TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING

Khwopa College of Engineering
Libali, Bhaktapur
Department of Computer Engineering

A REPORT ON

DEVANAGARI CHARACTER RECOGNITION SYSTEM USING CNN

Submitted in partial fulfillment of the requirements for the degree

BACHELOR OF COMPUTER ENGINEERING

Submitted by

Rabit Khaitu	KCE076BCT028
Romik Gosai	KCE076BCT032
Sagar Suwal	KCE076BCT034
Saman Chauguthi	KCE076BCT035

Under the Supervision of
Er. Jen Bati
Department of Computer Engineering

Khwopa College of Engineering
Libali, Bhaktapur
2022-23

CERTIFICATE OF APPROVAL

This is to certify that this minor project work entitled "Devanagari Character Recognition System Using CNN" submitted by Rabit Khaitu (KCE076BCT028), Romik Gosai (KCE076BCT032), Sagar Suwal (KCE076BCT034) and Saman Chauguthi (KCE076BCT035) has been examined and accepted as the partial fulfillment of the requirements for degree of Bachelor in Computer Engineering.

.....
Er. Sudeep Shakya
Associate Professor
Dept. of Computer Engineering
Kathmandu Engineering College

.....
Er. Jen Bati
Project Supervisor
Assistant Lecturer
Khwopa College of Engineering

.....
Er. Dinesh Gothe
Head of Department
Department of Computer Engineering,
Khwopa College of Engineering

Copyright

Acknowledgement

We are deeply indebted to our supervisor **Er. Jen Bati** for boosting our efforts and morale by his valuable advice and suggestions regarding the project and supporting us in tackling various difficulties. We would like to express our deep gratitude to our Head of Department (HoD) **Er. Dinesh Man Gothe** for his advice, encouragement and support for completion of this project. We are also grateful to the Department of Computer Engineering at Khwopa College of Engineering for providing us with the opportunity to work on this project and for giving us access to all the necessary resources to complete our project. We would also like to thank all our classmates who helped us directly or indirectly for completion of project. Finally we would like to acknowledge everyone who played a role directly or indirectly for completing this project.

Rabit Khaitu	KCE076BCT028
Romik Gosai	KCE076BCT032
Sagar Suwal	KCE076BCT034
Saman Chauguthi	KCE076BCT035

Abstract

Overall around the world, more than 600 million people use Devanagari Language which include Nepali, Sanskrit, Hindi etc. There are several research works based on recognition of printed as well as handwritten Devanagari text in past few years. This document focuses on Devanagari optical character recognition (OCR). Various works have been discussed as the possibility of the project methodology. Different approaches taken in order to build handwritten Devanagari character detection system is discussed. For the recognition process, Data is preprocessed in order to make the neural network model more effective with the help of segmentation, normalization etc. Then model is trained with training set of data using predefined Convolutional Neural Network(CNN) architectures LeNET, AlexNET, ZFNET. We also proposed a CNN model which perform better on the used dataset. After the model is trained, its effectiveness is tested using Validation data and testing data. Moreover, this document contains a comprehensive bibliography of many selected papers appeared in reputed journals and conference proceedings in the field of Devanagari OCR.

Keywords: *Convolution Neural Network (CNN), Deep learning, Optical Character Recognition (OCR), Devanagari Script*

Contents

Certificate i
Copyright ii
Acknowledgement iii
Abstract iv
List of Figures viii
List of Tables ix
List of Symbols and Abbreviation x

1 Introduction **1**
1.1 Background 1
1.1.1 Optical Character Recognition 1
1.1.2 Devanagari Script 1
1.2 Problem Statement 2
1.3 Objective 2
1.4 Scope and Application 3

2 Literature Review **4**
2.1 Historical Background 4
2.2 Review of related literatures 6

3 Requirement Analysis **8**
3.1 Hardware Requirements 8
3.2 Software Requirements 8
3.3 Functional Requirements 8
3.3.1 Image Detection 9
3.3.2 Decode Image 9
3.3.3 Character Recognition 9
3.4 Non-functional Requirements 9
3.4.1 Reliability 9
3.4.2 Maintainability 9
3.4.3 Performance 9
3.4.4 Accuracy 9

4 Feasibility Study **10**
4.1 Economic Feasibility 10
4.2 Technical Feasibility 10
4.3 Operational Feasibility 10

5 Methodology **11**
5.1 Software Development Approach . . . 11
5.2 Data Collection and Preparation . . . 12
5.3 Training/Validation/Testing set Split . . . 12
5.4 Model Implementation . . . 12
5.4.1 Convolutional Neural Network (CNN) . . . 12
5.4.2 Implementing LeNET-5 Architecture . . . 16
5.4.3 Implementing AlexNET architecture . . . 17
5.4.4 Implementing ZFNET architecture . . . 18
5.5 CNN Modelling . . . 19
5.5.1 Train the model in training dataset . . . 19
5.5.2 Evaluate the model on validation dataset . . . 19
5.5.3 Pick the model that does best on validation dataset . . . 19
5.5.4 Confirming the result on testing dataset . . . 19
5.5.5 Prepare a Proposed Model . . . 20
5.6 Image Processing for Character to be predicted . . . 21
5.6.1 RGB to gray scale inversion . . . 21
5.6.2 Noise Removal . . . 22
5.6.3 Segmentation . . . 23
5.6.4 Inversion . . . 23
5.6.5 Universe of Discourse . . . 23
5.6.6 Resizing . . . 24
5.6.7 Normalization . . . 24
5.7 Model Evaluation . . . 24
5.7.1 Precision . . . 24
5.7.2 Recall . . . 25
5.7.3 F1-score . . . 25
5.8 Model Deployment . . . 25

