International Journal on Engineering Technology and Infrastructure Development (InJET-InDev)
www.lec.edu.np/journal
Volume 2, Issue no.1, Page 81-105
Received Date: 8/30/2024
Revised Date: 6/2/2025
Accepted Date: 6/20/2025
ISSN: 3021-999X (Print)

Digitization of Devanagari Handwritten Text Using CNN

Bishwash Gurung[1], Manish Thapa[2], Ram Shrestha[3], Regan Sthapit[4]
[1]*Thapathali Engineering Campus, Thapathali, Kathmandu Nepal, bishwash.752419@thc.tu.edu.np*
[2]*Thapathali Engineering Campus, Thapathali, Kathmandu Nepal, manish.752419@thc.tu.edu.np*
[3]*Thapathali Engineering Campus, Thapathali, Kathmandu Nepal, ram.752419@thc.tu.edu.np*
[4]*Thapathali Engineering Campus, Thapathali, Kathmandu Nepal, regan.752419@thc.tu.edu.np*

Abstract

The Devanagari script used in Nepali, Hindi, Sanskrit, and other languages includes many more characters and modifiers, which makes the task of recognizing handwritten text very difficult. This paper outlines a system for recognizing handwritten Devanagari texts through the use of CNNs. The process begins with the generation of a large data set comprising different types of handwriting. The data is normalized, resized, and filtered to get rid of noise and all the unwanted features in the images to ensure that data quality is as high as possible. To allow for supervised learning, each of the images is associated with the corresponding character. A CNN model is then created and trained on this labeled dataset, which includes convolutional, pooling, and fully connected layers to capture the features of the Devanagari characters. The given model is evaluated using numerous evaluation measures. As per these evaluations, optimization is carried out to improve the precision and dependability of such a system. The refined model is then used to predict the new, unseen handwritten samples to check the generality of the model. Following validation, the model is subjected to more practical applications, and people are shown how the model can be applied in real-life scenarios. This work demonstrates the potential of CNNs to improve HTR technologies and contributes to the process of digitization for the Devanagari script.

Keywords: Handwriting Recognition, Devanagari Script, Convolutional Neural Network, Machine Learning, Digitization, Pattern Recognition

1. Introduction

Devanagari Handwritten Text Recognition (HTR) is the method of recognizing Handwritten characters of scripts used in Hindi, Sanskrit, Marathi, Nepali many more languages. This work focuses particularly on the Devanagari script as used in the Nepali language, although the approach can be generalized to other languages utilizing the script. The Devanagari script is one of the most widespread in South Asia, and the identification and, especially, the digitization of such a script produces significant implications in numerous fields. Devanagari HTR has applications in the banking, government, and education sectors, among others. For instance, in banking, any form that can be filled and written by hand such as checks, any forms that can be filled by hand can be scanned and taken through the system, hence saving time that would have otherwise been used to key in the data. In the government context, the digitization of historical documents in the Devanagari script could be beneficial for the purposes of preserving history and making such documents more accessible to the public. In education, HTR systems also offer a way of providing efficient and accurate grading of handwritten exams and assignments.

The project proposes the use of Convolutional Neural Networks (CNNs) as an effective approach to overcome challenges in HTR. Unlike conventional methods, which rely on handcrafted characteristics and rule-based divisions, CNN offers a way that can automatically grasp the required features from the image. This makes CNN suited for handling the complexities found in Devanagari handwritten scripts. While CNN has been explored in character recognition tasks before, this work focuses on creating an architecture that is tuned for the challenges of handwritten Devanagari with an emphasis on Nepali script recognition which has been less studied in comparison to Hindi.

1.1 Challenges in Devanagari Handwritten Text Recognition

However, there are several issues in the use of Devanagari HTR because the script in question is highly complicated as well. Some of the difficulties that one is likely to encounter include the presence of modifiers as well as compound characters. The Devanagari script has different types of marks, ligatures, and characters created by combining two or more basic elements, and hence it is challenging to segment and recognize the characters correctly. These modifiers change the form and location of the base characters, and this makes it difficult to recognize them.

Another problem that can be associated with the given set of symbols is the fact that some of them have similar shapes. Some of the Devanagari characters have similar structural characteristics, and this confuses the recognition system. This shape similarity puts a lot of pressure on the HTR system to accurately extract features and classify between such characters.

The variability in handwriting between individuals and styles adds a layer of complexity to recognition. It is therefore clear that handwriting may differ from one writer to another and also may change from one instance to another depending on the writing implement used, the speed at which the writer is writing, and the general writing environment. This is a big problem for a recognition system to sustain high accuracy on different samples.

Another problem that has an impact on recognition accuracy is the placement of characters. Handwritten text is not always aligned properly, and the spaces between the lines may also not be equal to those in the printed text. Characters can be written in any position on the paper, and the space between the characters, as well as between the words, can also be different, which makes it difficult to segment and recognize the characters.

1.2 Handwriting Recognition Systems and CNNs

To overcome these challenges, the project employs Convolutional Neural Networks (CNNs) as the main model for identifying the handwritten Devanagari script with a specific focus on the Nepali variant. CNNs belong to the category of deep learning models, which are widely utilized for image recognition tasks. It can also acquire high-level characteristics from raw image data and, therefore, can handle the script intricacies of the Devanagari script. The CNN model can automatically discover the attributes of the input image, which helps in avoiding the feature engineering process.

In the case of Devanagari HTR, while CNNs have demonstrated high recognition accuracy of characters in general, this project emphasizes developing a custom CNN architecture fine-tuned for Nepali script, the presence of a large and diverse training dataset is desirable. The project's strategy is to create a strong CNN architecture with a major emphasis placed on character segmentation, and feed it with a large set of labeled handwritten Devanagari characters. The system is developed to solve the problems of character similarity, handwriting differences, and the position of characters.

2. Literature Review

2.1 Segmentation

Segmentation of textual content at the character level is a crucial preprocessing method for optical character recognition (OCR). The challenge of dividing text is amplified in languages with a cursive writing form. Handling the inherent variation in the writing styles of many people presents the fundamental problem in handwritten character segmentation.

Various segmentation approaches have been employed in the OCR system for segmentation and classification. The approaches are:

1. The conventional method, where segments are chosen based on "character-like" characteristics. "Dissection" refers to separating the image into relevant sections.

2. Recognition-based segmentation: Kompalli, Suryaprakash, et al**.** [1] described a method that used graph representation to segment individual characters. With the help of this technique, we divide characters that overlap horizontally or vertically, those linked with non-linear borders, and others into smaller, more primal components.

3. Holistic Technique: B. Shaw, S. K. Parui, et al. [2] proposed a holistic Devanagari recognition method incorporating a hidden Markov model (HMM), where a sliding window scans the image-strip histogram of chain-code directions. Recognition is achieved by building an HMM for each word, determining the conditional probability for each HMM, and opting for the class with the highest likelihood.

2.2 Recognition

Different techniques of recognition are used for different segmentation techniques. Rao, N. Venkata, et al. [3] described different approaches to recognition, including Fuzzy logic, Structural analysis, Matrix Matching, Feature Extraction, and Neural Networks.

Pant and Nirajan [4] utilized a Random Forest algorithm combining Holistic methods and character-level recognition for printed Nepali text, achieving a 78.87% accuracy rate for character-level recognition and 94.80% for the hybrid approach. Puri, Shalini, et al. [5] developed an effective Devanagari character categorization model using SVM, which preprocesses images, segments them using projection profiles, and categorizes characters after removing Shiro Rekha. The model achieved a mean classification performance of 99.54% for printed images and 98.35% for handwritten images, outperforming existing methods. Pande, Dwarka Nath, Sandeep, et al. [6] developed a CNN-based system for recognizing Devanagari handwritten text using the DHCD dataset, which contains 46 classes with 2,000 images each. The system includes a conflict resolution technique and shows promising results in accuracy and training duration. Sayyad, Jadhav, et al. [7] utilized a multilayer perceptron with a hidden layer to recognize Devanagari characters, creating character patterns in a matrix (n*n) and storing them in binary form. The backpropagation neural network was used for effective recognition, with rectified neuron values delivered through the feed-forward method, and Python-supported Devanagari string processing. The Nisaba Brahmic library [8] supports critical Brahmic script functions like NFC, reversible transliteration, visual normalization, and validity checks inside a modular and extendable framework.

2.3 Related works

Shakya, Shanti, et al. [9] provided an overview of the Nepali OCR system, detailing five key methods: Preprocessing, Segmentation, Feature Extraction, Training, and Recognition. They reported achieving approximately 90% accuracy for segmentation and 80% for recognition. Ghimire, Ashutosh, et al. [10] developed a Nepali character recognition system using template matching in machine learning, specifically implementing the Back Propagation and Gradient Descent algorithms. The project utilized datasets collected from the Computer Vision Research Group, Nepal. Dawadi, Babu Ram, et al. [11] presented a solution for Nepali text recognition, exploring the pros and cons of segmentation techniques. They developed the system using a feed-forward backpropagation Artificial Neural Network (ANN). Prajapati, Sudan, et al. [12] analyzed the efficiency of Nepali script character recognition using the Tesseract engine and Artificial Neural Network, achieving 96% accuracy in training and 69% in testing with Tesseract. The ANN achieved 98% accuracy in training and 81% in testing. Ingroj [13] explains how to generate Unicode for characters in Nepali represented using the Devanagari script. With code points between u0900 and u097F. Nepali texts are encoded using UTF-8, a widely used character encoding that represents each character with an 8-bit block. S. Bag et al. [14] have developed a character segmentation method for handwritten Hindi words that uses structural patterns to address changes in writing approach and slanted header text. Tested on both handwritten and printed words, this method achieves an average success rate of 96.93% on their database. OCR for languages using the Devanagari script has previously undergone extensive investigation, with Hindi in particular claiming to have a performance accuracy rate of up to 93% at the character level. [15]

2.4 OCR tools developed for Devanagari

Numerous businesses and individuals in India and Nepal have started the creation of Devanagari OCR software (for Sanskrit, Hindi, Marathi, and Nepali). Chitrankan [16] a Hindi and Marathi OCR system, has been created by C-DAC in India. Based on the HTK tool and the Tesseract Open Source OCR engine [17] Nepalese Madan Puraskar Pustakalaya (MPP) has also created OCR initiatives for the Nepali language. Dr. Oliver Hellwing established Ind. Senz [18] is creating OCR software for languages written in the Sanskrit, Hindi, and Marathi scripts.

