

Spring 5-22-2019

Deep Learning Based Real Time Devanagari Character Recognition

Aseem Chhabra
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Artificial Intelligence and Robotics Commons](#)

Recommended Citation

Chhabra, Aseem, "Deep Learning Based Real Time Devanagari Character Recognition" (2019). *Master's Projects*. 717.
DOI: <https://doi.org/10.31979/etd.3yh5-xs5s>
https://scholarworks.sjsu.edu/etd_projects/717

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Deep Learning Based Real Time Devanagari Character Recognition

A Thesis

Presented to

Dr. Robert Chun

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 298

By

Aseem Chhabra

March 2019

The Designated Project Committee Approves the Project Titled
Deep Learning Based Real Time Devanagari Character Recognition

By

Aseem Chhabra

Approved for the Department of Computer Science

San Jose State University

May 2019

Dr. Robert Chun, Department of Computer Science
Dr. Thomas Austin, Department of Computer Science
Mr. Rahul Dalal, Software Developer, IBM

Acknowledgement

I would like to express my gratitude to my project advisor Dr. Robert Chun for his unwavering support and would like to thank him for his guidance during every step of the project. I would also like to thank my committee members, Dr. Thomas Austin, and Mr. Rahul Dalal, for supporting me throughout the project, their suggestions and time.

Abstract

The revolutionization of the technology behind optical character recognition (OCR) has helped it to become one of those technologies that have found plenty of uses in the entire industrial space. Today, the OCR is available for several languages and have the capability to recognize the characters in real time, but there are some languages for which this technology has not developed much. All these advancements have been possible because of the introduction of concepts like artificial intelligence and deep learning. Deep Neural Networks have proven to be the best choice when it comes to a task involving recognition. There are many algorithms and models that can be used for this purpose. This project tries to implement and optimize a deep learning-based model which will be able to recognize Devanagari script's characters in real time by analyzing the hand movements.

Index Terms – Optical Character Recognition, Artificial Intelligence, Deep Neural Network, Deep Learning, Devanagari Script

Table of Contents

1. Introduction	1
1.1 Architecture	2
1.2 Script Representation	7
1.3 Challenges:	8
2. Previous work.....	10
2.1 Recognition of Printed Devanagari Script.....	10
2.1.1 Preprocessing Techniques	10
2.1.2 Feature Extraction Techniques	12
2.1.3 Recognition Techniques	14
2.2.2 Character Classification using Pattern Classification	17
2.2.2.1 Support Vector Machine.....	17
2.2.2.2 Multilayer Perceptron.....	19
2.2.2.3 Hidden Markov Model	21
2.2.2.4 Recurrent Neural Network (RNN)	22
2.3 Character Recognition for Documents written in Devanagari Script	24
2.3.1 Why this research?	24
4. Implementation.....	27
4.1 Dataset.....	27
4.2 Challenges	30

5.	Implementation	34
•	Ridge Classifier	34
•	Bernoulli Naïve Bayes	34
•	Gaussian Naïve Bayes (GaussianNB)	35
•	DecisionTreeClassifier	35
•	ExtraTreeClassifier	37
•	KNeighborsClassifier	37
•	ExtraTreesClassifier:	38
•	RandomForestClassifier	38
6.	Result	47
7.	Conclusion and Future Work	58
8.	Future Work	60
	References	60

Table of Figures

Fig. 1.1: Training Pipeline and Testing Pipeline of a CNN Model [4]	4
Fig. 1.2: Weighted Inputs to the Neuron	5
Fig. 1.3:UNICODE Rearranging to Render Correctly [7]	9
Fig. 2.1: General Character Recognition System Pipeline [2]	17
Fig. 2.2.1: Multiple Ways in Which the Classes can be Separated	19
Fig. 2.2.2: Optimal Line Separating the Classes	19
Fig. 2.3: Artificial Neuron	20
Fig 2.4: FFNN With an Input Layer, One Hidden Layer, and an Output Layer	23
Fig. 2.5: Hidden Layer Having Additional Feedback	23
Fig. 4.1: Scanned and Greyscaled Documents Written in Hindi	32
Fig. 4.2: Process of Dataset Creation	32
Fig. 5.1: Decision Tree Regression [20]	36
Fig. 5.2: A Code Snippet Showing Tree Classifier on Iris Dataset	36
Fig 5.3: Scores Obtained from Various ML Algorithms	39
Fig. 5.4: Plot for Accuracy Score of K Nearest Neighbor Classifier	40
Fig 5.5: Plot for Accuracy Score of Extra Trees Classifier	40
Fig 5.6: Plot for Accuracy Score of Random Forests Classifier	41
Fig. 5.7: Plot for Learning Curve of Extra Trees Classifier	41
Fig. 5.8: Structure of Network	44
Fig. 5.9: Training of a CNN	45
Fig. 6.1: Accuracy vs Epoch for model 3x3	48
Fig. 6.2: Loss vs Epoch for model 3x3	48

Fig. 6.3: Accuracy vs Epoch for model 5x5	49
Fig. 6.4: Loss vs Epoch for model 5x5	50
Fig. 6.5: Loss vs Epoch for model 3x3 with increased number of filters	51
Fig. 6.6: Accuracy vs Epoch for model 3x3 with increased number of filters	50
Fig. 6.7: Accuracy vs Epoch for model 5x5 with increased number of filters.....	52
Fig. 6.8: Loss vs Epoch for model 5x5 with increased number of filters	50
Fig. 6.9: Accuracy vs Epoch for model 7x7 with increased number of filters	53
Fig. 6.10: Loss vs Epoch for model 7x7 with increased number of filters	54
Fig. 6.11: Accuracy vs Epoch for model 3x5	55
Fig. 6.12: Loss vs Epoch for model 3x5.....	56
Fig. 6.13: Accuracy vs Epoch for model 5x7	56
Fig. 6.12: Loss vs Epoch for model 5x7.....	57

List of Tables

Table 1.1: Major Language Families in India [2]	6
Table 1.2: Base form of consonant characters	7
Table 1.3: Vowel Characters	7
Table 1.4: Numeral Characters	7
Table 1.5: UNICODE ranges for major Indian scripts [4]	8
Table 4.1: numerical class	27
Table 4.2: Vowels Class	28
Table 4.3: Consonants Class	29
Table 4.4: Structural difference of characters	30
Table 4.5: Similar Looking Characters Due to the Handwriting Style	31
Table 6.1: Result of Machine Learning Algorithms	47
Table 6.2: Result of CNN model with different kernel sizes	48
Table 6.3: Result of CNN model with different kernel sizes and increased number of filters....	50
Table 6.4: Result of CNN model with added dropout layer	54
Table 6.5: Result of CNN model with cross combination of kernel size	55

1. Introduction

With the extreme pace of technology development, there has been a desire to make the machines work and function as humans do. The advancements in field of Artificial Intelligence and Deep Learning have gathered people's attention and interest in areas such as computer vision and human-computer interaction and has begun a phase of developments to make it possible for machines to see, think and process things around it in the same way as humans do with the help of their sensory organs such as eyes and brain.

Character recognition is the natural way of interacting with the computer. It has been a field of great interest for researchers and scientists. Character recognition is the process where the machine detects and recognizes the characters from a text image and converts that processed data into a code which is understood by the machine. It is a fundamental yet challenging task in the field of pattern recognition. It is termed as optical character recognition (OCR) as they deal with characters which are optically processed and not magnetically processed [2]. It has a wide variety of applications in many computer vision problems such as recognizing car number plate characters, recognizing the address and zip code on a letter and many more [3]. The concept of character recognition was introduced by Carey in 1870 when he developed the retina scanner [3]. Then in 1890, the sequential scanner was introduced which brought in the revolution for reading machines. Those were only proposed as an aid for visually impaired patients and later a successful attempt was made by a Russian scientist in 1900. However, commercial character recognition engines started to appear during the 1960s. These commercial OCRs worked based on template matching. This was followed by the launch of letter-sorting engine by Toshiba. It was able to sort the characters and letters for postal code numbers. During the time from 1970 to 1980 many OCR engines were introduced that could read characters printed or written in English.

The advancement of technology in terms of both scanning devices and the computation power has also led to the development in this field. Today, a number of OCR's are present in the market which can perform recognition task with minimal error rate. Some of these are also available for handheld and portable devices like cellular phones.

Lately, several such systems have been developed for more complex languages such as Chinese, Korean, Arabic, etc. but the accuracy and consistency are not at par as compared to those for the English language. One of the reasons for this is that English Script is not as complex and complicated as that of Chinese or Devanagari Script. Pertaining to this reason, the existing character recognition faces difficulties in understanding the not so popular languages like Sanskrit, Hindi, and Tamil, and hence are not useful for people whose native language is not English. Another shortcoming of the majority of the OCR's is that they fail to understand the context of the text.

1.1 Architecture

The process of optically recognizing a character of a text usually begins with an image of the document which is generally obtained by scanning or via camera captures. Document images obtained via these methods need to be preprocessed before they can be used for the process of recognizing the characters in that image. Preprocessing includes steps like noise removal, binarization and skew detection. It starts with Binarization, where the image is converted into a grey scaled image from a colored image. In a binary format image, the pixel value is either a 1 or a 0, 1 being black and 0 being white. Binarization is followed by the process of correcting skewness of the image. As mentioned earlier, since the document images that we use are scanned either via a scanner or a camera capture, the image tends to have a degree of skewness and is required to be corrected so as to have a minimal error. The method opted for the correction of skewness in the

image is dependent on the type of language. In case of Devanagari Script, this process gets a little easy as the words have Shiro-Rekha/ headline, which connects all the components.

After completion of these two steps, the document image is then processed by the layout engine to analyze the structure of the document. This helps in finding out all the components of that document. It may be possible that the document has various blocks of text, images, and tables together, hence finding these blocks is necessary. Further, the words are segmented into characters to be processed, recognized and classified.

One of the key points in a classification problem is feature extraction such as SIFT, appearance-based features, etc. A classifier is then used over these extracted features which classify them into different output labels. A number of machine learning algorithms can come in handy for this purpose depending upon the nature and amount of training data present. For example, we can use SVM if the features can be linearly separated. Neural Networks, Deep Learning, Random Forest, etc. can be used when the features are not linearly separable and are complex.

The classifiers convert the input into UNICODE labels which need to be re-ordered so that the text editor can render them properly. In case of Devanagari, consonants combined with vowels make a combined character, so for these cases, the classifier assigns a single label to them.

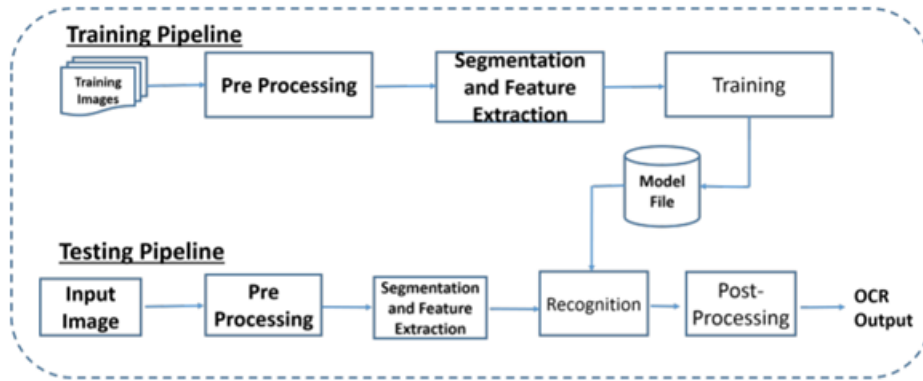


Fig 1.1: Training Pipeline and Testing Pipeline of a CNN Model [4]

In Fig 1.1, A postprocessing unit is generally used in an OCR to improve upon the errors in classifying by the classifier. For this purpose, the post-processing unit uses models of other languages and dictionary.

Character recognition for handwritten content is known as handwriting recognition. It is a technique that is used to recognize and classify the characters and symbols written by hand. It can further be classified into offline and online [4]. In online the recognition, the coordinates of continuous points in the 2D plane are stored as a sequential function of time. While in the offline recognition, only the writings from the images captured by the scanner are used, and this is the reason why online recognition works better. But offline recognition has its own applications and hence is an active area of development [5]. So, the data can be collected using both these methods.

A handwriting recognition system generally follows these steps:

1. Pre-processing: This step involves the modification of the input image in such a way that it is suitable for the next step.
2. Segmentation: It segments each character in that image which is then resized into the size required by the training example.

3. Feature Extraction: The segmented image is then used to extract the features of that character.

There are a lot of segmentation and feature extraction techniques which are used to select the best features out of those present so as to attain high accuracy.

The research by McCulloch and Pitts in 1943[6] gave birth to a human brain alike neural network called the Artificial neural network, and hence inspired the future work in the same field. However, it was in 1958 when Rosenbalatt first implemented a functioning neuron and demonstrated how a perceptron can be trained to predict the output for unseen data. He also came out with two publications for the same in 1958 and 1968 where he explained the working of an artificial neuron. Fig 1.2 depicts that weighted inputs are received by the neuron to generate an output. The weighted inputs are finalized to give the desired output based on the training it received on the training data. Once learned, the neuron is used to predict the output on unseen data [6].

