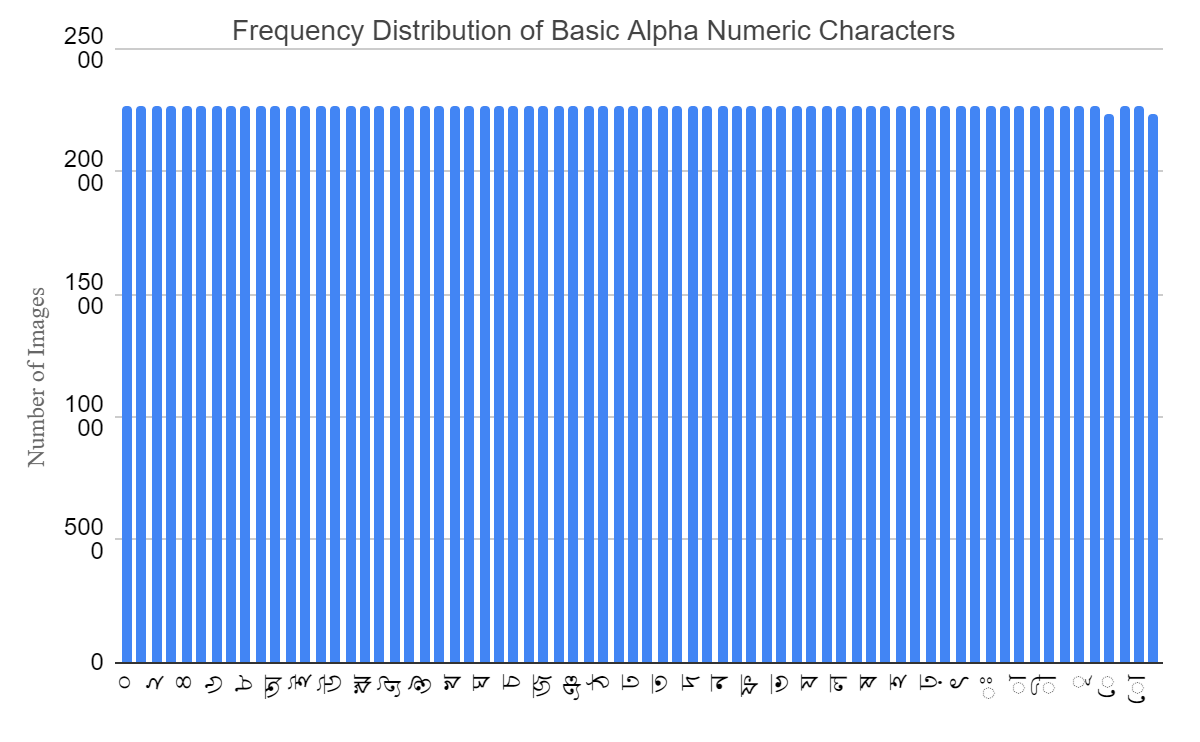
**5 Corpus Properties**

In this section, we will analyze different aspects of our corpus. This will include corpus volume and frequency distributions of different types of character classes.

****

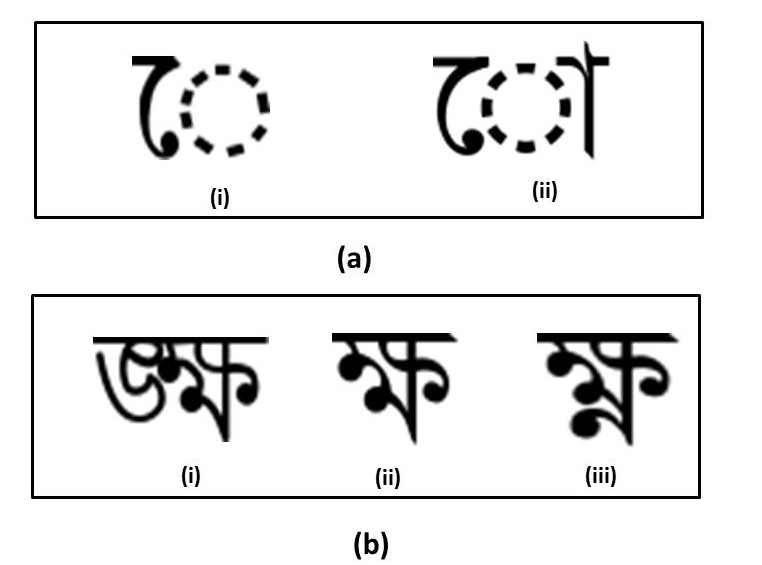
**Fig. 16.** Comparison of different stages of Corpus Volume

As this is a synthetic corpus, we increased our corpus size by adding a number of variations. Fig. 16 shows how our corpus volume changed with different variations. With 62 different fonts and 229 classes, we had 14,198 basic images in total. Then, all the images are augmented with slant variance. This transforms our corpus into 866,078 images. Next, we took this corpus and added 5 different types of noises. That leads to the final size for our copus, which is 5,196,468 images.