3. Methodology

3.1 Block Diagram

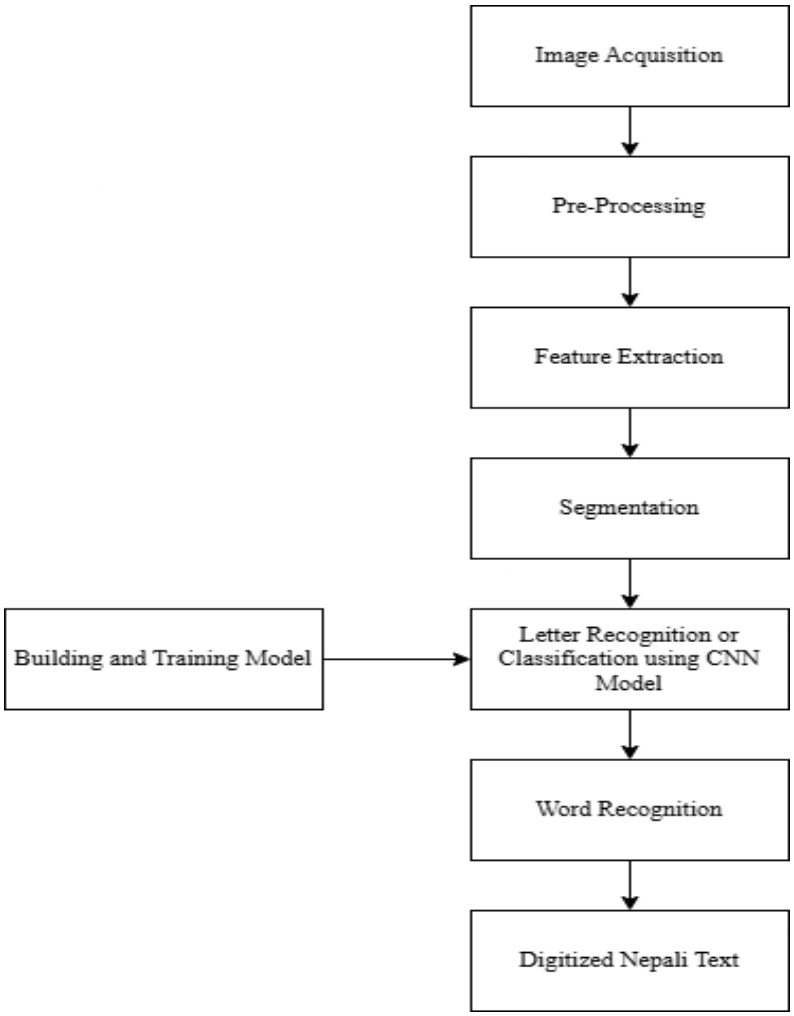


Figure 1: Block Diagram

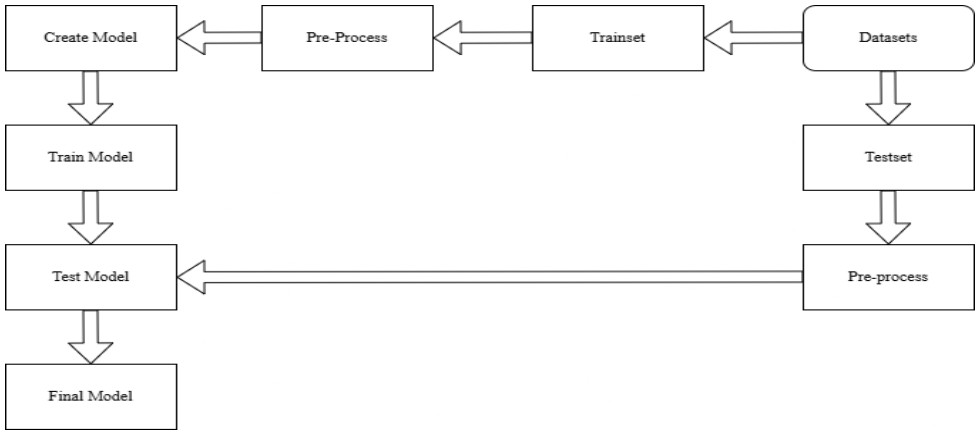


Figure 2: Model Training

3.2 Dataset Collection

The dataset used for analyzing the main characters was sourced from Kaggle [19]. It includes approximately 200 images per character, providing a robust foundation for character-based analysis.

For a collection of datasets for the modifiers in the Devanagari script, we had one primary approach: collecting handwritten data from school students and family members. In this approach, we requested many students and our family members we distribute the leaflet for writing the alphabet, and we primarily collected the handwriting of persons. Thus, the obtained data was captured by the mobile phone's camera, scanned, and cropped for the individual characters. The foreground of the images is white, and the backgrounds are black. Each image is 32 * 32 pixels.

3.3 Segmentation Process:

The picture is segmented into words as the next significant step following feature extraction. The character segmentation is carried out in the next stage by splitting the input into three zones: the top, middle, and lower zones. It involves the following steps:

1. Image pre-processing

Image pre-processing involves enhancing image quality by reducing distortions and emphasizing relevant features. This process includes cropping the image to focus on the text, converting it to grayscale, and organizing pixels into a NumPy array. We then apply thresholding to filter out noise and identify significant text regions by counting pixels with non-background colors and determining the top, bottom, left, and right boundaries of the text by scanning the image.

2. Resizing the image

To ensure accuracy, images must be cropped to square shapes and resized while preserving their aspect ratio. This involves calculating the original aspect ratio and adjusting dimensions accordingly to prevent distortion and retain important features.

3. Header line (Shiro Rekha) detection and removal:

To identify the header line of a word, we analyze horizontal pixel density and focus on the top 55% of the word to find the highest density area, which indicates the header line. This line, spanning the full word in handwritten text, is then removed by adjusting the gray level of the line and adjacent rows.

Figure 3: Word with header line

Figure 4: Word with header line removed

4. Character-level segmentation

After removing the header line, we scan each column to identify those that match the background, as they are the spaces between letters, and use a defined threshold to distinguish characters. Columns exceeding this space threshold are used to split images to create individual character images.

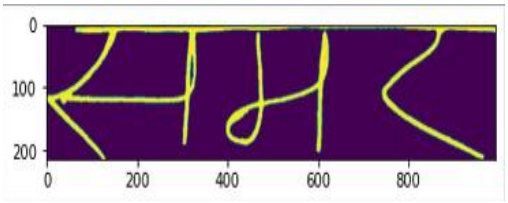


Figure 5: Non-segmented word

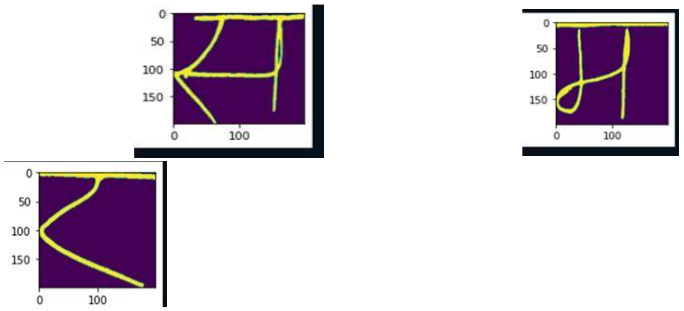
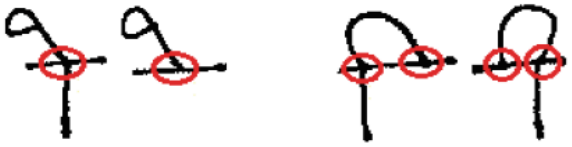


Figure 6: Segmented character

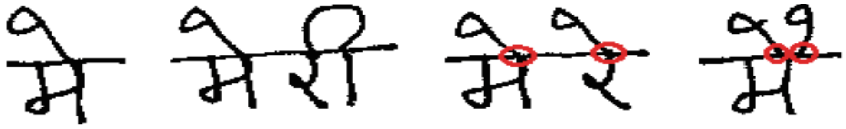
5. Segmentation of upper modifiers

To segment upper modifiers in Devanagari, follow these steps:

1: Identify whether the upper modifier touches the header line once or twice.

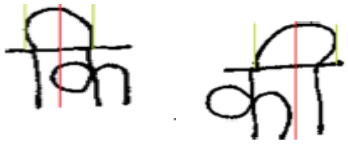


2: For two touching modifiers, check whether the distance between these two touching points exceeds a certain threshold, and if not, consider them as a single touching upper modifier and proceed accordingly.



3: For single touching, locate column a threshold distance from the touching point and connect it to components below if over a blank space, the upper modifier has a component in the middle zone and it needs to be attached with the upper modifier, else the upper modifier can be separated and does not need to be connected with the middle component.

4: For two touching as well, locate the column a threshold distance from the back touching point. Connect it to the components below if over a blank space, the upper modifier has a central component found behind the main letter, and it needs to be attached; else, the upper modifier has a middle zone which is ahead of the actual letter character, and it also needs to be attached accordingly.



6. Segmentation of lower modifiers

To detect lower modifiers in Devanagari, each element's lowest point is scanned, and the gap between the highest and lowest low points is compared against a threshold. If that difference exceeds that threshold, the average of the lowest points is calculated. If the element’s lowest point is below this average, that element has a lower modifier otherwise, it does not.

7. **Segmentation of half-form character**

To determine if a character is single or half-form, we first check if the image width is greater than its height, indicating a half-form character. We then scan column-wise below the header line, store pixel values in an array, and use the highest stored value to identify and split the image into a half-form character image and a full-form character image.

Some examples of segmentation:

Figure 7: Word containing half-form character

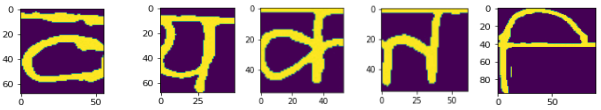


Figure 8: Segmentation of half-form characters

Figure 9: Word containing half-form, lower, and upper modifiers

Figure 10: Segmentation of each character

3.4 Algorithms

1. Cropping Algorithm

1. Define the threshold value as 9% of the total rows.
2. Texture with row value below this is considered noise.
3. For each row, on scanning from top to bottom
    a. Count the number of pixels
    b. Check for values other than background color.
4. If this value is greater than the threshold,
    a. The top part of the text is found.
5. For each row, on scanning from bottom to top,
    a. count the number of pixels
    b. Check for values other than background color.
6. If this value is greater than the threshold
    a. The bottom part of text is found.
7. Transpose the image.
8. Repeat steps ii. and iv. to find the left/right part.

2. Upper Modifier

1. Detect the header line row.
2. Scan the image from top
    a. To find the row with pixels > 0.85*(rightmost value of the cropped image.)
3. Separate the upper modifier by separating the image
    a. into an image above the header line and an image below it.

3. Middle Characters

1. Read the image with removed upper modifier.
2. Define space that separates the characters within a word.
3. Scan each row horizontally
    a. To determine column value, differentiate the background color.
4. If the difference between these columns > space
    a. Then append column values to the new list.
5. Segment the image using these column values.

4. Lower Modifier

1. From the segmented characters
    a. Determine the character with the lowest row value.
2. Define a threshold to check the presence of a lower modifier.
3. If the modifier is present
    a. Cropping the character image using the lowest row as a separator.

5. Algorithm for Word Combination of the Recognized Character

1. Define a list of characters and label them.
2. From the model, get the array with the recognized character as 1 and others as 0.
3. Append the characters to form a word.

3.5 Convolutional Neural Networks (CNNs):

CNNs are designed to identify complex data features through layers, including:

- Convolution Layer: Applies filters to create feature maps, detecting patterns in images.
- Max-Pooling Layer: Subsample feature maps to improve translation invariance.
- Dropout Layer: Avoids excessive fitting by randomly disabling neurons while training.
- Flatten Layer: Turns the feature maps of the previous layer from two-dimensional into one-dimensional
- Dense Layer: Composes all neurons from previous layers, often utilized in networks
- ReLU Layer: Activation function that outputs zero for negative values of the input and the input itself for positive values, enhancing the training process.

3.6 Adam Optimizer:

Momentum with RMSprop works together, where it helps in gradient descent by adjusting the learning rates, thus improving the model's convergence rate.

3.7 Categorical Cross Entropy:

Measures loss for multi-class categorization by comparing predicted probabilities with true labels.

3.8 Sparse Categorical Cross Entropy*:*

Identical to categorical cross-entropy but uses integer labels instead of one-hot encoding.

3.9 SoftMax Function:

Converts logits into probabilities for multi-class classification.