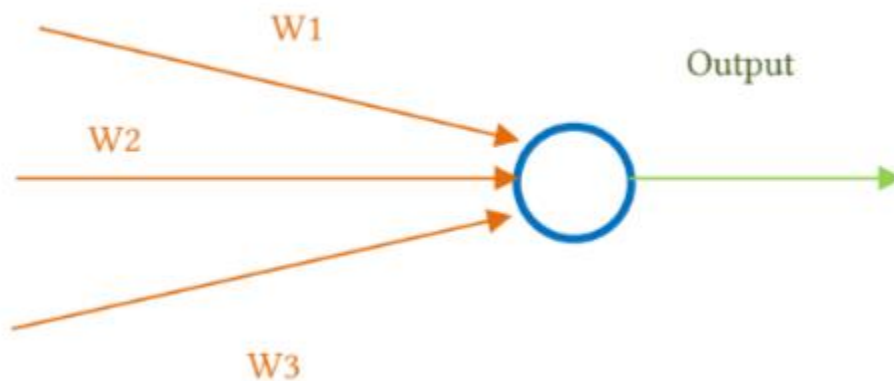


Fig 1.2: Weighted Inputs to the Neuron

There are more than 720 dialects derived from around 13 scripts that are used in India. Out of these 720 dialects, there are 22 official languages of India. Indo- Aryan, and Dravidian are the two

mainly used and adopted language families in India. The usage of each family group in accordance with the population of India is displayed in table 1.1.

Language Family	# of Languages	Sample Language	% of total population
Indo-European	24	Hindi, Bangla, Afghani, Urdu, Sanskrit	76.89%
Dravidian	17	Kannada, Malayalam, Tamil, Telugu	20.82%
Austro-Asiatic	14	Santali, Nicobarese, Korwa	1.11%
Tibeto-Burmeese	66	Tibetan, Bodo, Ladakhi, Manipuri	1.00%
Semito-Hamitic	1	Arabic	0.01%
Total	122	-	99.83%

Table 1.1: Major Language Families in India [2]

Both the Dravidian and Indo-European/Aryan language family are the widely used languages in India. Afghani, Bangla, Hindi, etc. are some of the languages from the Indo-Aryan family while Kannada, Malayalam, Tamil, etc. are the languages belonging to the Dravidian family group. Another common thing between these family groups is that they follow the Devanagari script's way of writing. A language based on this script is easy to identify as the words usually have a headline which connects all the components of the word, and the end of each sentence is marked by a poorna-viranam (.), unlike a full stop in English.

Devanagari is one of the Indic scripts that form the foundation for over a hundred languages that are used in South-East Asia including Sanskrit and Hindi. It mainly consists of 10 digits and 47 alphabets of which 14 are vowels and 33 are consonants.

Table 1.2 shows all the consonants of Hindi language in base form. Table 1.3 shows all the vowels in the Hindi language while table 1.4 shows the numerals in Devanagari script for the Hindi language.

क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट	ठ
ड	ढ	ण	त	थ	द	ध	न	प	फ	ब	भ
म	य	र	ल	व	स	ष	श	ह	क्ष	त्र	ज्ञ

Table 1.2: Base form of consonant characters

अ	आ	इ	ई	उ	ऊ	ए	ऐ	ओ	औ	अं	अः
---	---	---	---	---	---	---	---	---	---	----	----

Table 1.3: Vowel Characters

०	१	२	३	४	५	६	७	८	९
---	---	---	---	---	---	---	---	---	---

Table 1.4: Numeral Characters**1.2 Script Representation:**

In a general term, all these scripts are referred to as Indic Scripts. These are represented as a 16bit representation of a symbol, termed as UNICODE. Indic scripts are inherited from Indian Standard Code for Information Interchange (ISCII). Its derivative ISCII uses an 8bit representation of a symbol. The Unicode range for some of the Indic values is shown in table 1.5 as follows:

Language	Unicode Begin	Unicode End
Hindi	0900	097F
Bangla	0980	09FF
Gurumukhi	0A00	0A7F
Kannada	0C80	0CFF
Telugu	0C00	0C7F
Tamil	0B80	0BFF
Malayalam	0D00	0D7F

Table 1.5: UNICODE ranges for major Indian scripts [4]**1.3 Challenges:**

One of the major challenges while creating a recognizer and classifier for the Indic scripts is that they have a large number of symbols as compared to languages like English. English contains mainly 52 characters while a script like Devanagari consists of more than 200 symbols. Almost all the Indic languages are similar to each other. Some of them have a characteristic of allowing characters to be combined together to form another character, generally referred to as ‘sayuktakshar’ where Sayukt stands for combined and ‘akshar’ means word. Another challenge is the identification of vowel modifiers or ‘matras’. Vowel modifiers change the UNICODE value of that character, and hence, in order to identify the correct character and its Unicode value, we need to write script specific rules. These modifiers may be present at the left, right, bottom or top of the character. In some of the cases, the modifier may be present at two positions on the same character. Hence, it is required to identify these modifiers correctly to reduce the error in classification.

Most of the optical character recognizers for Indic languages work in such a way that they break down the whole word into corresponding symbols and then are recognized separately. It is done in accordance to the order in which they appear, but when the UNICODE is computed for those symbols, they need to be rearranged and sometimes combined to generate the UNICODE. The fact

that the ‘matra’ or vowel modifier needs to appear after the consonants, is why we need to rearrange the symbols. Fig 1.3 shows the process of UNICODE rearrangement which is done so that the output is rendered correctly.

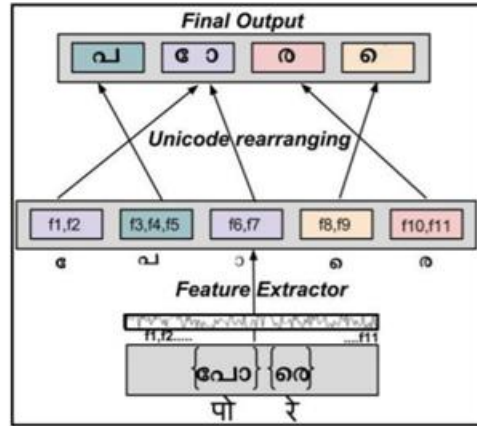


Fig 1.3: UNICODE Rearranging to Render Correctly [7]

It is interesting to note that many text editors do support the rendering of Indic scripts as they support UNICODE, but every editor renders every symbol and UNICODE differently. For example, the symbols which are formed by the combination of two or more symbols are rendered in different ways by different editors. Some editors read them as two different symbols while others read them as a combined symbol. Also, there are some cases where multiple symbols have the same value of the UNICODE.

2. Previous work

2.1 Recognition of Printed Devanagari Script

Handwritten document and Printed documents are generally the two categories of input which are given to the optical character recognizer. Of these, the printed documents are the most commonly used category of the input. There are several steps that need to be done before the document can be forwarded to the categorizer for processing of the input. This step is termed as preprocessing which usually includes steps like the removal of noise, correction of skew, feature extraction, and separation of characters. Following sections describe the previous research and work done in this field.

2.1.1 Preprocessing Techniques

The work towards handwritten character recognition for Devanagari script has been experimented by many researchers using various classification algorithms and feature extraction methods such as statistical, structural and topographical features. However, it was only in 1976 that I.K.Sethi and B.Chatterjee [8] came up with their research in which they tried to use a structural approach. Using this approach, they tried to find the relative positions of the horizontal segment, vertical segment and the slants of that character. On the other hand, N.Sharma [9] divided the image into 64-dimensional blocks and applied a quadratic classifier on those samples. The author reported a test accuracy of 80.36% for characters belonging to the Devanagari script. This way of dividing the data into smaller samples and performing processes on them is called chain code-based feature extraction. This approach was also used by Deshpande et al. [10] in their research in which they used regular expressions along with this approach to create an encoded character array from characters in order to improve upon the accuracy. As a result of this approach, they were able to attain an improvement of 1.74% in the test accuracy.

A small degree of skew is sometimes unavoidable when we are using the data of images that have been scanned using an optical scanner. Skew angle can be defined as the angle that the character makes with the horizontal axis and hence it becomes quite important to remove the skew from that image. For Devanagari script, the skew is estimated by the header line of each word. To overcome this issue of skew error in recognition problem, Chaudhari and Pal [11] introduced a header line detection technique. To complement this, a mathematical morphology based fast and script independent technique was proposed by Chanda and Das [12]. In this technique, the skew is removed followed by the separation of the word into individual characters. In order to segment a word into characters, the header line needs to be removed, which happened to be a challenge until Garain and Chaudhary [13] proposed their technique based on a fuzzy multifactorial analysis. Bansal and Sinha [14] presented a two-pass algorithm which also made it easier to remove the headline. In the two-pass algorithm for character segmentation, structural properties of the script are used to separate the header line. Horizontal and vertical incline were taken into consideration to predict the shape of the character. The segmentation was done based on the overlapping or touching of the characters; that is, all the characters in a sentence that touched each other were left unsegmented. The final selection of the portion of the image is based on the height and width of the text. Komapalli et al. [15] on the other hand used a graph-based technique to segment characters. In their research paper they stated that the header line can be determined based on run length and profile, and once the header line is segmented from the word, the individual characters can be classified based on their different height, width, and top-bottom zone identification. Vowel components which are present at the top and bottom of the word are scaled to a standard size. Then the segmentation is done based on fuzzy multifactorial analysis, where the cut points of the touching characters are selected using a predictive algorithm [16].

In order to analyze the text written in the Devanagari script, the images need to be analyzed before processing it. There might be chances that the image is not in a greyscale format; that is, the image might be a scene image which adds complexity to the process of extracting text from it. A way to extract Devanagari text from a scene image was proposed by N. B. Mapari et al. [17]. The method uses an adaptive thresholding technique. Similarly, C. V. Jawahar et al. stated that a water reservoir based technique can be used to extract the lines from a scene image. While in [18], Neeraj Pratap extracted the text from scene images using Linear Discriminant Analysis (LDA) and principal component analysis (PCA) [16]. If the scene image consists of more than one language, that is, the text is multilingual, then the words are segmented using a machine learning algorithm known as Support Vector Machine (SVM). This technique segments the text block into lines which are further segmented into words. These segmented words are processed based on their structural features, and to attain higher accuracy, a two-stage approach is suggested in [19].

2.1.2 Feature Extraction Techniques

Devanagari characters have various features that can be used to recognize a particular character. Sinha and Mahabala [20] proposed a system in their research in which they described the relations between characters of the script. Their research demonstrated how understanding the association between symbols can help in understanding and recognizing the characters. They divided the characters into three groups, and then considered the features group wise. Jayanthi et al. [21] in their research considered two main features of Devanagari script that are the lines – horizontal and vertical. Developing on their research, they tested the placement of vertical lines and concluded that for most of the characters, the vertical line exists on the right side of the character and it can be considered as a feature while trying to classify that character. They also considered other features like aspect ratio and broadness of the ending. Aspect ratio is the ratio of height to width

of the character, while broadness of the ending means whether the character symbol is narrow ended or broad [12]. Gradient features are one of the features that were considered by Govindaraju et al. [16] and Kompalli et al. [22]. They used gradient features to classify segmented images. Using a similar approach Dhurandhar et al. [21], extracted the significant contours of the character symbols and categorized them based on the reference of contour set with the coordinate system.

Like [10], who used PCA to extract the characters in his research, [12] also used the same approach to extract features from printed text characters. Developing further on this research, [13] developed a word level matching system that was capable of searching characters and words in printed document images. The system tries to extract all the local features of the character by scanning through the vertical lines of the word image, process the individual line and then combine them according to their relevance with each other. In another approach formulated by Ma and Doermann [14], script features and structures were used to identify Hindi words in multilingual documents. They both used a support vector machine (SVM), a machine learning algorithm, to do this. The words identified in the above-stated process were then segmented into characters where the characters were further broken down depending upon the script's structural properties and statistical information.

In order to identify features of machine printed text, they used 64-D CH gradient features and presented them in the form of a graph called geometric feature graph (GFG). 64-D CH gradient feature means that the image's bounding box is segmented into smaller blocks and a chain code of histogram (CH) is computed for each block. The features extracted from this technique of feature selection are then represented in the form of a graph and if required, its dimensionality could be reduced using LDA.

2.1.3 Recognition Techniques

A variety of classifiers and techniques like ANN [18], [19], HMM [20], fuzzy rules, rough sets [8], etc. have been used by researchers for the purpose of classification and recognition in the past. Basic characters were successfully recognized using a top-down tree-based classifier. [7] claimed this as a better method as binary trees are proven to be the fastest decision-making process of a computer program.