****

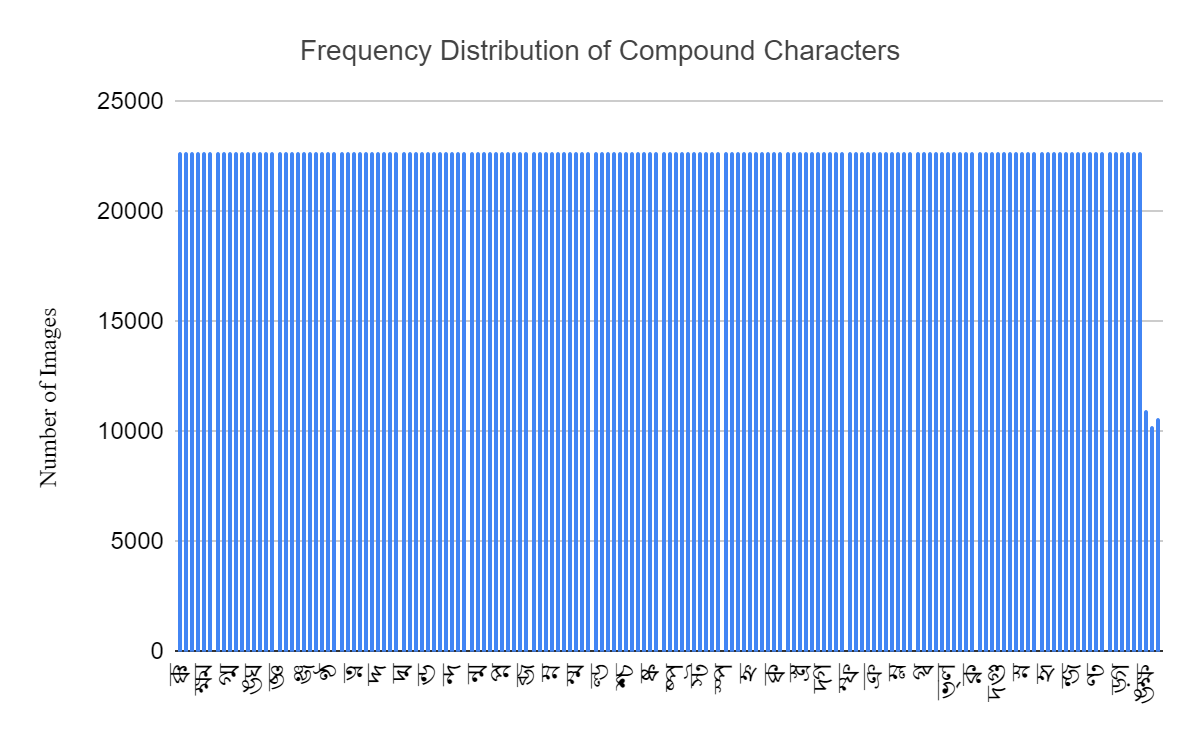
**Fig. 17.** Frequency distribution of Basic Alphanumeric Characters

Fig. 17 shows the number of images each alphanumeric character has in our corpus. In most of the cases, the number is evenly distributed among the characters. Still, we have some exceptions.



**Fig. 18.** Exceptions in frequency distribution

For vowel modifiers seen in part a of Figure 18, we have 22,326 images, where all other characters have 22,692 images. For these two modifier characters, some of the fonts have different rendering forms, which led our segmentation function to sometimes fail to detect that image.

****

**Fig. 19.** Frequency distribution of Basic Compound Characters

In case of the compound characters, most of the characters have the same number of images in our corpus as seen in Fig 19. However, for three of the characters seen in part b of Figure 18, we found some ASCII fonts are not compatible and our process failed to render images for these specific characters. As a result, we have found fewer images compared to other compound characters.

**6 Discussion**

In Bangla, there exists no printed character corpus that can be used for character recognition tasks. To fill in this gap, we have taken the initiative to build a synthetic corpus. Our corpus includes samples for all the possible Bangla characters for 62 different ASCII and Unicode fonts, where 45 are of ASCII and the rest 17 are of Unicode encoding. It is our hope that this corpus will help Bangla OCR research community to enhance their research outcomes.