3.10 Word Recognition:

We use a Convolutional Neural Network (CNN) to classify letters and digits, leveraging its ability to handle complex images with pixel dependencies. The CNN applies various filters to detect patterns and their interconnections, with the convolution layer reducing pixel count while preserving crucial features for accurate predictions. The Pooling layer further condenses spatial dimensions, reducing computational requirements and enhancing model training by focusing on invariant features. The network processes multiple convolutional and pooling layers before flattening the output and passing it through dense layers for classification.

Our CNN architecture includes four models: Main Character, Half Character, Lower Modifier, and Upper Modifier. Each model is trained on a dataset of 60 different letter types, with approximately 2,000 images per letter. Characters are resized to 32x32 pixels for the CNN, and predictions are made based on high-confidence outputs, although segmentation errors may impact accuracy.

Table 1: Model Parameters

Parameter	Value
Filter Size	(3, 3)
Number of Filters	128 per Conv2D layer
Strides	1
Pooling	MaxPooling2D (2, 2), stride 2
Dropout rate	0.25 after Conv blocks, 0.2 after Dense layers
Dense layer Size	256, 128
Activation Function	ReLU (hidden layers), Softmax (output layer)
Input Shape	(32, 32, 1)
Batch Size	32
Optimizer	Adam (learning rate = 1e-3, decay = 1e-5)
Loss Function	Sparse_Categorical_Crossentropy
Epochs	30 (set during training)

4. Results

4.1 Text Recognition

We have made a simple web app using Streamlit, which takes images such as PNG, JPG, and JPEG, as input and utilizes the trained model to classify the uploaded image and digitize it. The system permits the user to upload the image and then click on the “Digitize the text” button to proceed. The uploaded image is cropped and preprocessed, and finally digitized text is displayed.