In their research [23] took into consideration 38 characters and 83 frequently occurring character classes for classification [23]. So, the characters were clubbed into 4 categories based on their structural properties. Each of these was then classified using different layers of an ANN, which in this case was a three-layer ANN.

Kompalli et al. [23] in their research described two different techniques for character recognition of Devanagari text. They used neural network classifiers for recognition classifying printed characters and words [23]. In contrast to this, Jawahar et al. [20] used support vector machines for the same purpose. In this, they used 5 filters to process the image. The filters used structural features of the characters and include a vertical bar, horizontal crossing, moments, and vertex points.

In [6], an HMM model was created with 14 states and a maximum of 256 Gaussians in each HMM. Some other techniques like nearest neighbor classifier and hierarchical classification techniques were also used in their research.

On printed text, the resultant accuracy is not that good as the ancient print quality was not up to the mark and also the paper quality used to be poor [21]. Hence, the main reason for the low accuracy of those characters is because of the noisy images and distortion of characters. So, in

order to handle such documents, an approach was described by Dhingra et al. [21] in which the classification error was calculated, and was required to be minimum and hence called as minimum classification error (MCE). This helps in providing robustness of the system against noise. They created a model to simulate the distortions caused by errors during scanning. In this, the effectiveness of the Gabor filter was independently calculated from discrete cosine transformation. Also, the features were independently calculated using nearest neighbor and SVM classifiers. They computed that Gabor-SVM had an edge over the other combinations [23]. In [22] a classification system was created which helped in classifying the text to its respective script using the maximum voting scenario and SVM algorithm.

Chaudhari and Pal [19] proposed a model of a complete OCR on stroke based feature for Bangla language and script. For this, they used a tree classifier, which is usually used to recognize and classify compound characters. This approach was improved by using a template matching method. Here, usage of several heuristics made the model speed up and reduced the error rate. An error correction method based on a dictionary was integrated and different dictionaries were created for suffix and root.

Based on this approach an OCR was presented by Ghosh et al [3] for Assamese language using a two-stage Support Vector Machine (SVM) to increase the performance of the system. They used a combination of local and global features like directional features and profile-based features respectively. The model, being a two-stage model, helped in improving the performance as the confused classes, which did not provide a clear result in the first stage, could be recognized in the second stage. A similar two feature set approach was used by Lehal and Singh [23] for Gurmukhi script. They also used a combination of a binary tree and nearest neighbor classifiers at different stages which were then supplemented by a post-processing step. An accuracy of 96% was reported

for this approach on a 100-page test dataset. Another SVM based OCR, which captured the shapes of characters, was also presented for Tamil script. Structural features were then extracted from the captured shapes. These features could categorize the foreground pixels according to their directions.

An OCR for Gujarati language script was introduced by Antanani and Agnihotri [6]. They incorporated the use of Hamming distance and Euclidian distance along with KNN classifier. A character n-gram approach was used by Shrey et.al [17]. Profile based features were used in addition to the nearest neighbor classification algorithm for the purpose of recognition. Even though their approach was quite robust and innovative, the method proposed used a lot of memory during execution, and hence was not effective when the files to be recognized and classified were large. Another robust classifier for Telugu script was proposed by Rasagana et al. [23]. The classifier was able to classify characters even with variation of the font of the text. It was successful in dealing with font variation and could be scaled to a large number of classes. The model was tested against classes consisting of 15 variations of the font. This approach was further improved by Pavan Kumar et al. [3]. The accuracy of their model was increased for the segmented characters with the use of broken characters. The idea was to use the feedback from the distance measure used by classifiers to deal with broken characters. The segmentation accuracy was improved by using script specific segmentation method of the character.

Attempts were made to recognize Urdu script using LSTM algorithm. In this method, pixels of the character were used as features which helped in obtaining high accuracy rate of recognition.

2.2 Recognition Architecture

2.2.1 Recognition by recognizing each character individually

A word is considered to be a combination of individual characters, that is, in this approach, the word is separated into individual characters which are then recognized. The recognized result of each character is combined together to form the original word. However, breaking the word into individual symbols is a challenging job especially when the script is like Devanagari. For example, in case of Hindi, we need to remove the headline or the Shiro-Rekha as the first step and then individual characters are recognized. Once the recognition is done the output of each symbol is combined to form the initial word. E.g.: In “i”, if we have to recognize it using this method then it will be considered into two different components that are the dot and the body. Once recognized as individual components, they will be combined to form character i. This method has both advantage and disadvantage. The advantage is that the classes which need to be recognized are minimum in this case. On the other hand, the disadvantage being the requirement of a labeled dataset and getting such a large amount of dataset is not an easy task.

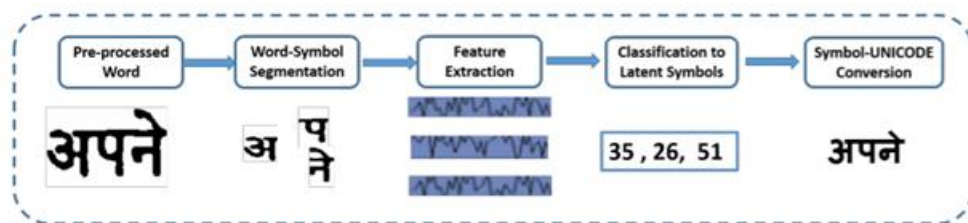


Fig. 2.1: General Character Recognition System Pipeline [2]

2.2.2 Character Classification using Pattern Classification

2.2.2.1 SVM - Support Vector Machine

Support Vector Machines is a supervised machine learning algorithm which requires labeled data to create and train a model that could align well to the actual function. It classifies two classes by

the means of adjusting a hyperplane between those two classes. We then use the created model to label the test data. Testing the model on test data helps us find the accuracy of the model so created. Suppose we have n -dimensional input vector, the aim of SVM is to find $n-1$ hyperplanes that can identify the training classes. SVM try to find that hyperplane whose margin is the highest between the classes. This is demonstrated in Fig 2.2.2. In this case, it is assumed that the input data classes can be linearly classified into n dimension but if they cannot be linearly classified then the input is transformed into a higher dimension in which the classes can be separated linearly.

Generally, the scenario is not like the one mentioned above. That is, usually, in a problem, there are multiple classes that one needs to deal with. In such a situation multi-class support vector machine comes into play. This works by diving the multi-class SVM into many two-class problems. There are many ways to do so, but the commonly used ways are as follows:--

- One against All: The classifier is trained to differentiate between one class and the remaining other classes together. While testing, the classifier which has the maximum output function is taken as the test sample class.
- One against One: For every pair of classes a set of classifiers is used between them. While testing the class with the maximum votes is assigned as the sample test class.
- DAG-SVM: It is a modified version of the above-mentioned approach. Here, the classifier is stored as a Directed Acyclic Graph (DAG).

Though the classes created from the above two approaches are similar, in DAGSVM, the testing time is much less.

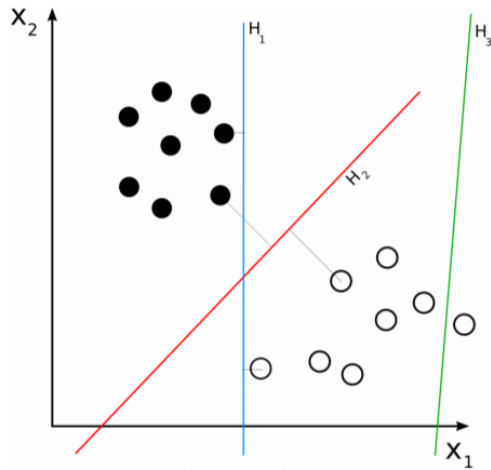


Fig. 2.2.1: Multiple Ways in Which the Classes can be Separated

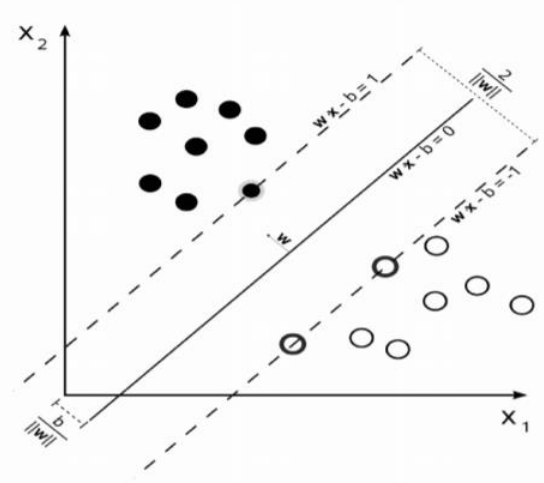


Fig. 2.2.2: Optimal Line Separating the Classes

2.2.2.2 Multilayer Perceptron

Artificial Neural Network (ANN) as it sounds is a human brain like a neural network that is based on the learning process. Like a human brain cell, a cell has a body that consists of a nucleus. A tail like structure called dendrites is attached to the cell body. Dendrites have axons attached to them which help with receiving and sending information. The information is passed between neurons as electrical signals. These electrical signals carry an electrical potential value and once the value of electrical potential reaches a threshold value, it is said to be activated. Similarly, ANNs are also

capable of processing the information and learning from that information by having all the learning and processing phases or the units together. There is a weight associated with each link of the neurons that help in long term storage. While learning the weights are updated continuously depending on various factors which help in generating a model close to the one desired.

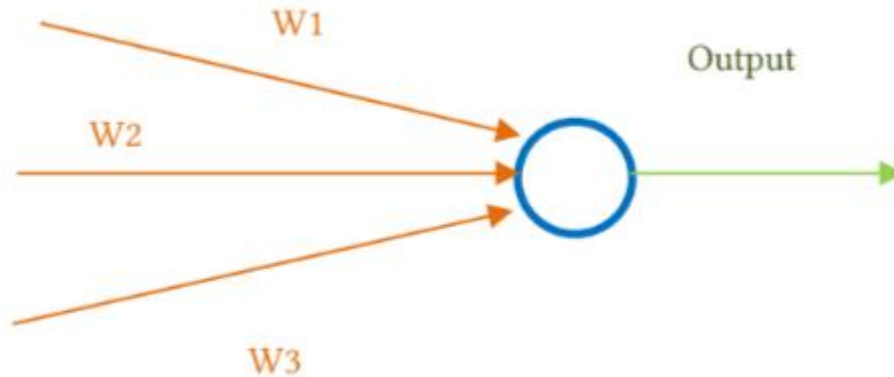


Fig. 2.3: Artificial Neuron

There are two ways of operation of a neuron: a training mode and a validation or testing mode. During training, the neuron is made to get active on a particular input of patterns. On the other hand, when the same neuron is put to test and a known input is received then it's corresponding output is set as the current output, but if the input is not known to the neuron, it is then decided by the neuron whether that input will cause its activation or not. It is determined by using an activation function such as linear, RELU, logistic, etc.

A multilayer perceptron has many nodes in each layer and all these nodes are interconnected to each other. For example, 3-layer perceptron will have one hidden layer and one layer each for input and output. These neural networks are also termed as feedforward neural networks as the data flows in just one direction. An activation function is present in each hidden layer and is the reason behind the transformation of summed activation at a layer. Multilayer perceptrons are a

better fit for classification as their output is dependent on the current input and is not related to any data from previous stages.

2.2.2.3 Hidden Markov Model

Hidden Markov Model (HMM) is a model in which the system whose model is being created is assumed to be a chain of events where it's probability is dependent on the previous event's state. This algorithm is useful for many recognition applications such as handwritten recognition, voice recognition, etc. Some of the elements of HMM models are as follows:

- States (S): States of a model are hidden but they play the main role in the working of a model.
- Distinct observation (V): observations are the output of a system.
- Distributed transition probability of a state,

$$A = P[q_{t+1} = S_j | q_t = S_i], 1 \leq i, j \leq N \dots [23]$$

- The probability of distribution symbol in state j,

$$B = P[V_k^t | q_t = S_j, 1 \leq j \leq N; 1 \leq k \leq M] \dots [23]$$

Complete parameter set of a model is denoted by lambda, $\lambda = (A, B, \pi)$. when using HMM as a recognizer our aim is to maximize the probability of O given λ . For each word, a different HMM is created. For training, all the different HMMs are taken into consideration as an input. Our aim in this is to adjust A, B, and π so as to maximize the probability of O given λ .

The model so formed may not be the best one, hence we try to improve upon this model and try to generate better outputs. The need is to have such a state of the model that the whole model is optimized. This can be done by increasing the number of states. In this approach, it might be

possible that we have multiple optimal state sequences, in such a scenario the best sequence would be the one that maximizes the probability of Q given O and λ . An algorithm named Viterbi algorithm is used to get the optimal sequence. It is a dynamic programming based approach to find the best sequence.

To test, the word is taken as the input and is sent to every model. The output which almost clones the desired model is the one that is selected. The drawback of this method is that it takes a lot of time to traverse through every possible state and hence takes a long time to compute.