There are some observations worthy of mention here. The first one is the issue of diacritics. In Bangla, a vowel or consonant can be replaced with a shorthand notation depending on where that character occurs within a word context. Not only that these notations can occur on top, bottom, left or right of another character, they can also occur in conjunction to compound characters, which in turn are composed of multiple characters. Our current corpus has not collected these notations. Secondly, in a serious OCR solution for Bangla, we have to address the issue of graphemes, not just isolated characters. Because of the complexities introduced related to the use of diacritics and compound characters, the task of segmentation becomes very complex. We have not included these graphemes in our corpus. Thirdly, we have also not considered the punctuation marks in Bangla, primarily because without the exception of (‘.’ - a period), all are identical to the punctuation marks in English. But this is a shortcoming that we need to address nonetheless. Fourthly, with the proliferation of online publications and sites, newer fonts are continually being introduced in Bangla, and we only were able to collect 62 fonts so far. There is one final issue related to the development of corpus that has not been addressed in this paper at all, which is the issue of building a corpus that is representative of the language usage, most commonly referred to as frequency distribution of the character classes. This was not possible at the time of building this corpus because there is still no serious Bangla language model in existence. However, in a parallel work, we are working on building a Bangla language model, which, once completed, will be useful in augmenting the make up of this corpus.

In the future iterations of this corpus, we plan to address all these five issues.

**6 Conclusion**

In this paper, we built a novel synthetic corpus of printed Bangla characters containing vowels, consonants and compound characters of both Unicode and ASCII font classes. This is a ‘first-of-its-kind’ work, since there exists no such corpus currently for the Bangla language. As the corpus contains over 5 million images, this should be an adequate starting point for training and validating various machine learning algorithms and help design and validate effective Bangla printed character OCRs.

**7 Acknowledgement**

The authors would like to acknowledge the encouragement and funding from the “Enhancement of Bangla Language in ICT through Research & Development (EBLICT)” project, under the Ministry of ICT, the Government of Bangladesh.

**References**

1. Biswas, M., Islam, R., Shom, G.K., Shopon, M., Mohammed, N., Momen, S. and Abedin, A.,: Banglalekha-isolated: A multi-purpose comprehensive dataset of handwritten bangla isolated characters. Data in brief, 12, pp.103-107, (2017).
2. Alam, S., Reasat, T., Doha, R.M. and Humayun, A.I.: NumtaDB-Assembled Bengali Handwritten Digits. arXiv preprint arXiv:1806.02452, (2018).
3. Rabby, A.S.A., Haque, S., Islam, M.S., Abujar, S. and Hossain, S.A.,: December. Ekush: A Multipurpose and Multitype Comprehensive Database for Online Off-Line Bangla Handwritten Characters. In International Conference on Recent Trends in Image Processing and Pattern Recognition (pp. 149-158). Springer, Singapore, (2018).
4. Bengali.AI Handwritten Grapheme Classification, [https://www.kaggle.com/c/bengaliai-cv19/dat](https://www.kaggle.com/c/bengaliai-cv19/data)a, last accessed 2020/03/15.
5. Ferdous, J., Karmaker, S., Rabby, A. S. A., and Hossain, S. A. (2021). MatriVasha: A multipurpose comprehensive database for Bangla handwritten compound characters. In: Hassanien A. E., Bhattacharyya S., Chakrabati S., Bhattacharya A., Dutta S. (eds) Emerging Technologies in Data Mining and Information Security. Advances in Intelligent Systems and Computing. Springer, Singapore
6. Off-line Handwritten Bangla Numeral Database,, <https://www.isical.ac.in/~ujjwal/download/BanglaNumeral.html>, last accessed 2020/4/25
7. Sayem, A.: Speech analysis for alphabets in Bangla language: automatic speech recognition (2014).
8. Rahman, M. and Kumar Dey, E.,: Datasets for aspect-based sentiment analysis in bangla and its baseline evaluation. Data, 3(2), p.15, ( 2018).
9. Hossain, M.Y., Hossain, I., Banik, M., Hossain, M.I.A. and Chakrabarty, A.,: Embedded System based Bangla Intelligent Social Virtual Robot with Sentiment Analysis. In 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR) (pp. 322-327), IEEE (2018).
10. Gaylord, H.,: Character representation. Computers and the Humanities, 29(1), pp.51-73, (1995).
11. Wasserkrug, S., Dalvi, N., Munson, E., Gogolla, M., Sirangelo, C., Fischer-Hübner, S., Ives, Z., Velegrakis, Y., Bevan, N., Jensen, C. and Snodgrass, R.,: Unicode. Encyclopedia of Database Systems, pp.3231-3232, (2009).
12. Bijoy Bayanno <https://bijoybayanno.info/> last accessed on 2020/6/1.
13. OpenCV Documentation, <https://docs.opencv.org/master/>, last accessed on 2020/3/21.
14. PDF to Image Converter, <https://pdftoimage.com/>, last accessed on 2020/3/23.
15. Microsoft Word Document, <https://www.microsoft.com/en-us/microsoft-365/word>, last accessed 2020/3/19.
16. Python Imaging Library, <https://pillow.readthedocs.io/en/stable/>, last accessed 2020/3/29.
17. Smoothing Images, <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html>, last accessed on 2020/4/9.
18. Python Scientific Computing Library, <https://numpy.org/>, last accessed 2020/4/13.
19. Morphological Transformations, <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html>, last accessed 2020/4/18.