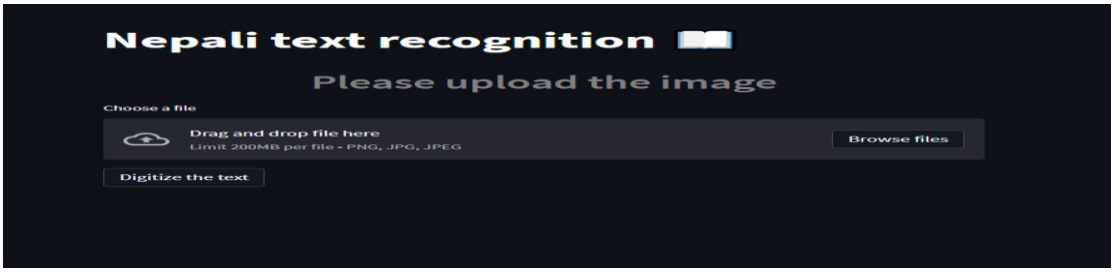


Figure 11: Home screen of user interface

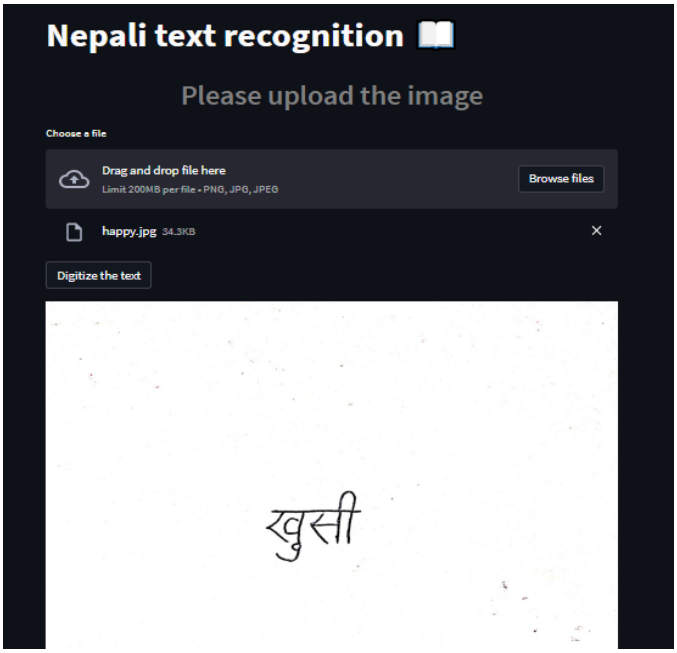


Fig 12: Image uploaded

Fig 13: Digitized result

Main character model -1

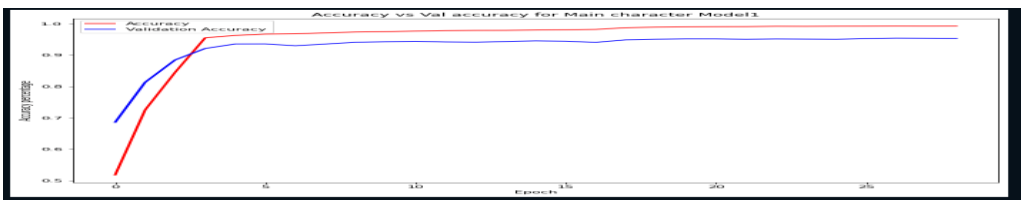


Figure 14: Training vs Validation Accuracy

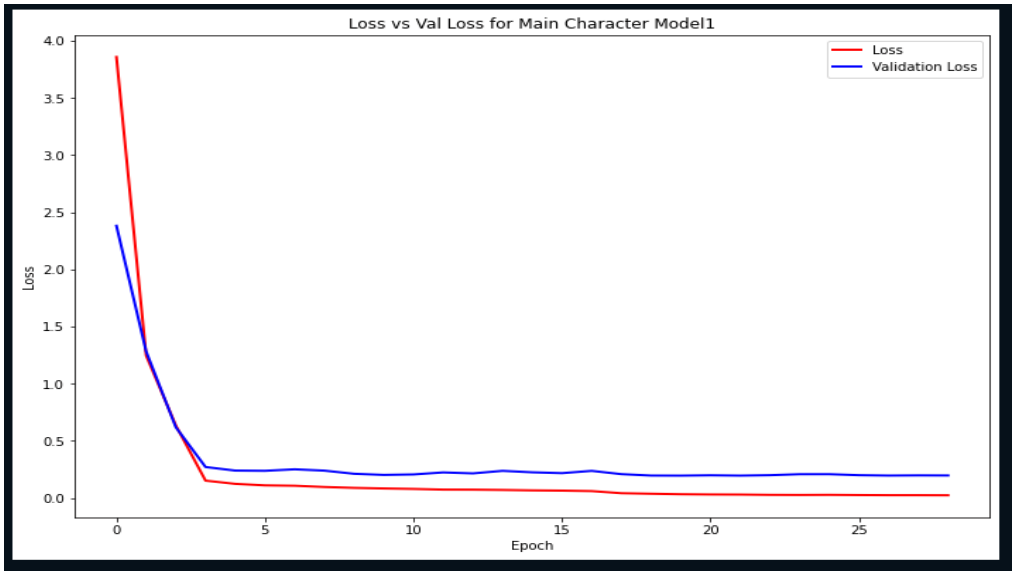


Figure 15: Training vs Validation loss

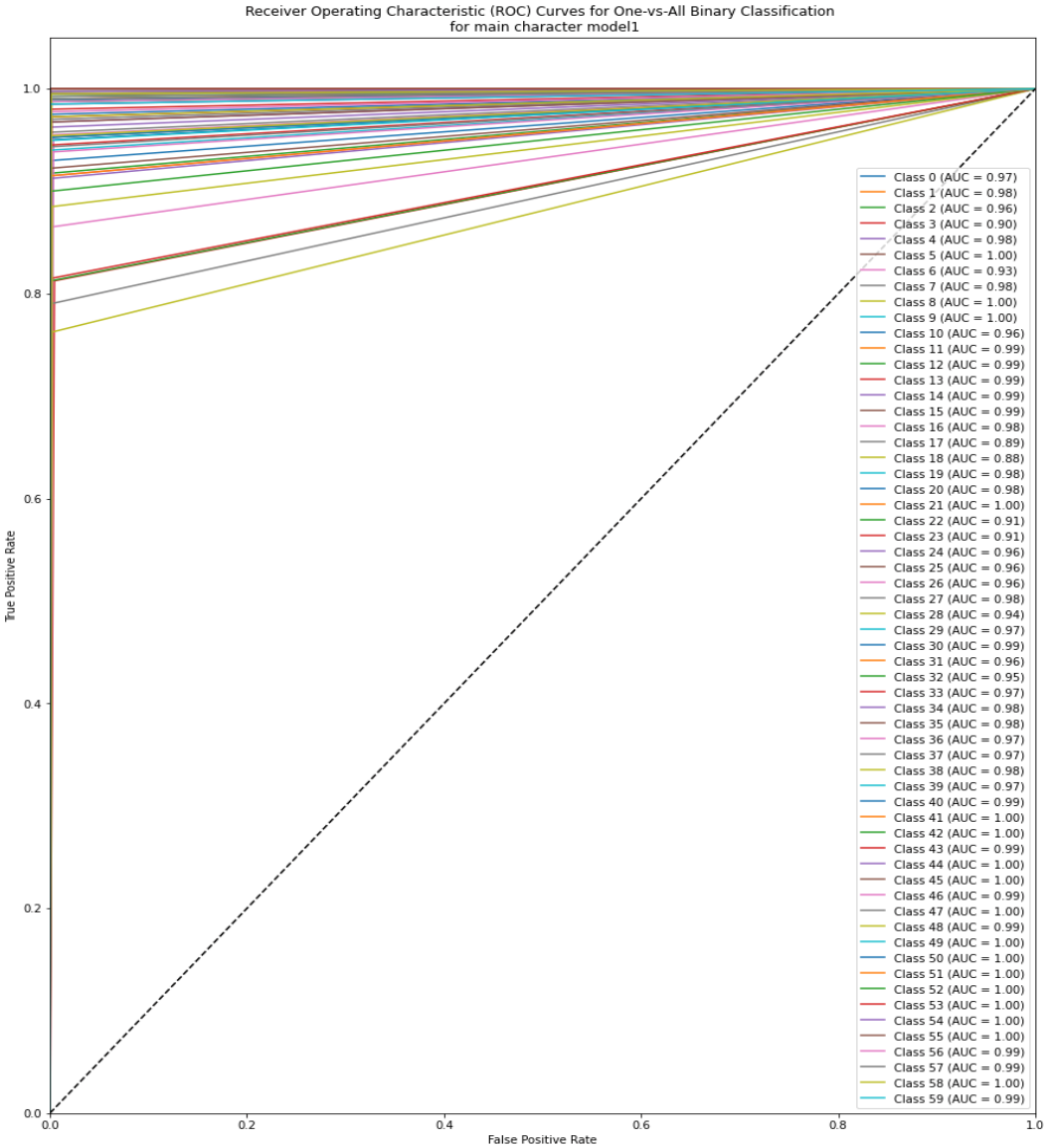


Figure 16: ROC Curve

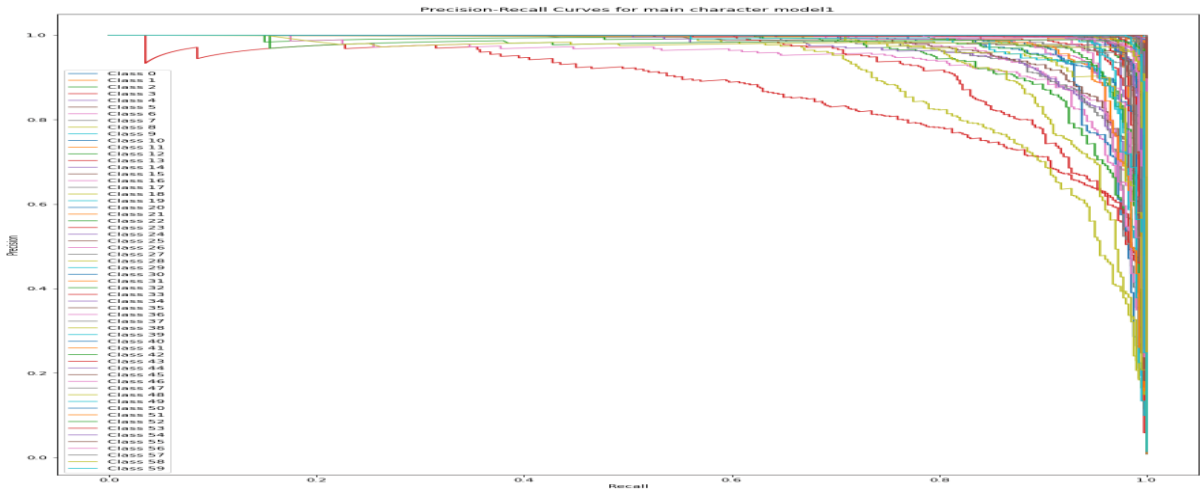


Figure 17: PR Curve

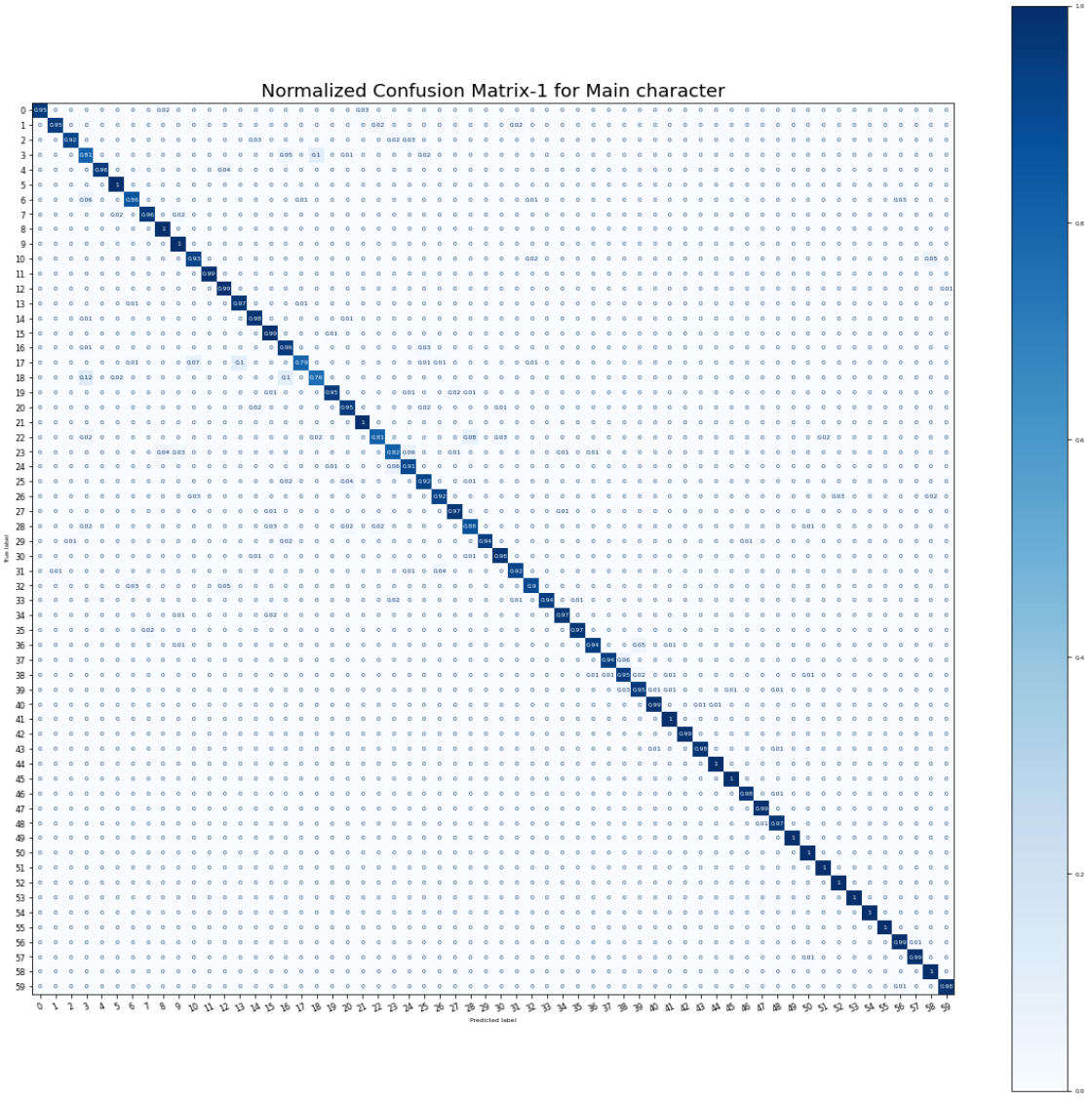


Figure 18: Normalized Confusion Matrix – Main Character Model -1

	precision	recall	f1-score	support
0	0.99	0.95	0.97	400
1	0.98	0.95	0.97	400
2	0.97	0.92	0.94	400
3	0.76	0.81	0.79	400
4	1.00	0.96	0.98	400
5	0.95	1.00	0.98	400
6	0.94	0.86	0.90	400
7	0.96	0.96	0.96	400
8	0.94	0.99	0.97	400
9	0.94	1.00	0.97	400
10	0.89	0.93	0.91	400
11	0.99	0.99	0.99	400
12	0.92	0.99	0.95	400
13	0.90	0.97	0.94	400
14	0.95	0.98	0.97	400
15	0.92	0.99	0.95	400
16	0.83	0.95	0.89	400
17	0.97	0.79	0.87	400
18	0.86	0.76	0.81	400
19	0.97	0.95	0.96	400
20	0.92	0.95	0.93	400
21	0.97	1.00	0.98	400
22	0.94	0.81	0.87	400
23	0.90	0.81	0.85	400
24	0.89	0.91	0.90	400
25	0.91	0.92	0.92	400
26	0.95	0.92	0.93	400
27	0.97	0.97	0.97	400
28	0.88	0.89	0.88	400
29	0.99	0.94	0.97	400
30	0.95	0.97	0.96	400
31	0.96	0.92	0.94	400
32	0.94	0.90	0.92	400
33	0.99	0.94	0.97	400
34	0.98	0.97	0.97	400
35	0.99	0.97	0.98	400
36	0.97	0.94	0.95	194
37	0.98	0.94	0.96	194
38	0.90	0.95	0.93	196
39	0.93	0.95	0.94	198
40	0.97	0.99	0.98	188
41	0.99	1.00	0.99	202
42	1.00	0.99	0.99	400
43	0.99	0.98	0.98	400
44	0.99	1.00	1.00	400
45	0.99	1.00	0.99	402
46	0.98	0.98	0.98	400
47	0.99	0.99	0.99	400
48	0.97	0.97	0.97	400
49	1.00	1.00	1.00	309
50	0.97	1.00	0.99	400
51	0.97	1.00	0.99	400
52	0.96	0.99	0.98	400
53	0.99	1.00	1.00	400
54	0.99	0.99	0.99	400
55	1.00	1.00	1.00	400
56	0.96	0.99	0.97	400
57	0.99	0.99	0.99	400
58	0.93	0.99	0.96	400
59	0.99	0.98	0.99	400
accuracy			0.95	22683
macro avg	0.95	0.95	0.95	22683
weighted avg	0.95	0.95	0.95	22683