2.2.2.4 Recurrent Neural Network (RNN)

RNN is a neural network that can be categorized under the feedforward neural network (FFNN). In this, the connection between the neurons can be from the previous layers to the layers ahead and also from the layers ahead to previous layers. In other words, it has a forward flow as well as a backward flow. RNNs is of great interest to a lot of people for the purpose of sequence classification as the human brain is also considered to be recurrent.

The two different types of FFNN are shown in Fig 2.4. and Fig 2.5 A basic FFNN consisting of an input layer, a hidden layer, and an output layer is shown in Fig 2.4. Similarly, the hidden layer can have an extra feedback connection to the same layer. This will help in storing the node's state before getting to the initial state. The previous value of the node is stored in the memory. Hence, it makes it possible for the node to know the context of the pattern. This type of FFNN is shown in Fig 2.5.

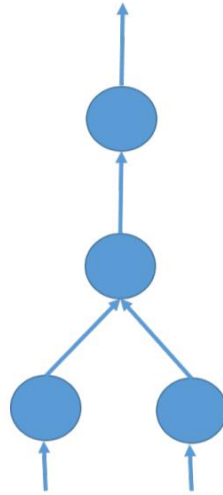


Fig. 2.4: FFNN With an Input Layer, One Hidden Layer, and an Output Layer

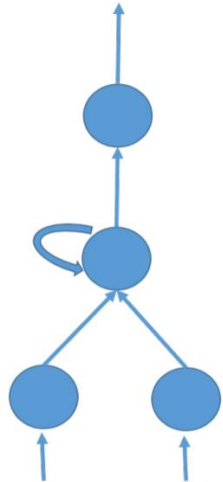


Fig. 2.5: Hidden Layer Having Additional Feedback

Later, it was realized that the process can be improved upon by saving both the context instead of just having the future context saved. That is, having both future and past context will help in getting a better result instead of having just the context of the future. This led to the introduction of Bidirectional Recurrent Neural Network (BRNN). The architecture of BRNN is slightly different as compared to that of an FFNN as it has two hidden layers between an input and the output layer. Both of these hidden layers are attached to the output layer providing it the context of both the past and the future.

2.3 Character Recognition for Documents written in Devanagari Script

OCRs for Indic scripts like Devanagari, Tamil, etc. have been in the market and research place from a long period of time now but they have not been able to attain accuracy comparable to the OCRs for the English language. There are a lot of reason behind this, including the script complexity, large pool of symbols, word segmentation, etc.

Generally, the error and issue in the existing OCRs occur because of two main parts. The first one being the division of words into individual characters and their respective symbols. The second being the generation of valid UNICODE from the outputs obtained through a classifier.

2.3.1 Why this research?

Even though a lot of research has been done in this field, none of the previous methods were successful enough in providing a working prototype. In most of the cases, the evaluation was conducted by the author on their own collection of sample set which might not be enough to tell the accuracy correctly.

The aim of this project and research is to provide a model that is robust enough to be successful in real-life documents.

3. Tools

3.1 Python

Python is the main language that we have used for writing the implementation of this project. The program components consist of preprocessing of the data, creating a classification model and a neural network. Python is really a powerful tool that has support for a lot of libraries and extensions. For example, one of the libraries used in this project is Numpy which provides support for high-level math functions on matrices and multidimensional arrays. This library is useful for the creation of a simple neural network as it helps with the multiplication of the sigmoid functions and large dimensional arrays. Other libraries used in this project are Keras, OpenCV, Tensorflow, matplotlib, etc.

- Keras: It is a python API used for the high-level neural network. It uses Tensorflow as its base and is written in python. It enables fast experimentation. It is a user-friendly and modular library that allows models to be understood as a graph or a sequence. It helps in minimizing the number of actions required by the user for use cases that are common. The error feedback mechanism provided by Keras is an added plus. This is the reason behind Keras being easy to use. Even though it is easy to use that doesn't mean that the flexibility provided by the library is affected because of that. Keras is integrated with Tensorflow, which is a low-level learning library language. Hence giving the user freedom to implement anything on the base language.
- Tensorflow: It is an opensource framework that consists of tools integrated into a flexible ecosystem, libraries, and other resources which are useful for the developers to easily create and deploy machine learning based applications. It provides abstraction at multiple levels and allows the creation and training of models using high-level API Keras. It allows the

user to train and deploy the models easily irrespective of the language and platform. For this project, we have used python as the language for implementing our model.

- **Numpy:** Numpy is a python module that provides support for high-level math functions on matrices and multidimensional arrays. This library is useful for the creation of a simple neural network as it helps with the multiplication of the sigmoid functions and large dimensional arrays. It allows defining arbitrary datatypes and can be efficiently used as an n-dimensional container of the generic data. This allows numpy to integrate and combine with a different type of databases quickly.
- **OpenCV:** Python is slower as compared to its other counterparts like C and C++ but it has a feature to be extended using C and C++, which means that these faster languages can be used to write computationally extensive code and then a python wrapper class is created which can be used as a python module. This gives us advantages such as fast performing code which can be used with easy to use language Python. OpenCV is one such module which allows the user to take advantage of multicore processing and is written in C++. It is used for computer vision problems. OpenCV is generally combined by Numpy library as the operations available in Numpy can be combined with it. It is one of the most optimized library that can be used for mathematical operations. OpenCV is a solution for solving computer vision problems. For this project, OpenCV is used to track and trace the hand movement of the user and to covert those traced pixels into an image. This helps us in providing real-time recognition feature to the project where the user writes in front of a webcam and then that written character is understood and classified by our model.

4. Implementation

4.1 Dataset

Dataset is the main thing behind any classification project. Images are required to train the model. Training data is labeled which helps in teaching AI models, whereas on the other hand test dataset is used to evaluate the trained model. Test data is basically a dataset which the model has not seen before. For this project, I have used a combination of the pre-existing dataset and a dataset that I created. I have used is Devanagari Handwritten Character Dataset (DHCD), which was created as a part of research at a University in Nepal and is available online at the University of California Irvine Machine Learning Repository. There are three different categories in the dataset and the samples in the dataset are collected from 40 people. The collected data was in the form of separate fields and were then cropped for each character boundary. The categories of the dataset are as follows:

- Numerical: There are 10 classes, ranging from 0 to 9, in this category of the dataset. For each of the digit from 0 to 9, there are 2000 samples in the dataset. The classes are as follows:

0	०
1	१
2	२
3	३
4	४
5	५
6	६
7	७
8	८
9	९

Table 4.1: numerical class

- Vowels: There are 12 classes of vowels consisting of 2000 samples per class. The classes are shown in table

1	अ
2	आ
3	इ
4	ई
5	उ
6	ऊ
7	ए
8	ऐ
9	ओ
10	औ
11	अं
12	अः

Table 4.2: Vowels Class

- Consonants: it is the biggest category in the Devanagari handwritten character dataset. There are 36 consonants and hence 36 classes with 2000 samples in each class. The different types of classes in consonants are shown in table 4.3.

1	क
2	ख
3	ग
4	घ
5	ङ
6	च
7	छ
8	ज
9	झ
10	ञ
11	ट
12	ठ
13	ड
14	ढ
15	ण

16	त
17	थ
18	द
19	ध
20	न
21	प
22	फ
23	ब
24	भ
25	म
26	य
27	र
28	ल
29	व
30	श
31	ष
32	स
33	ह
34	क्ष
35	त्र
36	ज्ञ

Table 4.3: Consonants Class

DHCD consists of ninety-two thousand images that were generated from the images of characters written by individuals. Each individual has a different way of writing and hence provide a huge variation in the style of writing the characters. The dataset is divided into training and testing with a split of 85 to 15 respectively. There are 78200 images in the training set while the testing dataset consists of 13800 images.

The images are 32x32 pixels in dimension, but the characters are centered within 28x28 pixels. There is a padding of 2 pixels on each side of the image. The images are greyscaled and the intensity of images in the dataset is inverted which makes the character white in color on a black colored background. The details of the dataset are as follows:

- Data type: Greyscale images (.png format)
- Classes: 46 (10 digits and 36 characters)
- Dataset size: 92000 sample images (2000 samples for each class)
- Training data: 78200
- Test data: 13200
- Details/ Description:
 - Size of image: 32x32 pixels
 - Actual Size: 28x28 pixels
 - Padding: 2 pixels each side
 - Color:
 - Background: Black (0)
 - Text: White (255)

4.2 Challenges

In Devanagari script, there are many characters that are quite similar to each other in terms of structure. In these characters, there is a slight difference like a difference of a dot, a horizontal line connecting the Shiro-Rekha or the headline, etc. Table 4.4 shows an example of such characters:







		Difference being horizontal line at top.
		Difference being presence of single dot on right side.
		Difference being presence of small circle and small down stroke line

Table 4.4: Structural difference of characters

As shown in the table above, there are characters that have barely a minute difference in their structure and this sometimes makes it difficult for the model to differentiate. For example, the first row of table 4.4, there is a difference of horizontal line at the top. Similarly, for the second example in the table, there is a slight difference of a dot on the right-hand side of the character. Both the characters are almost the same in structure except the dot. In the third case as well, there is a difference of a stroke line in one of the characters, rest their appearance is almost identical.

The problem of similar looking characters gets intensified when the characters are handwritten and due to the varied styles of writing, people tend to write different characters in quite an identical manner. In such a scenario it becomes really difficult for a system to tell the difference. An example of the same can be seen in table 4.5.

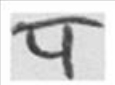

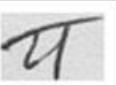





			
			

Table 4.5: Similar Looking Characters Due to the Handwriting Style

Because of the challenges mentioned above and the absence of image samples for the vowels, there was a need for more data. Hence, we collected some more data to reduce the error rate because of these things and to provide support for vowel recognition as well. The data collection task was kind of manual. We used scanned documents written in Hindi and converted them into an image file of JPEG format. Each of these images was looked into and the characters were cropped from the document image. Each of the images was then resized to 32x32 pixels where the character size is 28x28 pixels. Padding of 2 pixels is added to each side of the character image. We added 4600

sample images belonging to 46 classes. That means each class was added with 100 images each. The model was trained and tested on the combination of both the dataset.

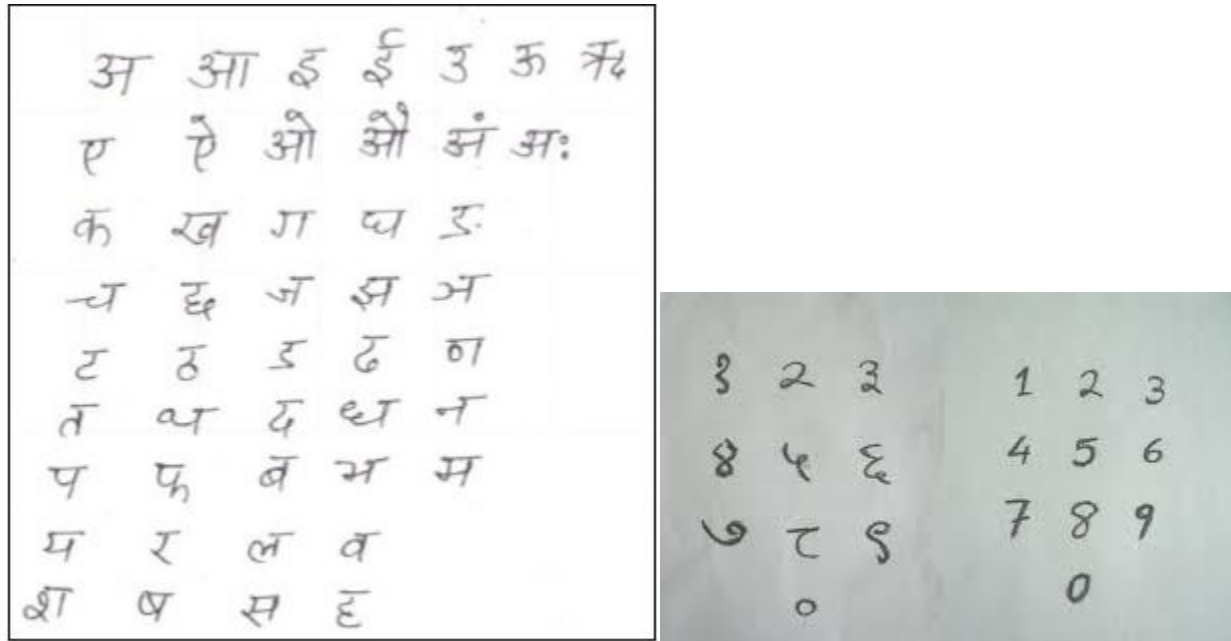


Fig. 4.1: Scanned and Greyscaled Documents Written in Hindi

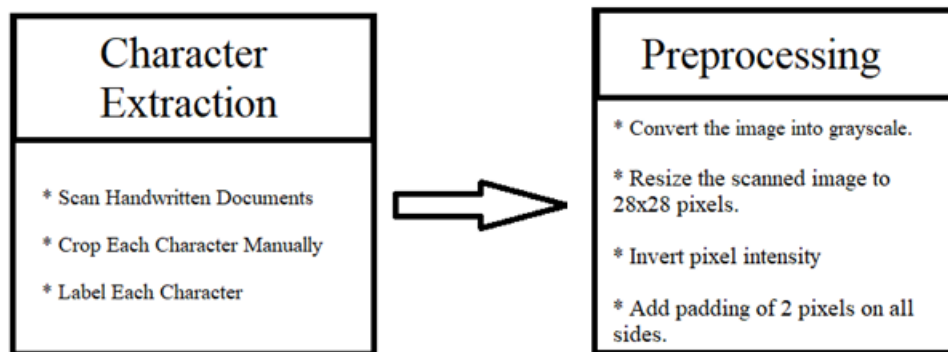


Fig. 4.2: Process of Dataset Creation

Fig 4.2 Shows the process of creation of additional dataset. The process started by scanning the handwritten Devanagari character documents. A sample of the scanned document is shown in Fig 4.1. The characters in the document were converted into grayscale and then were cropped and

labeled manually. Cropped and labeled images of the characters were then resized to the same size as the rest of the images in the other dataset, that is, 32x32 pixels. Of this size, the character image occupied 28x28 pixels while the padding occupied 2 pixels on each side of the image. After this the intensity of the image was inverted, that is, the character being white and the background being black, to match the format of images that are already there in the dataset.