Figure 19: Evaluation Metrics for Main Character Model – 1

Half Character Model - 2

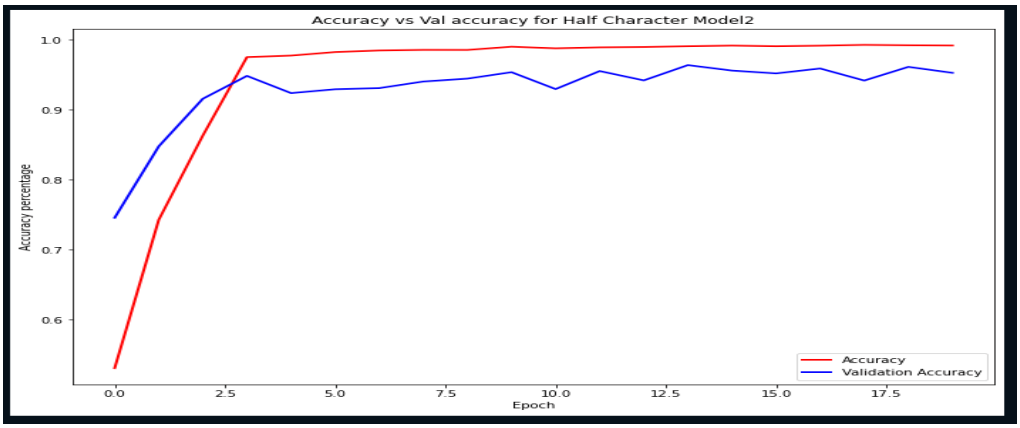


Figure 20: Training vs Validation Accuracy

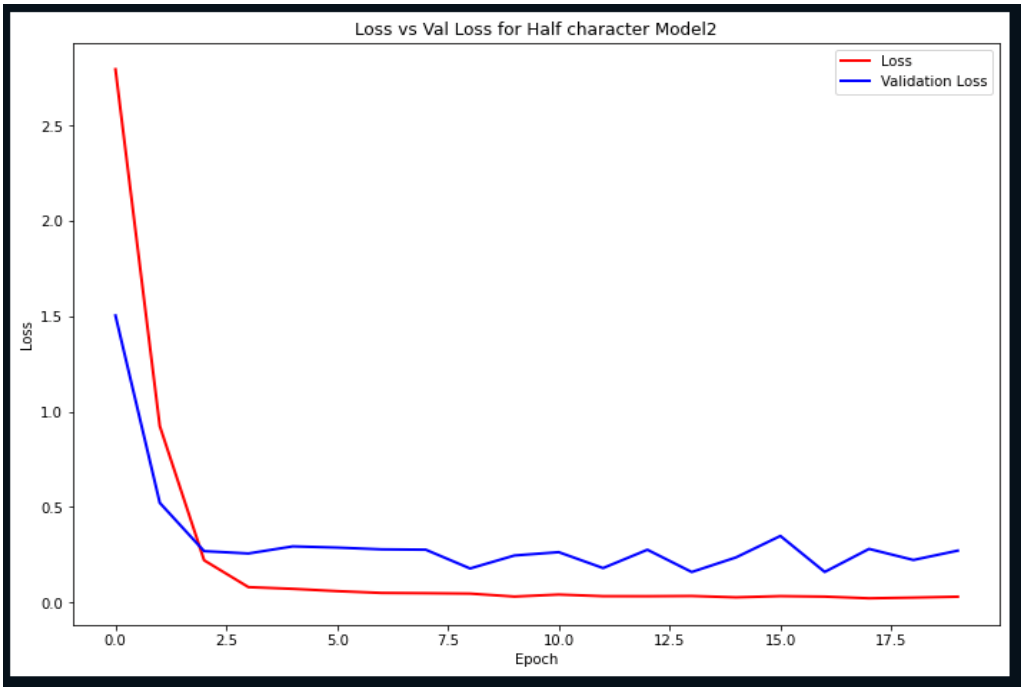


Figure 21: Training vs Validation Loss

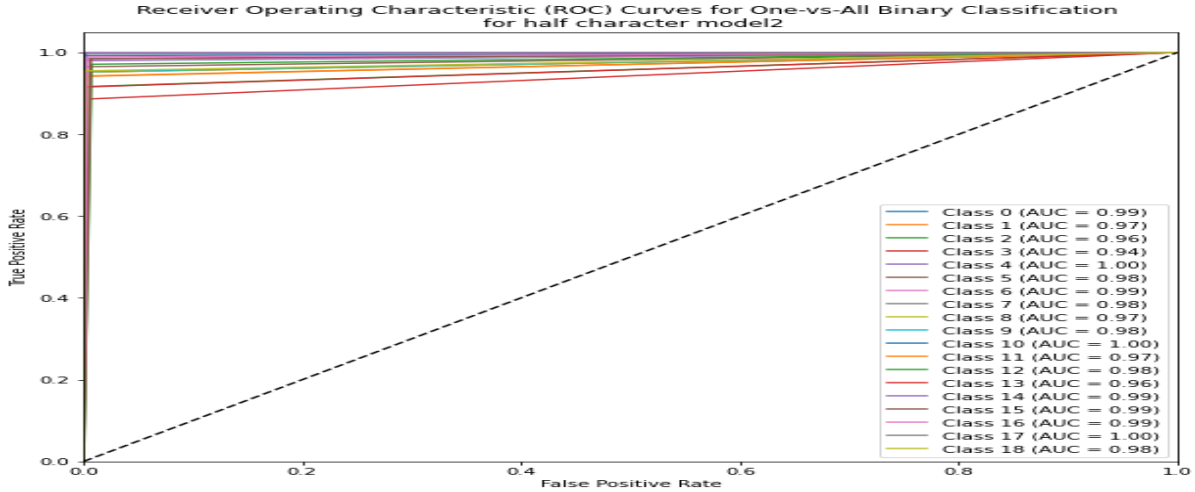


Figure 22: ROC curve

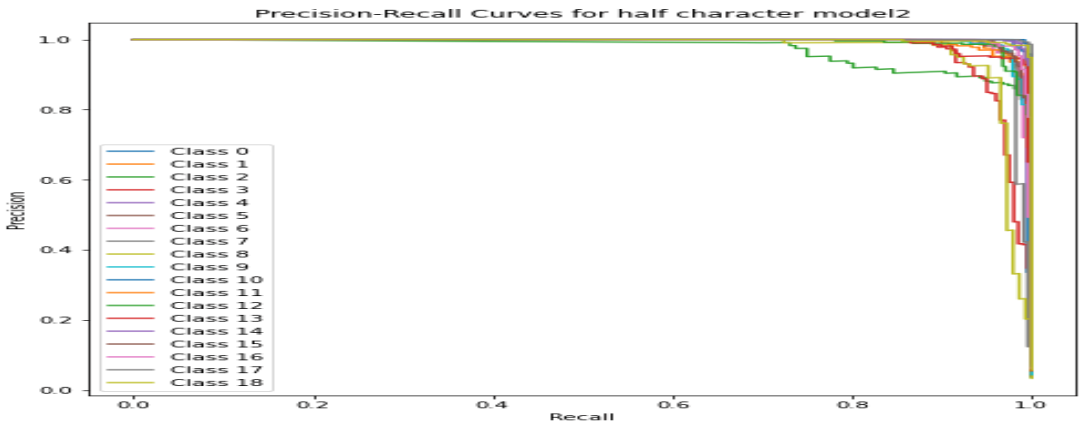


Figure 23: PR curve

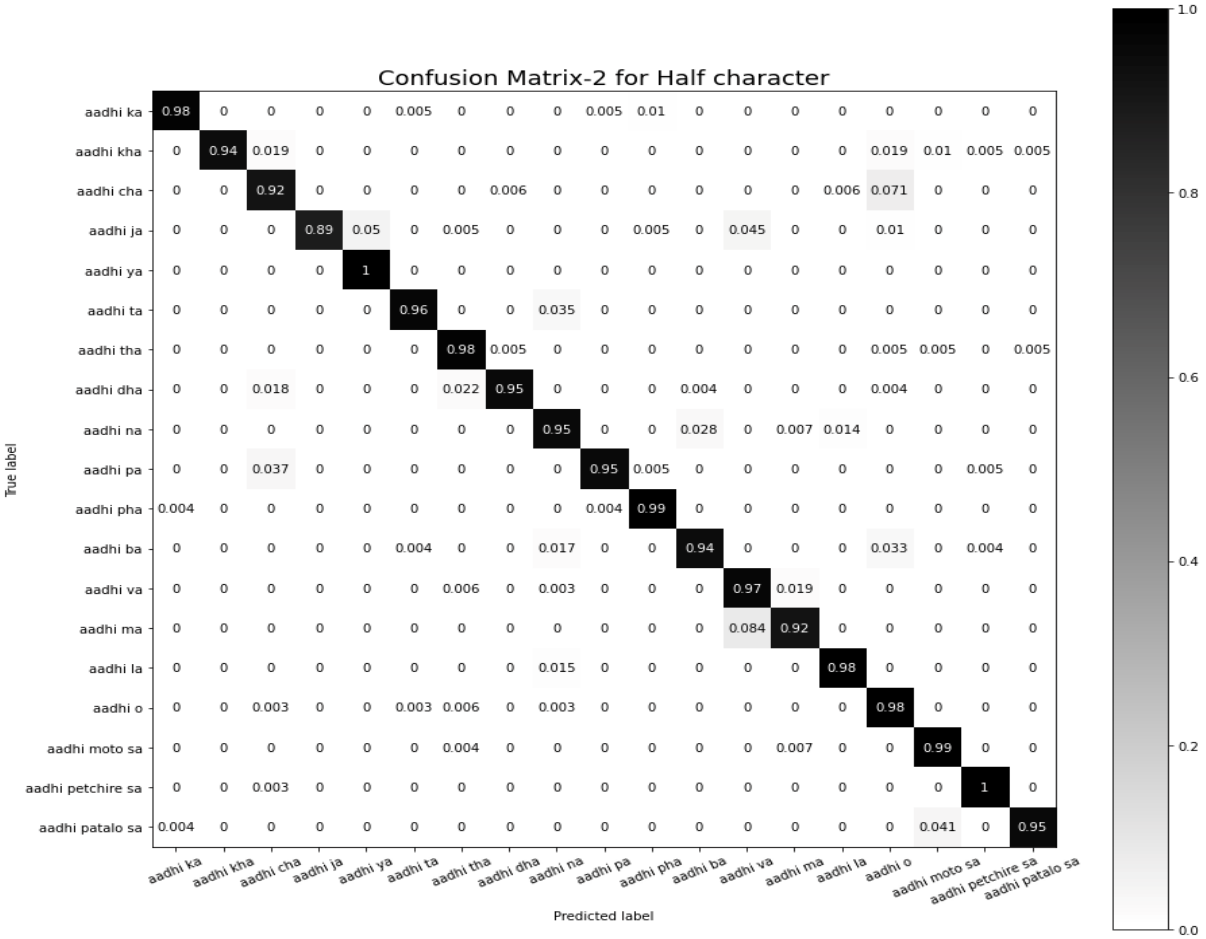


Figure 24: Confusion Matrix for Half Character Model -2

	precision	recall	f1-score	support
0	0.99	0.98	0.98	199
1	1.00	0.94	0.97	208
2	0.89	0.92	0.91	156
3	1.00	0.89	0.94	201
4	0.95	1.00	0.98	199
5	0.98	0.96	0.97	171
6	0.95	0.98	0.96	210
7	0.99	0.95	0.97	227
8	0.90	0.95	0.92	145
9	0.99	0.95	0.97	190
10	0.98	0.99	0.99	255
11	0.98	0.94	0.96	239
12	0.91	0.97	0.94	311
13	0.96	0.92	0.94	237
14	0.99	0.98	0.99	262
15	0.92	0.98	0.95	329
16	0.95	0.99	0.97	271
17	0.99	1.00	0.99	296
18	0.99	0.95	0.97	244
accuracy			0.96	4350
macro avg	0.96	0.96	0.96	4350
weighted avg	0.96	0.96	0.96	4350

Figure 25: Evaluation Metrics for Half Character Model - 2

Upper Modifier Model - 2

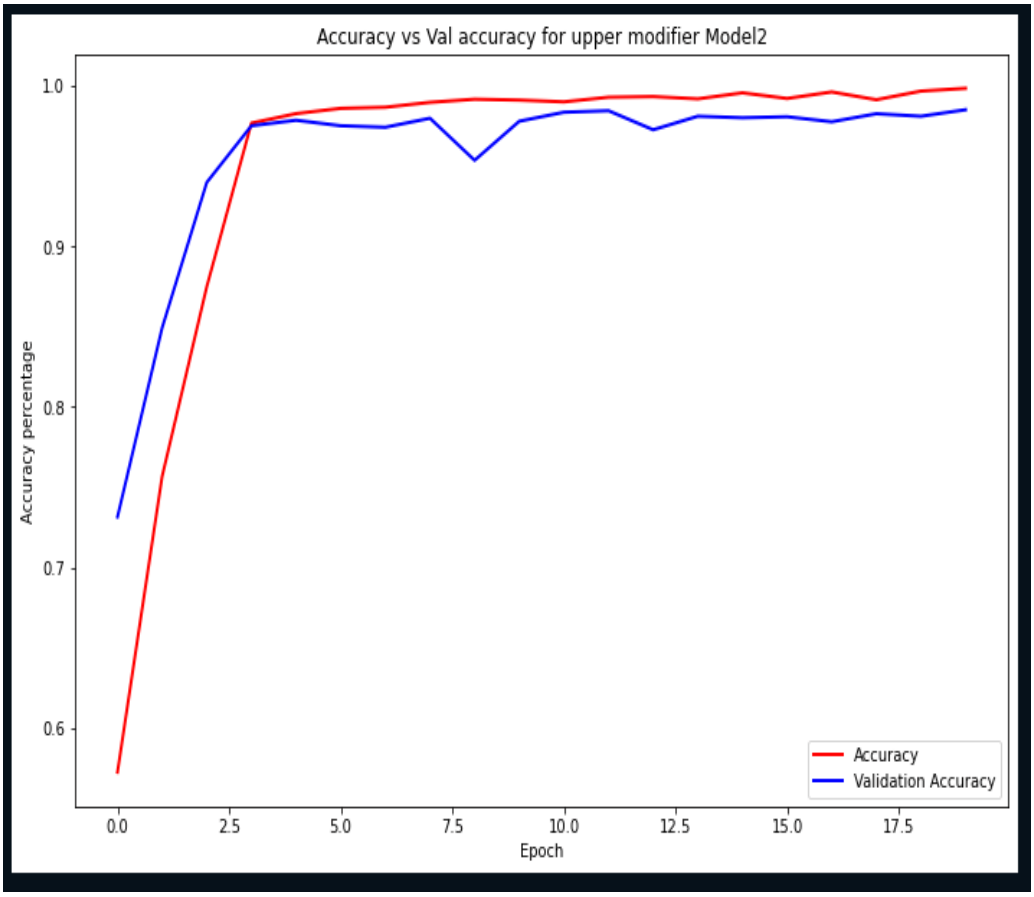


Figure 26: Training vs Validation Accuracy

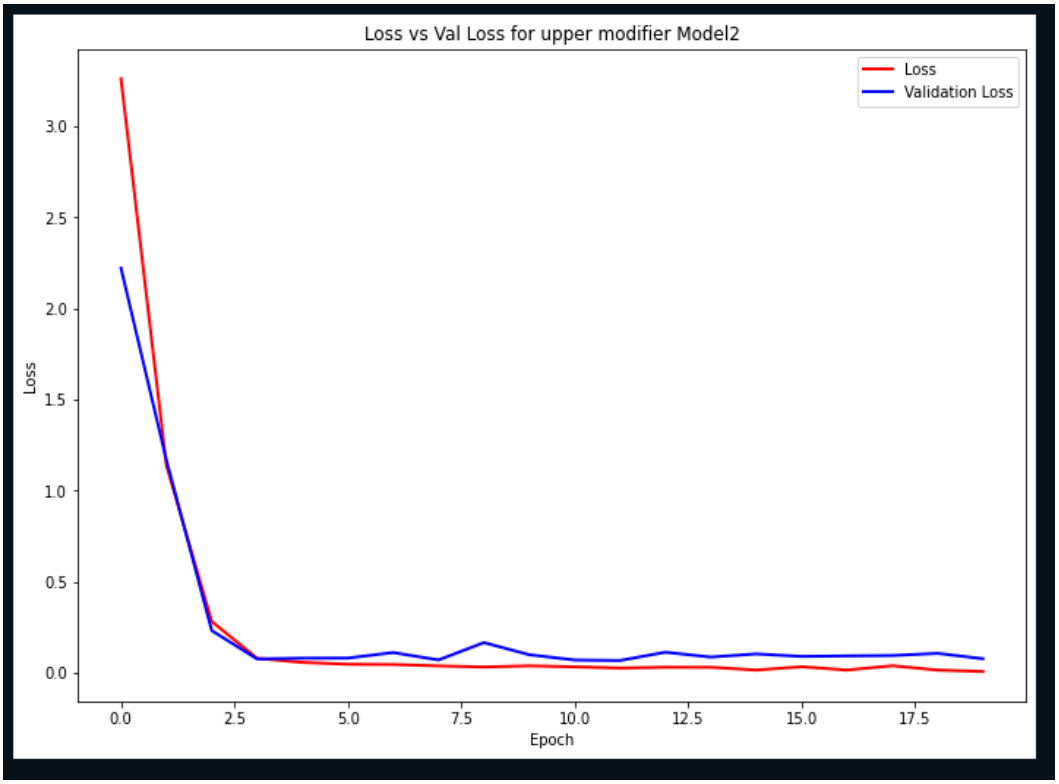


Figure 27: Training vs Validation Loss

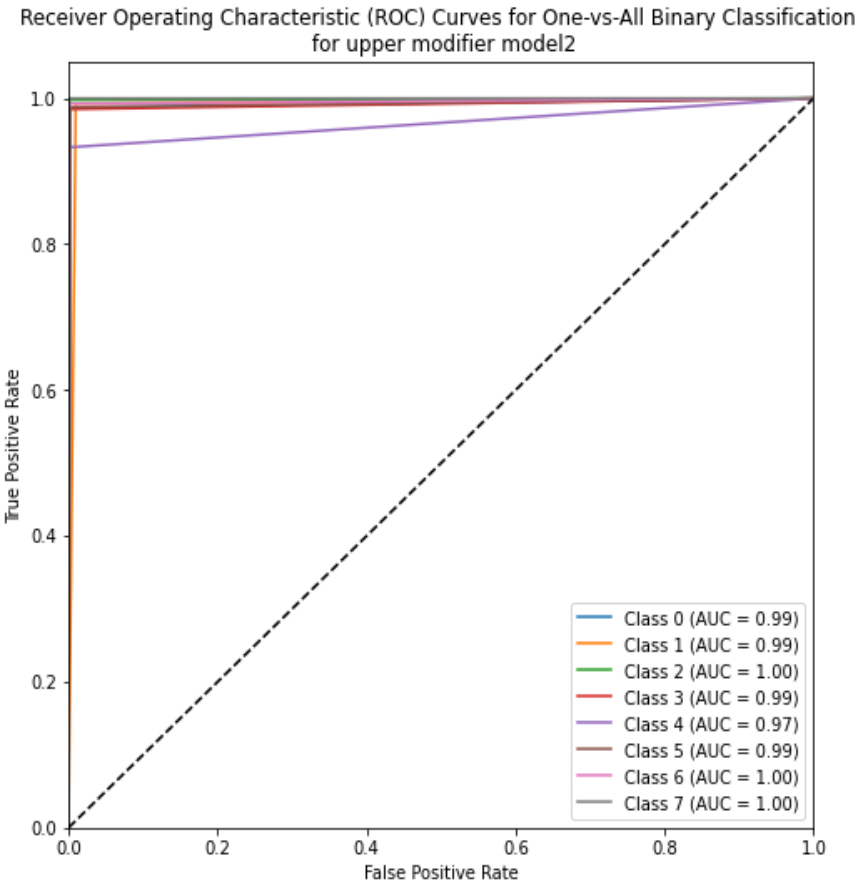


Figure 28: ROC Curve

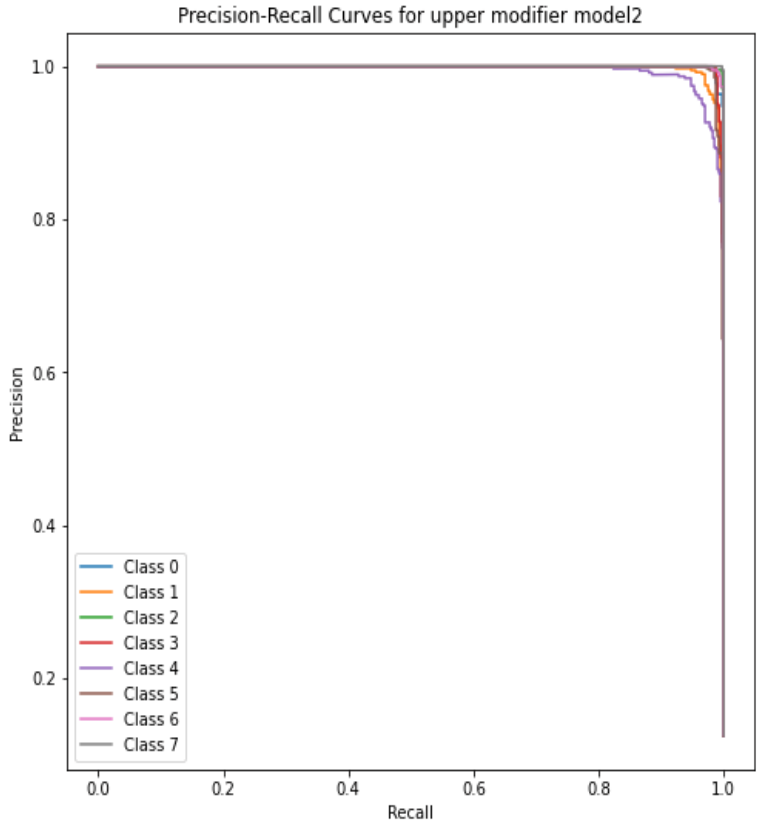


Figure 29: PR curve

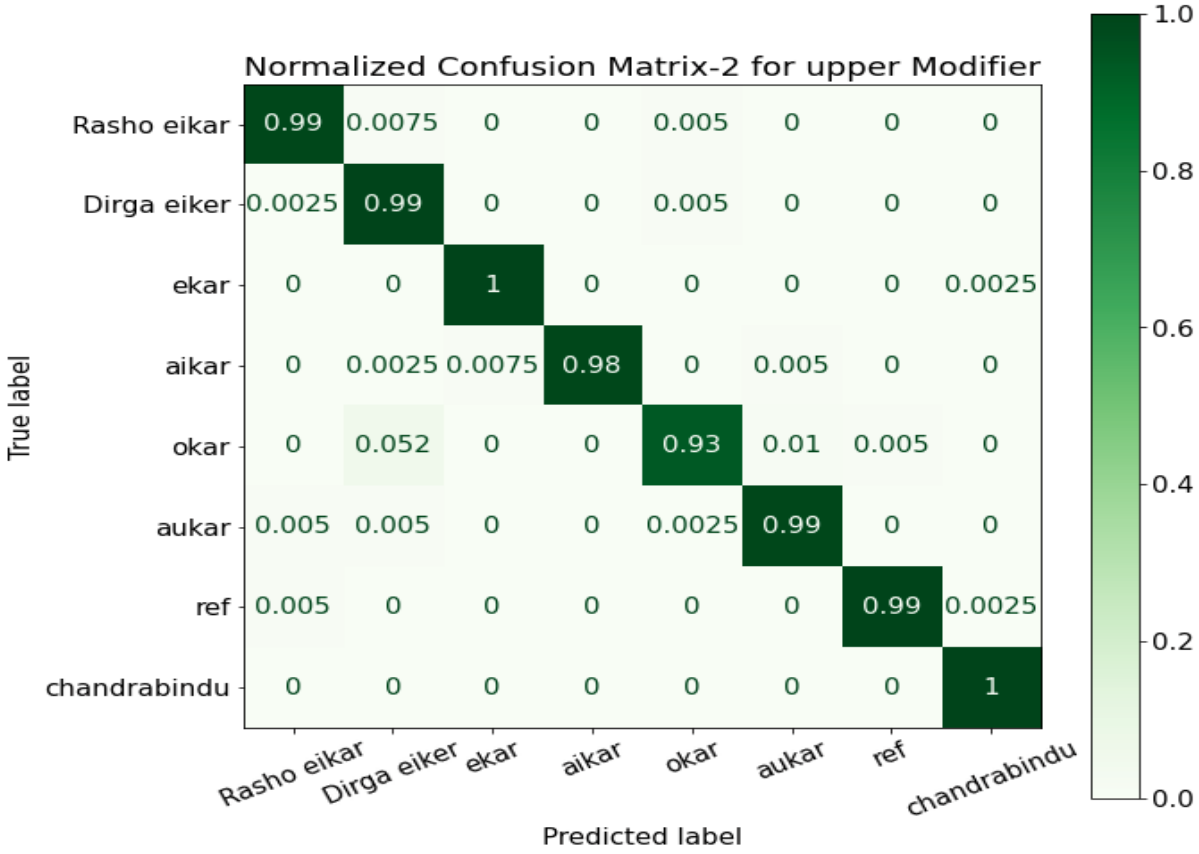


Figure 30: Confusion Matrix for Upper Modifier Model 2

	precision	recall	f1-score	support
0	0.99	0.99	0.99	400
1	0.94	0.99	0.96	402
2	0.99	1.00	1.00	400
3	1.00	0.98	0.99	400
4	0.99	0.93	0.96	402
5	0.99	0.99	0.99	400
6	1.00	0.99	0.99	402
7	1.00	1.00	1.00	400
accuracy			0.98	3206
macro avg	0.98	0.98	0.98	3206
weighted avg	0.98	0.98	0.98	3206