5. Implementation

There are many machine learning classification algorithms that are used for the purpose of recognition and classification. To start with the implementation of the project, we selected some of the machine learning algorithms that were used by various researchers in previous work. We used these algorithms to check their performance for the same dataset. The used algorithms are as follows:

- **Ridge Classifier:**

It helps in solving linear regression / ordinary least squares problem by having a penalty or weight associated with the coefficient size. It does that by minimizing the residual sum of squares.

$$\min_w \|X_w - y\|_2^2 + \alpha \|w\|_2^2$$

Here, the complexity parameter α controls the amount of shrinkage which is directly proportional to the α . The greater the value of complexity parameter the greater will be the shrinkage and hence making coefficients healthy and robust to collinearity. [20]

- **Bernoulli Naïve Bayes:**

For data which is distributed according to Bernoulli multivariate distribution, naïve Bayes training and classification can be implemented using BernoulliNB. A multivariate Bernoulli distribution means that each feature is assumed to be binary valued. Therefore, it requires the samples to be presented as a binary valued feature vector. The formula on which Bernoulli's decision is based is as follows:

$$P(x_i|y) = P(i|y)x_i + (1-P(i|y))(1-x_i) \quad [20]$$

In the case of this project, it uses the word occurrence vector to train and use this classifier. BernoulliNB is suitable for discrete data just like MultinomialNB but the former is designed for binary features while the later works on occurrence count.

- **Gaussian Naïve Bayes (GaussianNB):**

NB is a supervised learning method. It is based on using Bayes theorem with the assumption of naïve of conditional independence. It assumes every pair of features given the value of the class variable. The relationship between the class variable (y) and the feature vector (x) is as follows:

$$P(y|x_1, \dots, x_n) = P(y)P(x_1, \dots, x_n|y) / P(x_1, \dots, x_n) \quad [20]$$

After some simplifications the equation can be written as:

$$y = \arg \max_y P(y) \pi^n P(x_i|y) \quad [20]$$

And $P(y)$ and $P(x_i|y)$ can be estimated by using Maximum A Posteriori (MAP).

GaussianNB is used to implement Gaussian Naïve Bayes classification algorithm. The likelihood of the features is assumed to be Gaussian:

$$P(x_i|y) = 1/(2\pi\sigma_y^2)^{1/2} \exp(-(x_i-\mu_y)^2/2\sigma_y^2)$$

The σ_y and μ_y are estimated using maximum likelihood.

- **DecisionTreeClassifier:**

Decision Trees are a method of doing supervised learning, generally used for classification and regression. The aim of this algorithm is to predict the target value by learning decision rules inferred from the data features. Fig 5.1 shows a plot for a decision tree regression problem.

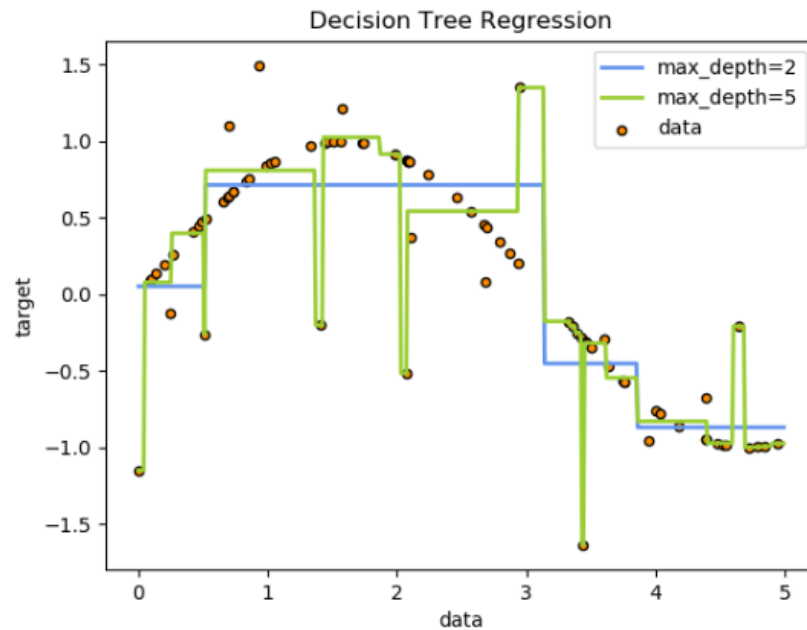


Fig. 5.1: Decision Tree Regression [20]

Trees are easy to understand, interpret and visualize. The data preparation is not that much as compared to another method, which needs their input data to be normalized, blanks removed, etc. The cost of using a tree is logarithmic and they can handle multi-output problems. Even though there are so many positives about the tree classifier, there are some drawbacks as well which includes the creation of over complex trees that do not generalize the data back. A small variation in data results in a different tree generation making it unstable. Fig 5.2 shows the use of decision tree classifier in Python using the sklearn module.

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(iris.data, iris.target)
```

Fig. 5.2: A Code Snippet Showing Tree Classifier on Iris Dataset

- **ExtraTreeClassifier:**

ExtraTreeClassifiers are extremely randomized tree classifiers. They differ from the basic decision tree in the way they are built. To separate the samples of a node into two groups, random splits are drawn for each of `max_features` and random features. The best split among them is chosen and if `max_feature` is 1, this creates a totally random decision tree.

- **KNeighborsClassifier:**

Supervised and unsupervised neighbor based learning method functionality is provided by `sklearn.neighbors`. Supervised learning has two applications which include the classification and regression. Classification is for discrete labeled data while the regression is for continuous labeled data. Regression is the foundation for a lot of learning methods like spectral learning, manifold learning, etc.

The idea behind the nearest neighbor algorithm is to search and find a preset number of samples having less distance from the new point and using these to predict the label. Here the number of samples, that is `k`, is user-defined value. The distance can be any metric measure of the distance such as Euclidean which is the most commonly used distance measure. Even though it is a simple algorithm for classification and regression problems, it has been quite successful in solving these problems like handwritten digits and satellite images. It is usually successful in classification problems where the decision boundary is not regular as it is a non-parametric method.

Using this algorithm on a sample dataset from each class, we calculate the test score, train score and the time taken by the algorithm to compute the results.

K-Neighbor classification is one of the most commonly used methods. Here the choice of k depends on the data. A larger value of k would suppress the effects that happen due to noise but will make the classification boundaries difficult to be distinguished.

- **ExtraTreesClassifier:**

These are extremely randomized tree classes. The splits are computed in a random way.

Among randomly chosen feature candidates, thresholds are drawn at random for each candidate feature and the best among these is selected as the splitting rule.

A meta estimator is implemented by this class that fits a number of randomized decision trees on various subsamples of the dataset and uses averaging to improve the predictive accuracy and control overfitting.

- **RandomForestClassifier:**

In this algorithm, each tree is built from a sample drawn with replacement from the training set. The split selected is the best one among a random subset of features. Due to this, the bias of the forest usually increases but its variance also decreases because of averaging and hence almost compensating for the increase in bias. This yields an overall better model.

For this project, all these algorithms were used to check how well they perform with the dataset that we are using in this project. We find out the train and test score. The classifier functions were taken as an array and for each of the value of that array was passed with the data sample and scores were generated.

The results obtained by the algorithms are shown below in Fig 5.3:

	Classifier	Fit_Time	Score_Time	Test_Score	Train_Score
0	RidgeClassifier	0.694055	0.018355	0.423384	0.841250
1	BernoulliNB	0.115145	0.053828	0.512742	0.546638
2	GaussianNB	0.164446	1.247284	0.396972	0.433721
3	ExtraTreeClassifier	0.098612	0.010459	0.311093	1.000000
4	DecisionTreeClassifier	4.072366	0.007965	0.365119	1.000000
5	NearestCentroid	0.100165	0.027360	0.530247	0.568857
6	KNeighborsClassifier	0.668098	35.745628	0.721442	0.841367
7	ExtraTreesClassifier	0.534121	0.029591	0.574804	1.000000
8	RandomForestClassifier	0.904359	0.027685	0.543047	0.997936

Fig. 5.3: Scores Obtained from Various ML Algorithms

We tried to further improve the results obtained through these machine learning algorithms. We selected the top three performers, that is, KNeighborsClassifier, ExtraTreesClassifier (Extremely Randomized Decision Forest Classification) and Random Forest Classification. On these algorithms, we applied a Grid Search to get the best parameters and scores of these algorithms.

Hyperparameter tuning was done to find the best value for the parameters and using those values of the parameters the accuracy score plots were plotted for the best three algorithms, that is, K Nearest Neighbors, Random Forest, and Extra Tree Classifier. We also plotted the accuracy score obtained from these algorithms which are shown as follows:

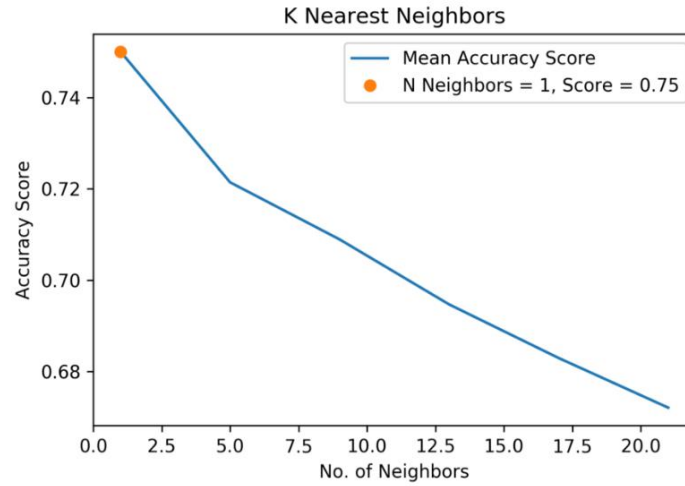


Fig. 5.4: Plot for Accuracy Score of K Nearest Neighbor Classifier

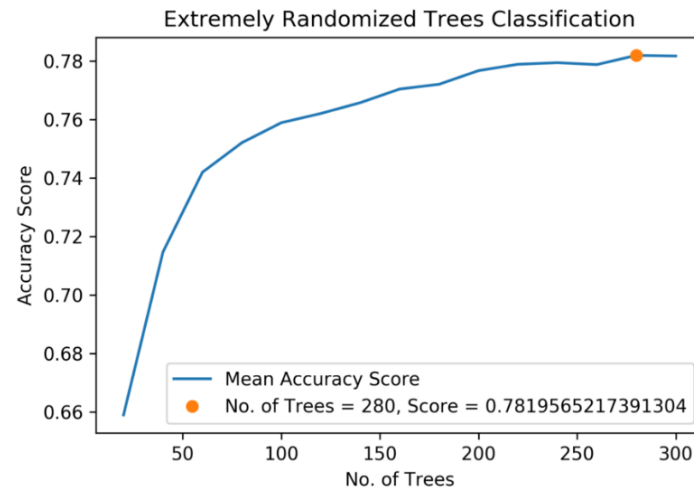


Fig. 5.5: Plot for Accuracy Score of Extremely Randomized Trees Classifier/ Extra Trees Classifier

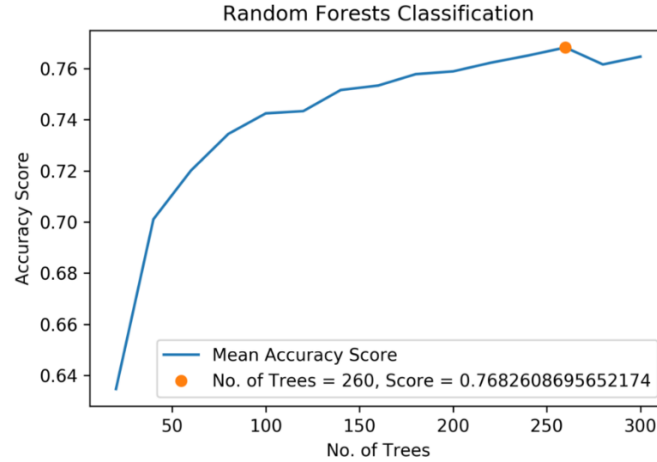


Fig. 5.6: Plot for Accuracy Score of Random Forests Classifier

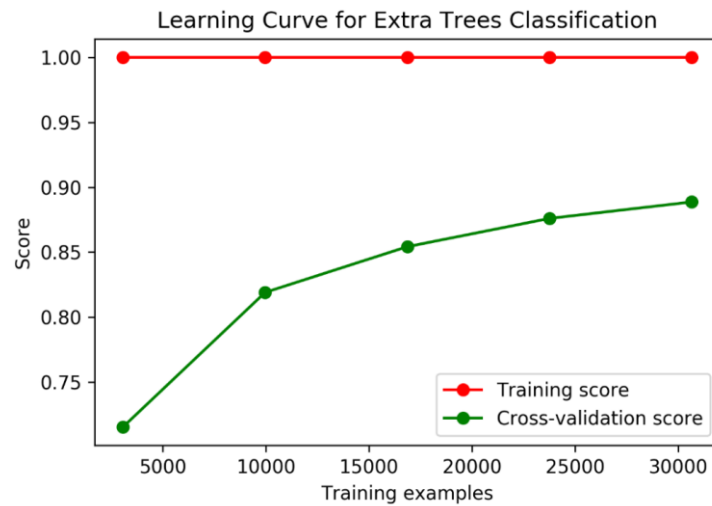


Fig. 5.7: Plot for Learning Curve of Extremely Randomized Trees Classifier/ Extra Trees Classifier

As the main implementation, we built a CNN model and trained it to classify the Devanagari Characters. The model takes in input in the form of video frames which are captured while recognizing the hand gesture movements of the user.

A CNN is composed of many layers, of which the first and last being the input and output layer, while the other being hidden layers. Generally, the hidden layers are convolution layer, pooling layer, normalization layer, etc.

Convolution operations are applied by the convolution layer to input. Each neuron is responsible for processing. The result received after processing it passed on to the next layer which matches the output of every neuron to visual stimuli. Each neuron processes that belong to its field only. Though a dense and fully connected FFNN can be used to learn features and be used to classify data, that is not practical to do so. As the number of images is too high, the associated input size is variable. For example, if we have a fully connected layer where the input is in the form of images of size 100x100 will have 10000 weights for each neuron in the layer next to it. CNN with the help of convolution operations brings a solution to this issue by reducing the number of free parameters and by letting the network to be deeper with few parameters. That is, whatever may be the size of the image a tile of the 5x5 region each with the same value of shared weights, require 25 learnable parameters [15]. by doing this the problem of exploding or vanishing gradients in training of a neural network having many layers is resolved with backpropagation.

In a fully connected layer, each neuron in one layer is connected to every neuron in another layer which makes it sound more like a multi-layer perceptron.

Convolution network generally has a pooling layer. The role of a pooling layer is to combine outputs from the cluster of neurons at the previous layer and into a single neuron in the next layer. For example, In Max pooling, the maximum value is selected from the cluster/grid of neuron and is used as an input to the next layer neuron, while average pooling uses the average value of neurons of a cluster at the previous layer.

The architecture of our implementation is as follows consists of two convolution layers, two maxpooling layers which are followed by layers performing softmax regression. A detailed version of architecture is as follows:

CONV2D → MAXPOOL → CONV2D → MAXPOOL → FC → SOFTMAX → CLASSIFY

The main idea behind this model was to have a convolution layer followed by a pooling layer. Each convolution layer applies convolution operations on the neurons which help in finding the features. The main function of a convolution layer in a model is to find out features in the image. which is followed a pooling layer. The role of a pooling layer is to downsize the image and hence downsizing the number of features. The reduced number of features are further passed to a convolution layer that again applies convolution operations and tries to find out features in the downsized image. This process continues and finally, the output layer outputs the result.

For this project, several CNN models were trained to monitor the convergence of the model and find the best hyperparameters. We trained the convolution layer for various values of features layer size. We tried 3x3, 5x5 and 7x7 sizes of the kernel. The architecture for the neural network is shown in Fig 5.8 and the training of the model is shown in Fig 5.8.

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d_1 (MaxPooling2)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 10, 10, 64)	51264
max_pooling2d_2 (MaxPooling2)	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 37)	9509
=====		
Total params: 61,605		
Trainable params: 61,605		
Non-trainable params: 0		
=====		

Fig. 5.8: Structure of Network

The model consists of a convolution layer which is followed by a max pooling layer. These two layers are repeated once more. The output from these layers is then pushed to layer performing softmax regression and the result for that is moved to the output layer. We trained the model for a number of epochs until the value got stagnant. The training process is shown in Fig 5.10 below.