Figure 31: Evaluation Metrics for Upper Modifier Model -2

Lower modifier model - 2

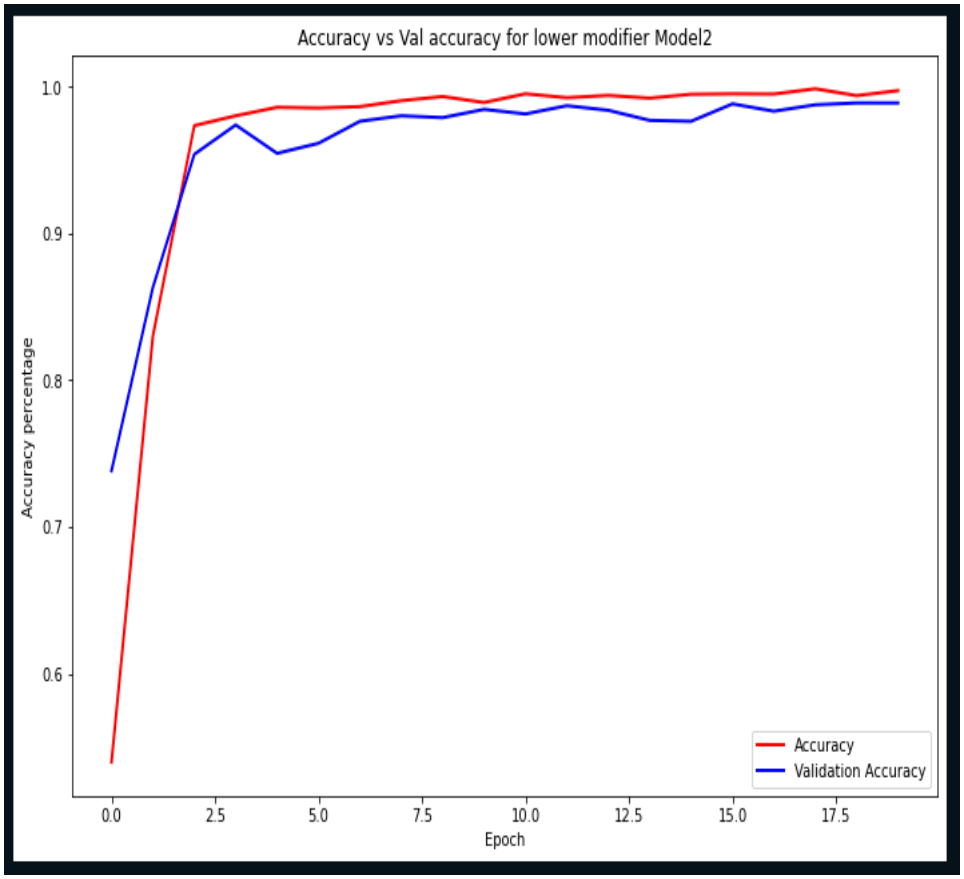


Figure 32: Training vs Validation Accuracy

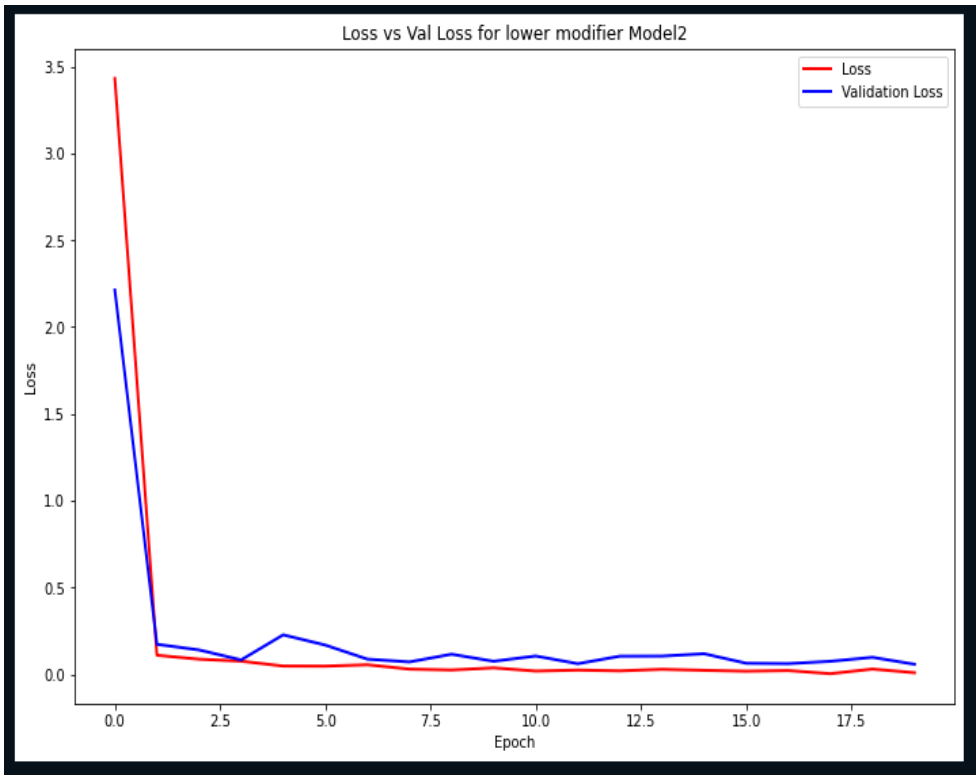


Figure 33: Training vs Validation Loss

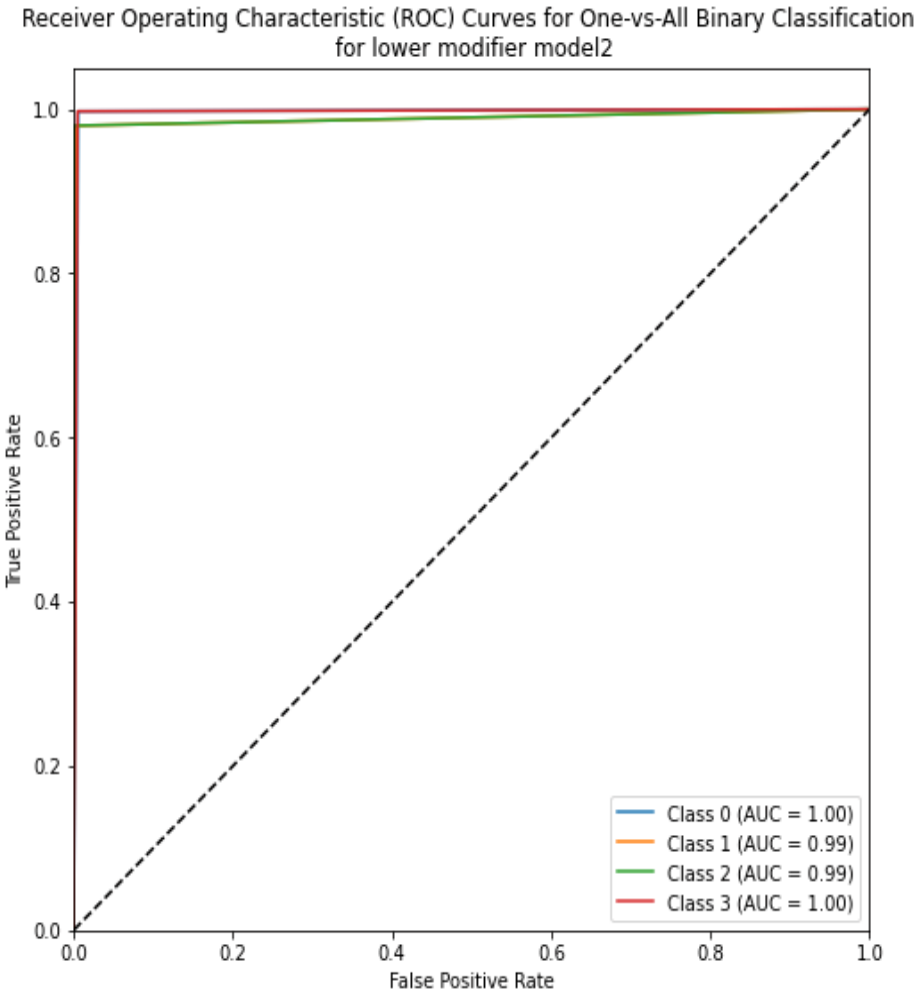


Figure 34: ROC Curve

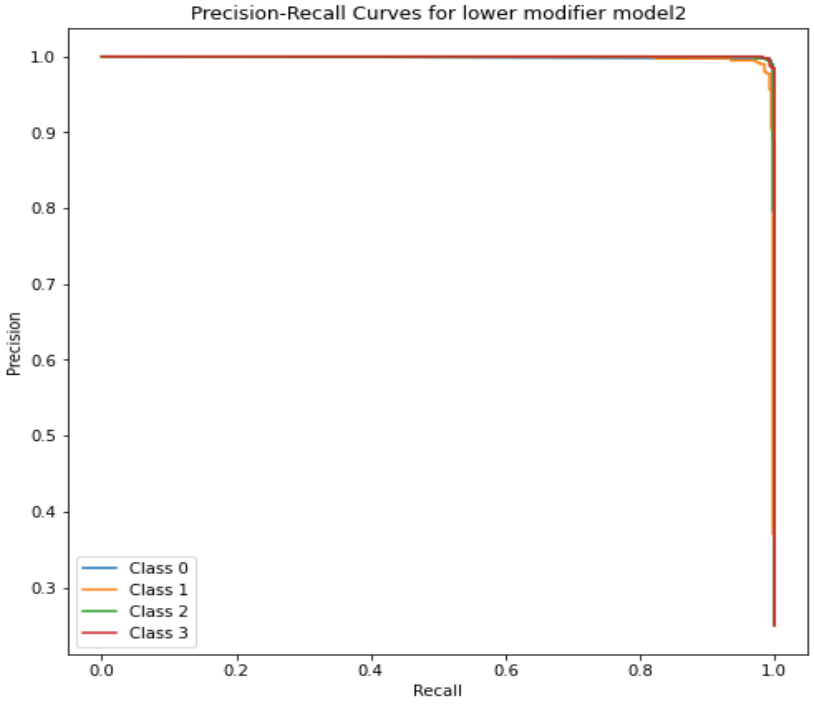


Figure 35: PR Curve

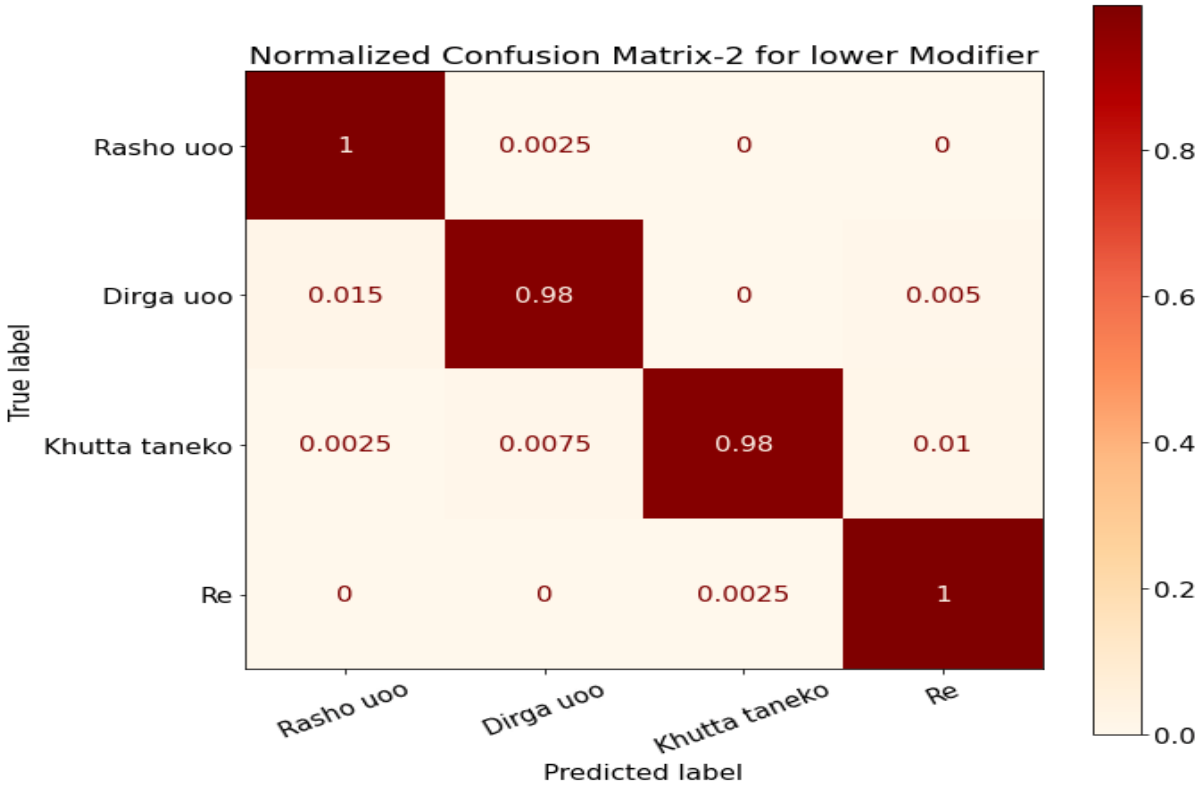


Figure 36: Confusion Matrix – Lower Modifier Model – 2

	precision	recall	f1-score	support
0	0.98	1.00	0.99	402
1	0.99	0.98	0.98	400
2	1.00	0.98	0.99	400
3	0.99	1.00	0.99	400
accuracy			0.99	1602
macro avg	0.99	0.99	0.99	1602
weighted avg	0.99	0.99	0.99	1602

Figure 37: Evaluation Metrics for Lower Modifier Model - 2

While comparing the graphs of training accuracy vs. validation accuracy, training loss vs. validation loss, confusion matrix, evaluation metrics, ROC curve, and Precision-Recall (PR) curve for each model, we use the model that yields the better results. The result is shown below:

- For Main Character Model - Main Character Model -1
- For Half Character Model - Half Character Model -2
- For Lower Modifier Model - Lower Modifier Model -2
- For Upper Modifier Model - Upper Modifier Model -2

In our main model, we have 60 different classes of Devanagari words, where each row and column of the matrix's 60x60 grid represents a certain category of Devanagari words. The values in the matrix represent the percentage of accurate and inaccurate predictions produced by the model for each class.

Analyzing the matrix may reveal which classes the model is having the greatest trouble with. It could be feasible to determine which courses are most frequently misclassified as another class or which classes have the lowest percentage of accurate predictions. By concentrating on increasing the accuracy of the model for those specific classes, this knowledge may be leveraged to enhance the model's performance.

Table 2: Comparison of Proposed Approach vs Prior Work

Topic	Methodology	Accuracy
Devanagari OCR using a recognition driven segmentation framework and stochastic language models.	Recognition driven segmentation using a graph	The recognition driven BAG segmentation has an accuracy range of 72 to 90%
Offline Handwritten Devanagari Word Recognition: A holistic approach based on directional chain code feature and HMM.	HMM (Hidden Markov Model)	80.2%
Improving Nepali OCR performance by using hybrid recognition approaches.	Character Level Recognition and Hybrid approach.	78.87% & 94.80% respectively
An efficient Devanagari Character classification in printed and handwritten document using SVM.	SVM (Support Vector Machine)	99.54% for printed & 98.35% for handwritten images
A Complete OCR for printed Hindi text in Devanagari script	Collapsed Horizontal Projection Technique	93% at character level
Dictionary Based Nepali Word Recognition using Neural Network.	Multi-layer Feed Forward Back Propagation Artificial Neural Network (ANN)	About 90% for simple words, 60% for complex words, and near about 50% for handwritten word
Character Segmentation of Hindi Un-constrained Handwritten Words.	Character segmentation method	Average accuracy rate of 96.93 %.
Proposed Method – Digitization of Devanagari Handwritten Text	Character Segmentation and CNN	Average Accuracy Rate of 95% (For Individual and Simple Words Datasets)

5. Discussion

Sanskrit, Marathi, Hindi, and other Indian languages are all presented in Devanagari characters. The script of Devanagari is complex, and as a result, CNN has not been easy to apply in the recognition of text. An important step in the evaluation of the CNN-based Devanagari text recognition system is the error analysis. The CNN model employed in this study was trained on the Devanagari character set dataset. For each Devanagari character, there were 2000 images in the dataset obtained from an online resource. The batch size used in training the model was 32, while the learning rate adopted was 0.001. The performance of the model was checked by using Precision-Recall and the F1 score. In the observation, it was noted that the model suffered in terms of the segmentation and recognition of the Devanagari script, which contains ligatures, thereby leading to a decline in the performance. This was particularly the case since there was no training data available for the Devanagari text with ligatures. Future work will focus on expanding the dataset to include diverse examples of Devanagari ligatures and developing preprocessing approaches to improve segmentation and recognition of such complex structures. The

error analysis showed that the model was confused between some of the Devanagari characters, such as ज, ञ, ङ, etc. In addition, the model did not capture some word compounds, for example, words with more than one ligature and words with tiny spaces between characters. The CNN-based Devanagari text recognition system was quite good for individual character recognition and word recognition issues. However, it was noted that the model could not distinguish between letters that resembled each other and could not read most of the Devanagari text containing ligatures. The error analysis indicated that the proposed model should be trained on more examples of the Devanagari text with ligatures and a clearer distinction between the visually similar letters. The model achieved around 95% accuracy on clean datasets with individual and simple words but only about 80% on real-world noisy data, which has not been extensively tested. On a broader view, it was demonstrated that it is possible to create a CNN-based Devanagari text recognition system, and there is a need for more research in this area.

6. Ethical Implications

The segmentation algorithm discussed here can also, in theory, be used apart from character recognition. A case in point is forgery detection, where the system deconstructs and examines every component of a handwritten word. By comparing the segmented patterns of different handwriting samples, a model can be established to determine inconsistencies or anomalies that indicate forgery.

However, it is important to note that the dataset itself is biased towards handwritten Nepali script. Since some of the more difficult Devanagari characters are rendered heterogeneously across languages and geography (e.g., Hindi, Marathi, and Sanskrit), the model won't generalize well to other scripts based on the Devanagari alphabet. This kind of bias may introduce fairness issues in applying the model to more general, multilingual settings, and needs to be addressed in future work by collecting more representative data.

7. Conclusion

Handwritten text recognition in Nepali script is challenging due to its large variations in the writing style of handwritten text. However, with the help of new technologies in machine learning and computer vision, there have been improvements in the ability to develop the recognition systems of the Devanagari handwriting. These systems possess the capacity to greatly enhance access to information and knowledge, especially in areas where traditional printed text is not readily available. Future improvements in accuracy, variant handling, contextual recognition, multi-script recognition, integration with other technologies, and improved user interface will likely help systems. This is more efficient and more accessible. However, the Devanagari handwriting recognition field is constantly evolving, and Additional study and advancement are required to continue to improve the performance and capabilities of these systems.

Acknowledgments

We wish to convey our heartfelt appreciation to everyone who supported us with our major project in our final year. Firstly, we wish to convey our appreciation to Thapathali Campus for giving us the chance to pursue our academic goals and providing us with access to the materials and facilities we needed to finish our project. We are very grateful to our supervisor Er. Praches Acharya for providing the necessary guidance and supervision to complete the project. We would like to express our appreciation for the advice he provided in the definition of the work direction and the time he dedicated to making sure that we get in the right direction.

We would like to thank all the friends and family who have been a support system all through the years of our study. We have been privileged to have them with us, and the love they showered on us is immeasurable. We would like to express our appreciation to everyone and will strive to make them proud in the future.

References

[1] S. Kompalli, S. Setlur and V. Govindaraju, "Devanagari OCR using a recognition driven segmentation framework and stochastic language models," *International Journal on Document Analysis and Recognition (IJDAR) ,* vol. 12, pp. 123-138, 2009.

[2] B. Shaw, S. K. Parui and M. Shridhar, "Offline Handwritten Devanagari Word Recognition: A Holistic

Approach Based on Directional Chain Code Feature and HMM," *2008 International Conference on Information Technology,* pp. 203-208, 2008.

[3] N. V. Rao, D. A. Sastry, A. Chakravarthy and K. P, "OPTICAL CHARACTER RECOGNITION TECHNIQUE," *Journal of Theoretical and Applied Information Technology,* vol. 83, 2016.

[4] N. Pant, "Improving Nepali OCR performance by using hybrid recognition approaches," *7th International Conference on Information, Intelligence, Systems & Applications (IISA,* pp. 1-6, 2016.

[5] S. Puri and P. S. Singh, "An efficient Devanagari character classification in printed and handwritten documents using SVM," *Procedia Computer Science 152,* pp. 111-121, 2019.

[6] D. S. Pande, P. P. Jadhav, R. Joshi, A. D. Sawant, V. Muddebihalkar, S. Rathod, M. N. Gurav and S. Das, "Digitization of handwritten Devanagari text using CNN transfer learning- A better customer service support," *Neuroscience Informatics 2,* vol. 2, no. 3, 2022.

[7] S. Sayyad, A. Jadhav, M. Jadhav, S. Miraje, P. Bele and A. Pandhare, "Devnagiri Character Recognition Using Neural Networks," *International Journal of Engineering and Innovative Technology (IJEIT),* vol. 3, no. 1, pp. 476-480, 2013.

[8] C. Johny, L. Wolf-Sonkin, A. Gutkin and B. Roark, "Finite-state script normalization and processing utilities:The Nisaba Brahmic library," *The 16th Conference of the European Chapter of the Association for Computational Linguistics ,* pp. 14-23, 2021.

[9] S. Shakya, S. Tuladhar, R. Pandey and B. K. Bal, "Interim Report on Nepali OCR," *Madan Puraskar Pustakalaya,* 2009.

[10] A. Ghimire, A. Chapagain, U. Bhattarai and A. Jaiswal, "Nepali Handwriting Recognition using Convolution Neural Network," *International Research Journal of Innovations in Engineering and Technology,* vol. 4, no. 5, pp. 5-9, 2020.

[11] B. R. Dawadi, R. C. Pandey, S. Sharma and A. Basnet, "Dictionary Based Nepali Word Recognition using Neural Network," *International Journal of Scientific & Engineering Research,* vol. 8, no. 5, pp. 473-479, 2017.

[12] S. Prajapati, A. Maharjan, S. R. Joshi and B. Balami, "Assessing and Analyzing Tesseract Based Nepali Script OCR," *Deerwalk Journal of Computer Science and Information Technology,,* vol. 1, pp. 37-46, 2019 .

[13] Ingroj, "nepalinlp," 14 12 2017. [Online]. Available: https://nepalinlp.com/processing-unicodedevnagari-in-python. [Accessed 20 12 2022].

[14] S. Bag and A. Krishna, "Character Segmentation of Hindi Unconstrained Handwritten Words," *IWCIA 2015: Combinatorial Image Analysis ,* vol. 9448, pp. 247-260, 2016.

[15] M. Sinha and V. Bansal, "A complete OCR for printed Hindi text in Devanagari script," *Proceedings of*

Sixth International Conference on Document Analysis and Recognition, pp. 800-804, 2001.

[16] Centre for Development of Advanced Computing , [Online]. Available: https://www.cdac.in/.

[17] Madan Puraskar Pustakalaya, [Online]. Available: https://madanpuraskar.org/.

[18] indsenz, [Online]. Available: http://www.indsenz.com/.

[19] "Kaggle," [Online]. Available: https://www.kaggle.com/datasets/inspiring-lab/nepali-number-plate-characters-dataset.