```

68224/70000 [=====>.] - ETA: 1s - loss: 2.0831 - acc: 0.4759
68288/70000 [=====>.] - ETA: 1s - loss: 2.0818 - acc: 0.4762
68352/70000 [=====>.] - ETA: 1s - loss: 2.0807 - acc: 0.4765
68416/70000 [=====>.] - ETA: 1s - loss: 2.0794 - acc: 0.4768
68480/70000 [=====>.] - ETA: 1s - loss: 2.0781 - acc: 0.4772
68544/70000 [=====>.] - ETA: 1s - loss: 2.0769 - acc: 0.4775
68608/70000 [=====>.] - ETA: 1s - loss: 2.0755 - acc: 0.4778
68672/70000 [=====>.] - ETA: 1s - loss: 2.0742 - acc: 0.4782
68736/70000 [=====>.] - ETA: 1s - loss: 2.0728 - acc: 0.4786
68800/70000 [=====>.] - ETA: 1s - loss: 2.0717 - acc: 0.4788
68864/70000 [=====>.] - ETA: 1s - loss: 2.0704 - acc: 0.4792
68928/70000 [=====>.] - ETA: 1s - loss: 2.0690 - acc: 0.4795
68992/70000 [=====>.] - ETA: 1s - loss: 2.0678 - acc: 0.4798
69056/70000 [=====>.] - ETA: 1s - loss: 2.0667 - acc: 0.4801
69120/70000 [=====>.] - ETA: 0s - loss: 2.0654 - acc: 0.4804
69184/70000 [=====>.] - ETA: 0s - loss: 2.0640 - acc: 0.4808
69248/70000 [=====>.] - ETA: 0s - loss: 2.0631 - acc: 0.4810
69312/70000 [=====>.] - ETA: 0s - loss: 2.0618 - acc: 0.4814
69376/70000 [=====>.] - ETA: 0s - loss: 2.0605 - acc: 0.4817
69440/70000 [=====>.] - ETA: 0s - loss: 2.0591 - acc: 0.4821
69504/70000 [=====>.] - ETA: 0s - loss: 2.0578 - acc: 0.4825
69568/70000 [=====>.] - ETA: 0s - loss: 2.0566 - acc: 0.4828
69632/70000 [=====>.] - ETA: 0s - loss: 2.0554 - acc: 0.4831
69696/70000 [=====>.] - ETA: 0s - loss: 2.0540 - acc: 0.4835
69760/70000 [=====>.] - ETA: 0s - loss: 2.0526 - acc: 0.4839
69824/70000 [=====>.] - ETA: 0s - loss: 2.0513 - acc: 0.4842
69888/70000 [=====>.] - ETA: 0s - loss: 2.0500 - acc: 0.4845
69952/70000 [=====>.] - ETA: 0s - loss: 2.0488 - acc: 0.4849
70000/70000 [=====] - 76s 1ms/step - loss: 2.0479 - acc: 0.4851 - val_loss: 0.7023 - val_acc: 0.8365

```

Fig. 5.9: Training of a CNN

Hand Movement Recognition

This above model uses input image formed as a result of tracked hand movements by the cv2 module of Python. It recognizes the difference in color of the background and the pen held in hand. The hand movements are read frame by frame in real time from webcam using cv2.VideoCapture() method. The color to be recognized is defined in the code. In this, we used a blue colored pen and hence initialized the upper and lower bound of that color beforehand. We try to find out contours. Once the contours are fetched, it is smoothened using a series of image operations using cv2 methods such as erode, morph and dilate. Then the center of the contour is used to draw on the screen the center moves along with the hand movements. The drawing formed is displayed both on frame and blackboard. The one drawn on the frame is used for external display while the other one is passed to the model for processing.

The model, on receiving the image preprocesses. Preprocessing is done to ensure that the format of image received is same as that of the images on which the model was trained. Preprocessing includes steps like changing the image size, color scheme and inverting pixel intensity. After the preprocessing the new formed image is passed to the CNN, which predicts the character as an output of the model.

6. Result

An experiment was done on a dataset consisting of 46 classes, each having 2000 image samples. This totals to 92000 images in total that we worked on. The dataset was divided randomly into 85 to 15 ratio for training and testing respectively.

For the first part of the implementation, we implemented various machine learning algorithms which were used by researchers for their research on this dataset to check their performance with respect to this dataset. The result obtained were as follows:

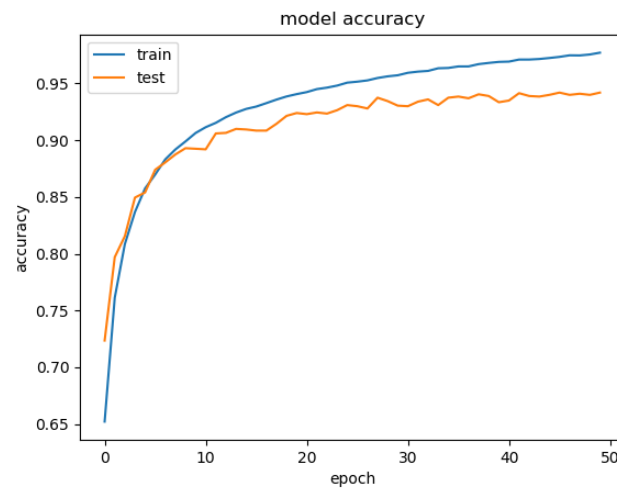
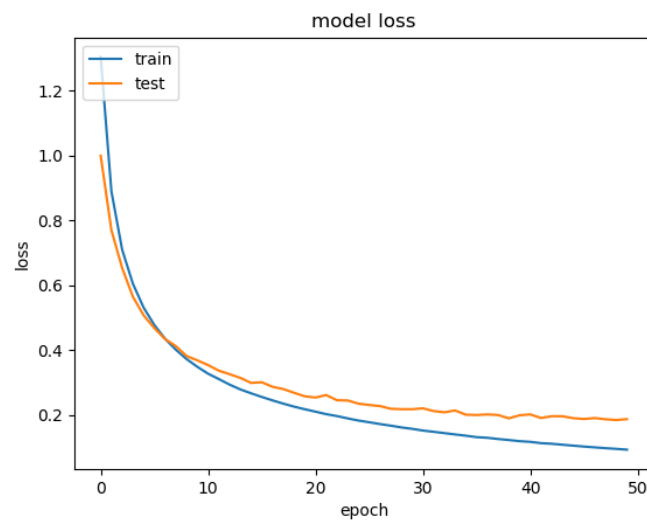
	Classifier	Test Score	Train Score
0	RidgeClassifier	0.423384	0.84125
1	BernoulliNB	0.512742	0.546638
2	GaussianNB	0.39672	0.433721
3	ExtraTreeClassifier	0.311093	1
4	DecisionTreeClassifier	0.365119	1
5	NearestCentroid	0.530247	0.568857
6	KNeighborsClassifier	0.721442	0.841367
7	ExtraTreeClassifier	0.574804	1
8	RandomForestClassifier	0.543047	0.997936

Table 6.1: Result of Machine Learning Algorithms

After tuning the parameters for the best three machine learning algorithms out of these. The previous researchers were able to attain an accuracy of 96.4% using ExtraTreeClassifier.

Then we implemented a convolution network and tried it with various combinations of parameters. The first set of experiment was done by changing the size of the kernel in convolution layer. The result obtained from them are as follows:

Structure	#1	#2	#3
Convolution layer	3x3	5x5	7x7
Pooling Layer	2x2	2x2	2x2
Convolution layer	3x3	5x5	7x7
Pooling Layer	2x2	2x2	2x2
Accuracy	92.40%	95.30%	97.25%
Cost (In terms of loss)	7.6%	4.7%	2.75%

Table 6.2: Result of CNN model with different kernel sizes**Fig. 6.1:** Accuracy vs Epoch for model 3x3**Fig. 6.2:** Loss vs Epoch for model 3x3

We plotted an accuracy vs epoch and loss vs epoch graph for the convolution neural network model and found the best performance of this model to be 92.4%. The model with kernel size of 5x5 was able to attain an accuracy of 95.30, whereas the model with kernel size of 7x7 was able to attain an accuracy of 97.25%.

Fig 6.3 and Fig 6.4 shows the graph plots for accuracy vs epoch and loss vs epoch for the model with kernel size 5x5.

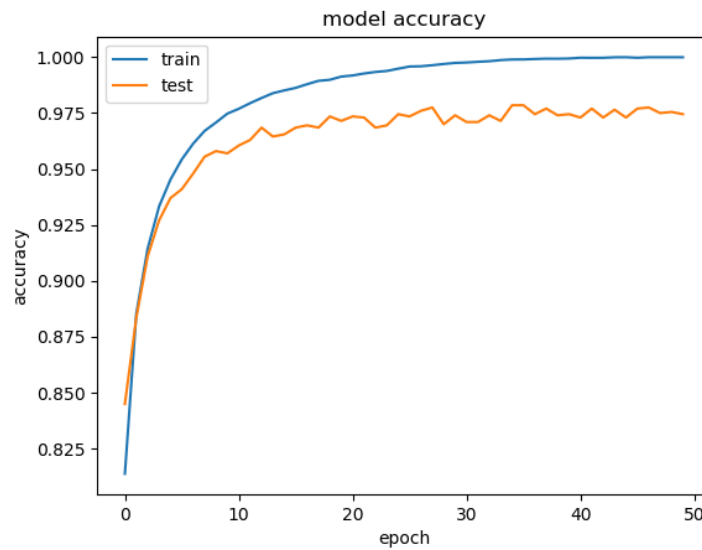


Fig. 6.3: Accuracy vs Epoch for model 5x5

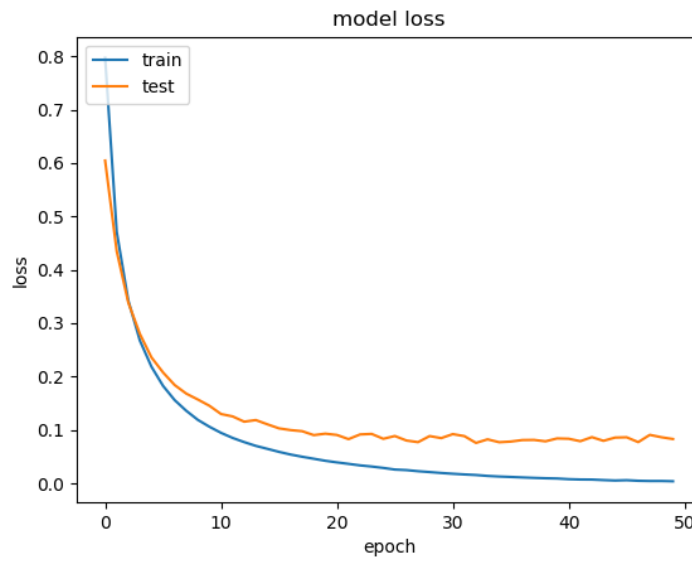


Fig. 6.4: Loss vs Epoch for model 5x5

The smaller the size of the kernel, the finer it will be able to find the features of the object and hence it will take more time to train as compared to the model with larger kernel size.

Another experiment was done by changing the number of filters in the convolution layer of these 3 variations of models. In the earlier models the size of filters were kept as 32 for the first layer and 64 for the second layer. For this experiment the number of filters was changed to 64 and 128.

The result obtained from this experiment is shown as follows:

Structure	#1	#2	#3
Convolution layer	3x3	5x5	7x7
Pooling Layer	2x2	2x2	2x2
Convolution layer	3x3	5x5	7x7
Pooling Layer	2x2	2x2	2x2
Accuracy	94.37%	96.9%	94.9%
Cost (In terms of loss)	5.63%	3.1%	5.1%

Table 6.3: Result of CNN model with different kernel sizes and increased number of filters

The change in the filter size of both the convolution layer helped in improving the accuracy for convolution model with kernel size of 3x3 and 5x5 but not for the model with 7x7 kernel size model. The greater number of filter layers implies that the model will be able to find more features in the input image. Usually each filter layer is used for one feature of the input image. The plots for the above-mentioned experiments are shown as follows:

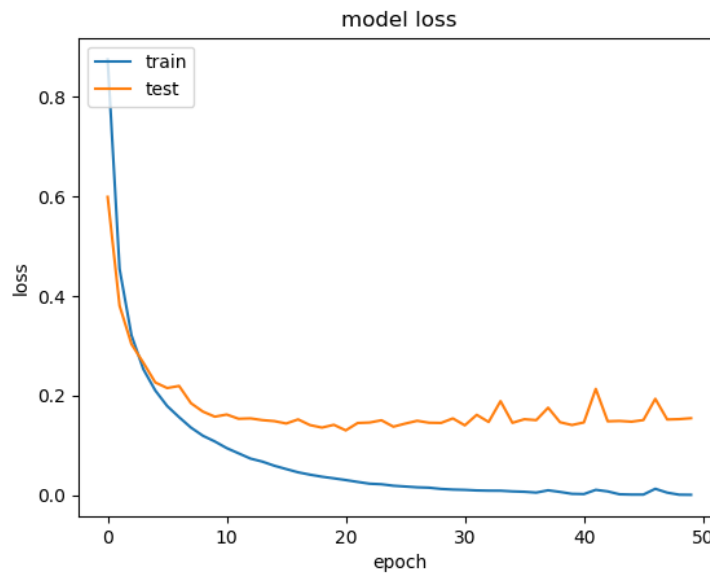


Fig. 6.5: Loss vs Epoch for model 3x3 with increased number of filters

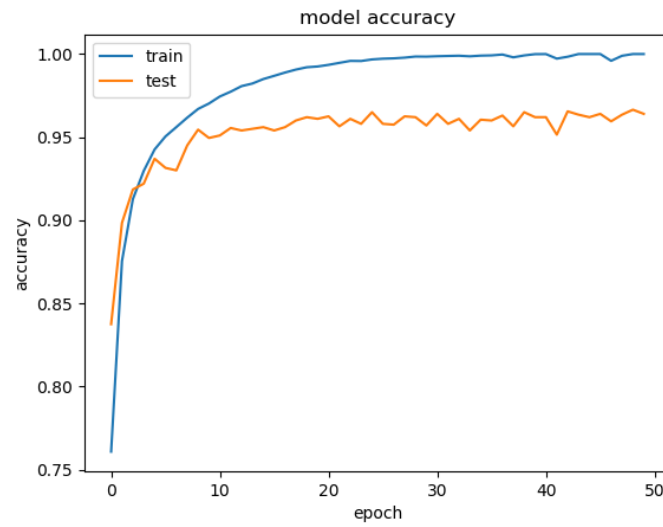


Fig. 6.6: Accuracy vs Epoch for model 3x3 with increased number of filters

Both the models showed improvements in the accuracy. The plots for model with kernel size of 5x5 is shown as follows:

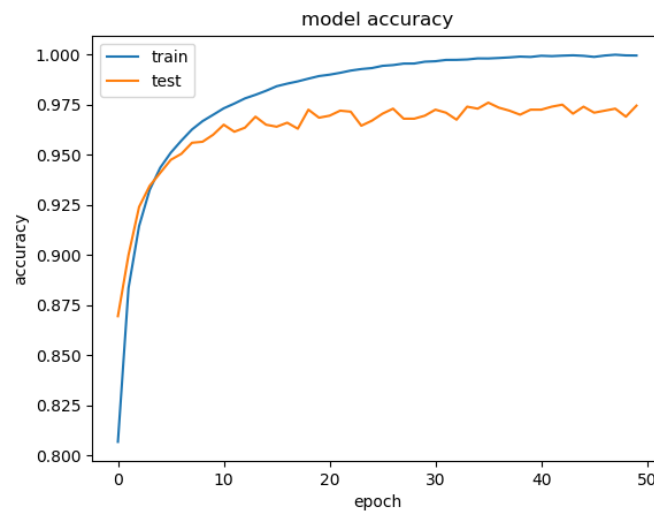


Fig. 6.7: Accuracy vs Epoch for model 5x5 with increased number of filters

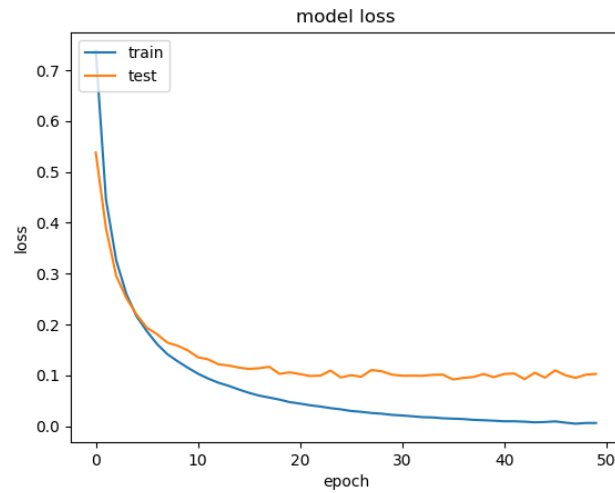


Fig. 6.8: Loss vs Epoch for model 5x5 with increased number of filters

Unlike the other two models, which are mentioned above the model with kernel size 7x7 showed unexpected results. The accuracy instead of improving went down in this case. The plots for the same are shown below:

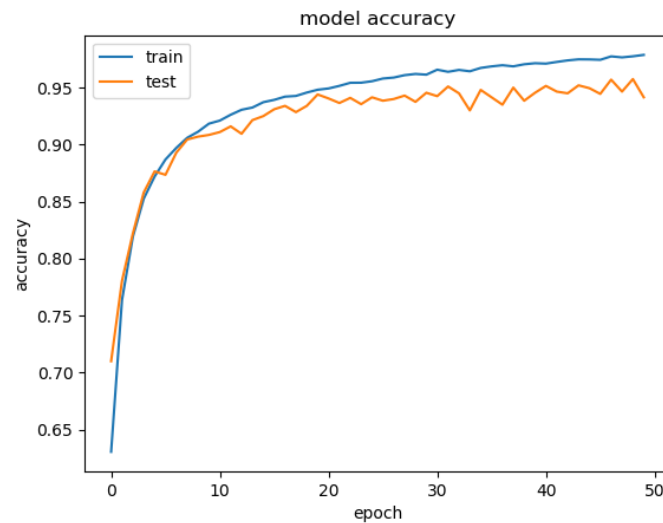


Fig. 6.9: Accuracy vs Epoch for model 7x7 with increased number of filters

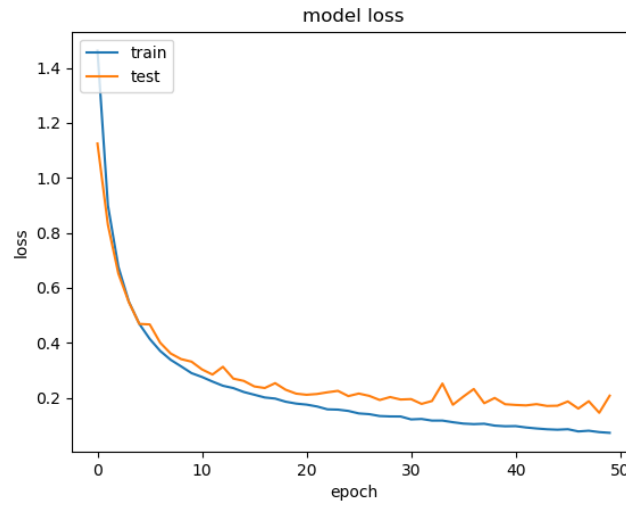


Fig. 6.10: Loss vs Epoch for model 7x7 with increased number of filters

Further an experiment was performed on these convolution models, which included adding a dropout layer after the convolution layer and maxpool layer in each model. Adding a dropout layer to the model helps in reducing the probability of an overfitted model.

The results obtained after the inclusion of dropout layers are as follows:

Structure	#1	#2	#3
Convolution layer	3x3	5x5	7x7
Pooling Layer	2x2	2x2	2x2
Convolution layer	3x3	5x5	7x7
Pooling Layer	2x2	2x2	2x2
Dropout layer	20%	20%	20%
Accuracy	94.67%	96.1%	96.32%
Cost (In terms of loss)	5.33%	3.9%	3.68%

Table 6.4: Result of CNN model with added dropout layer

From the above table it can be seen that the results did not improve much when we added dropout layer. It improved a little bit in case of 3x3 and 7x7 but reduced a bit in case of 5x5 kernel size model.

For the last set of experiment, we cross combined the kernel sizes of the convolution layer. The first one being the smaller one and the other being a larger one. The tried combinations were model with convolution layer of kernel size 3x5 and 5x7. The results and observations of this experiment are as follows:

Structure	#1	#2
Convolution layer	3x3	5x5
Pooling Layer	2x2	2x2
Convolution layer	5x5	7x7
Pooling Layer	2x2	2x2
Dropout layer	20%	20%
Accuracy	97.5%	96.1%
Cost (In terms of loss)	2.5%	3.9%

Table 6.5: Result of CNN model with cross combination of kernel size

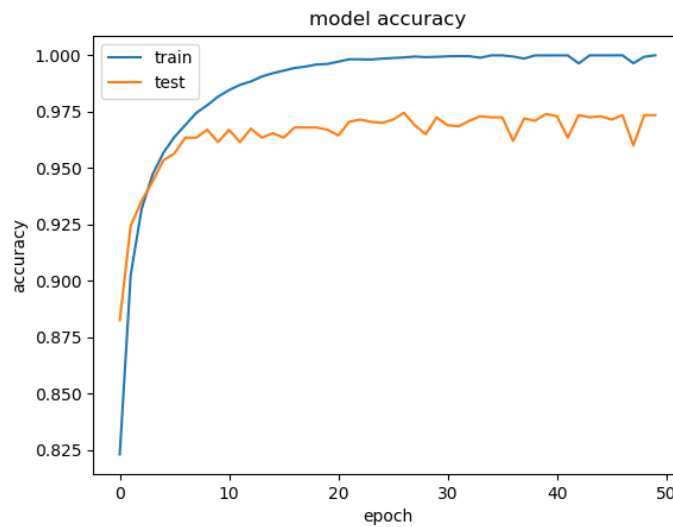


Fig. 6.11: Accuracy vs Epoch for model 3x5

The accuracy achieved by the combination of two different sized convolution layer has increased, when compared to the accuracy achieved by using just the layer of kernel size 3x3.

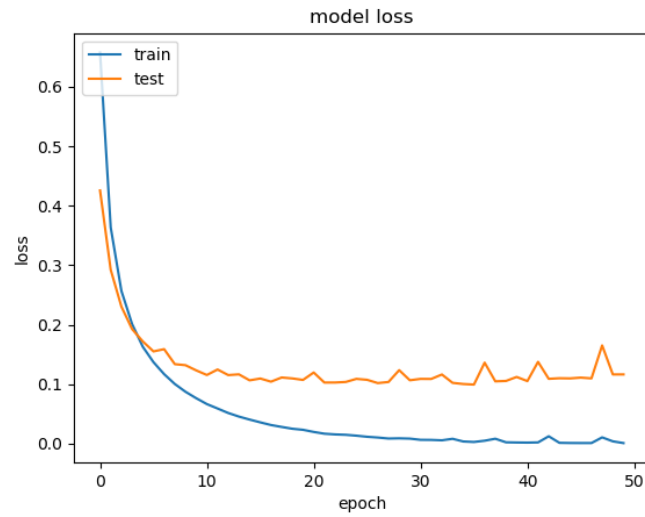


Fig. 6.12: Loss vs Epoch for model 3x5

From above plot, it can be seen that the combination of two different sized convolution layers and resulted in decrease of the loss for the model.

Similarly, the plots for the other model with the combination of layers having kernel size as 3x5 are shown below.

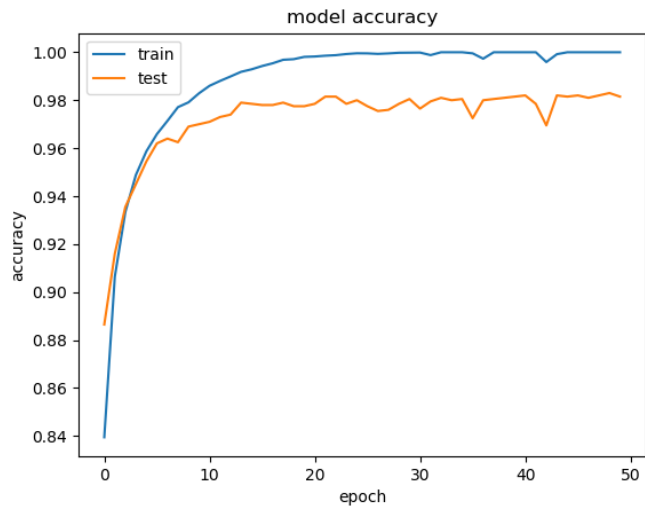


Fig. 6.13: Accuracy vs Epoch for model 5x7

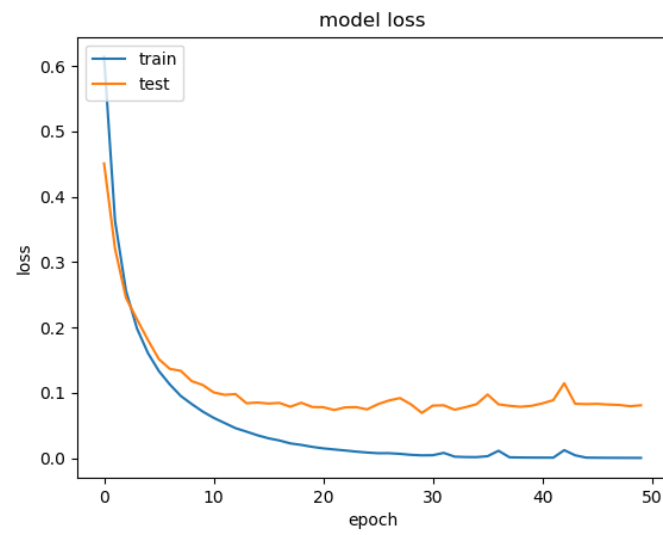


Fig. 6.14: Loss vs Epoch for model 5x7

7. Conclusion

The paper proposed a deep learning based neural network model for Devanagari character recognition. We trained a CNN classification model on DHCD dataset to achieve an accuracy of 97.5%. This is the best accuracy obtained to the best of our knowledge on this dataset. Along with that, we explained our ideation, method, and experiments that could provide more information about Devanagari character recognition. We trained the model with various hyperparameters to see the effect of the change on the accuracy of the model. We also incorporated the use of data augmentation techniques to improve the accuracy of recognition and classification.

We started by implementing a simple convolution neural network having two convolution layers and two max pool layers in the feature extraction section followed by a softmax layer for final classification. The model was able to recognize and classify characters. We improved upon this model by trying and testing different combinations of the parameters in convolution. To attain more accuracy and to be sure of the fact that our model doesn't overfit, we experimented on the model by adding a dropout layer. To our surprise, the dropout layer did not help in increasing the accuracy of the model that much. It was then followed by an experiment where we changed the size of the kernel in the convolution layer. Here the model with kernel size 7x7 was able to perform better than the rest. To build on these results we tried to experiment with the number of filters in the convolution layer. Initially, the number of filters in the convolution layer was set to 32, but for this experiment, we tested the model by changing it to 64, then to 128 and then to a combination of both. That is, we kept the number of filters in the first layer and the second layer as 64 and 128 respectively. For the part of the implementation, we tried to imitate the concept of inception layer in our model. Here, we tested this concept for two different set of values of the kernel layer. For the first set of values, we kept the kernel size of the first convolution layer as 3x3 and the other

layer with a kernel size of 5x5. The other set of values for the kernel size were 5x5 and 7x7 for the first and the second layer respectively.

The previous researchers were able to attain an accuracy of 96.4% on this dataset. This was the best accuracy attained on this model so far. Our experiment with a cross combination of different kernel size was able to beat this record. The model with a kernel size of 3x3 and 5x5 for the first and the second layer respectively was able to attain the accuracy of 97.5%. This was followed by another model having kernel size as 7x7. This model was able to attain an accuracy of 97.25%.

8. Future Work

This project can be extended to recognize and classify words of Devanagari script. We could experiment with the architecture of the CNN model and could also use different networks like Inception network and a network with a varied number of filter and size. A model like VGG16, VGG19 can be tried. We could also use models like ResNet which are generally used while using transfer learning. The recognition and classification application can be extended to recognize words and sentences with the capability of storing them in a file. We could also experiment it on scene images and try to recognize the characters in actual world images.

References

- [1] D. Berchmans and S. S. Kumar, "Optical character recognition: An overview and an insight," *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Kanyakumari, 2014, pp. 1361-1365.
doi: 10.1109/ICCICCT.2014.6993174
- [2] R. M. K. Sinha, "A journey from Indian scripts processing to Indian language processing," *IEEE Ann. Hist. Comput.*, vol. 31, no. 1, pp. 8–31, Jan./Mar. 2009.
- [3] S. Acharya, A. K. Pant and P. K. Gyawali, "Deep learning based large scale handwritten Devanagari character recognition," *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, Kathmandu, 2015, pp. 1-6.
doi: 10.1109/SKIMA.2015.7400041
- [4] Kai Ding, Zhibin Liu, Lianwen Jin, Xinghua Zhu, "A Comparative study of GABOR feature and gradient feature for handwritten chinese character recognition", *International Conference on Wavelet Analysis and Pattern Recognition*, pp. 1182-1186, Beijing, China, 2-4 Nov. 2007
- [5] K. Vijayalakshmi, S. Aparna, G. Gopal, and W. J. Hans, "Handwritten character recognition using diagonal-based feature extraction," *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, 2017, pp. 1178-1181.
doi: 10.1109/WiSPNET.2017.8299949
- [6] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85-117, Jan. 2015. [Online]. Available: <https://arxiv.org/abs/1404.7828>.

- [7] A. Ray, A. Chandawala and S. Chaudhury, "Character Recognition Using Conditional Random Field Based Recognition Engine," *2013 12th International Conference on Document Analysis and Recognition*, Washington, DC, 2013, pp. 18-22.
doi: 10.1109/ICDAR.2013.13
- [8] V. Ganapathy and C. C. H. Lean, "Optical Character Recognition Program for Images of Printed Text using a Neural Network," *2006 IEEE International Conference on Industrial Technology*, Mumbai, 2006, pp. 1171-1176.
doi: 10.1109/ICIT.2006.372591
- [9] V. Bansal and R. M. K. Sinha, "Segmentation of touching and fused Devanagari characters," *Pattern Recognit.*, vol. 35, pp. 875–893, 2002.
- [10] S. Kompalli, S. Setlur, and V. Govindaraju, "Devanagari OCR using a recognition driven segmentation framework and stochastic language models," *Int. J. Document Anal. Recognit.*, vol. 12, pp. 123–138, 2009.
- [11] B. B. Chaudhuri and U. Pal, "Skew angle detection of digitized Indian script documents," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 2, pp. 182–186, Feb. 1997.
- [12] A. K. Das and B. Chanda, "A fast algorithm for skew detection of document images using morphology," *Int. J. Document Anal. Recognit.*, vol. 4, no. 2, pp. 109–114, 2001.
- [13] U. Garain and B. B. Chaudhuri, "Segmentation of touching characters in printed Devanagari and Bangla scripts using fuzzy multifactorial analysis," *IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev.*, vol. 32, no. 4, pp. 449–459, Nov. 2002.
- [14] V. Bansal and R. M. K. Sinha, "Segmentation of touching and fused Devanagari characters," *Pattern Recognit.*, vol. 35, pp. 875–893, 2002.

- [15] S. Kompalli, S. Nayak, S. Setlur, and V. Govindaraju, "Challenges in OCR of Devanagari documents," in Proc. 8th Conf. Document Anal. Recognit., 2005, pp. 1–5.
- [16] U. Garain and B. B. Chaudhuri, "On OCR of degraded documents using fuzzy multifactorial analysis," in Proc. AFSS Int. Conf. Fuzzy Syst. (AFSS-ICFS), 2002, pp. 388–394.
- [17]] A. Bhardwaj, S. Setlur, and V. Govindaraju, "Keyword spotting techniques for Sanskrit documents," in Lecture Notes in Artificial Intelligence 5402, G. Huet, A. Kulkarni, and P. Scharf, Eds. Berlin, Germany: Springer-Verlag, 2009, pp. 403–416.
- [18] H. Zhao, Y. Hu and J. Zhang, "Character Recognition via a Compact Convolutional Neural Network," *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Sydney, NSW, 2017, pp. 1-6.
doi: 10.1109/DICTA.2017.8227414
- [19] Q. Wu, Y. Liu, Q. Li, S. Jin, and F. Li, "The application of deep learning in computer vision," *2017 Chinese Automation Congress (CAC)*, Jinan, 2017, pp. 6522-6527.
doi: 10.1109/CAC.2017.8243952
- [20] K. Vijayalakshmi, S. Aparna, G. Gopal, and W. J. Hans, "Handwritten character recognition using diagonal-based feature extraction," *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, 2017, pp. 1178-1181.
doi: 10.1109/WiSPNET.2017.8299949
- [21] U. Bhattacharya, S.K. Parui, and S. Mondal, "Devanagari and Bangla text extraction from natural scene images," in Proc. 10th Int. Conf. Document Anal. Recognit., 2009
- [22] T. N. Vikram and D. S. Guru, "Appearance based models in document script identification," in Proc. 9th Int. Conf. Document Anal. Recognit., 2007, pp. 709–713.

- [23] R.Jayadevan, Satish R. Kolhe, Pradeep M. Patil, Umapada Pal, “Offline Recognition of Devanagari Script: A Survey”, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 2011.