6 System Design and Architecture **26**
6.1 Use Case Diagram . . . 26
6.2 State Chart Diagram . . . 27
6.3 Sequence Diagram . . . 27
6.4 System Block Diagram . . . 28

7 Results and Discussion **30**
7.1 Model Evaluation Result . . . 30
7.1.1 Train Validation Loss Graphs . . . 30
7.1.2 Train and Validation Accuracy Graph . . . 32
7.1.3 Confusion Matrix . . . 34
7.1.4 Evaluation Result . . . 36
7.2 Discussion . . . 41

8 Conclusion **42**

9 Limitation and Future Enhancement **43**
9.1 Limitations . . . 43
9.2 Future Enhancements . . . 43
Bibliography . . . 46

Appendix **47**
A Assistance for Cloning the Project 47
A.1 Devanagari Character Recognition System 47
B Snapshot 48
B.1 Gantt Chart for Project 48
B.2 Outcome of User Interface 48

List of Figures

1.1 Characters used in Devanagari Script 2

5.1 Prototype Model for Software Model 11
5.2 Lenet Model Architecture 16
5.3 AlexNET Model Architecture 17
5.4 ZFNET Model Architecture 18
5.5 CNN Modelling 19
5.6 Proposed Model Architecture 20
5.7 Image Processing Steps 21
5.8 Captured Image 22
5.9 Grayscaled Image 22
5.10 Working of non linear median filter 22
5.11 Filtered Image 22
5.12 Binarized Image 23
5.13 Inverted Image 23
5.14 Universe of Discourse 24
5.15 64x64 Resized Image 24

6.1 Use Case Diagram 26
6.2 State Chart Diagram 27
6.3 Sequence Diagram 27
6.4 System Block Diagram 28

7.1 Train and Validation Loss Graph for LeNET architecture 30
7.2 Train and Validation Loss Graph for AlexNET architecture 30
7.3 Train and Validation Loss Graph for ZFNET architecture 31
7.4 Train and Validation Loss Graph for Proposed architecture 31
7.5 Train and Validation Accuracy Graph for LeNET architecture . . . 32
7.6 Train and Validation Accuracy Graph for AlexNET architecture . . 32
7.7 Train and Validation Accuracy Graph for ZFNET architecture . . . 33
7.8 Train and Validation Accuracy Graph for Proposed architecture . . 33
7.9 Confusion Matrix of LeNET architecture 34
7.10 Confusion Matrix of AlexNET architecture 34
7.11 Confusion Matrix of ZFNET architecture 35
7.12 Confusion Matrix of Proposed architecture 35

List of Tables

2.1 Review Matrix with Research Papers, Authors and Accuracy. . . . 6

5.1 Data for Character Detection 12
5.2 LeNET Model Architecture 16
5.3 AlexNet Model Architecture 17
5.4 ZFNET Model Architecture 18
5.5 Proposed Model Architecture 20

7.1 Evaluation Result of LeNET Architecture 37
7.2 Evaluation Result of AlexNET Architecture 38
7.3 Evaluation Result of ZFNET Architecture 39
7.4 Evaluation Result of Proposed Architecture 40

List of Symbols and Abbreviation

CH	Chain code Histogram
CNN	Convolutional Neural Network
FN	False Negative
FP	False Positive
HCR	Hand printed Character Recognition
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
KNN	K-Nearest Neighbour
MED	Minimum Edit Distance
MQDF	Modified Quadratic Discriminant Function
OCR	Optical Character Recognition
RE	Regular Expression
ReLU	Rectified Linear Unit
SDLC	Software Development Life Cycle
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

Chapter 1

Introduction

1.1 Background

1.1.1 Optical Character Recognition

Various organization come across documents which are handwritten such as forms, checks, etc. Such documents need to be converted and stored in digital format. Simplest form of storing them in electronic form, is to scan and store them as images. However, storing images requires lot of storage and becomes impractical as the size of the document increases. Hence, hand printed character recognition (HCR)/ optical character recognition (OCR) which recognizes the printed/handwritten text to recognize the characters is a natural choice. [1]
Character recognition deals with recognizing the characters accurately from a document. Character recognition is done either from printed documents or handwritten documents. Character recognition is distinguished into two classes: on-line character recognition and off-line character recognition. [2] In an off-line handwritten text document only a sample of the text is available after the text has been written with variations in the handwriting style of writers, which makes the off-line handwriting recognition system more challenging than the on-line handwriting recognition system. Since handwriting recognition system has potential applications in the field of offline handwritten historical document digitization, bank cheque processing, postal automation, automatic data entry, etc., there is a necessity to improve the handwriting recognition system more accurately. [3]

1.1.2 Devanagari Script

Devanagari is the script used for writing many official languages in Nepal, such as Nepali, Maithili, Bhojpuri, Sanskrit etc. Many languages originated from Sanskrit language use similar scripts to Devanagari. Most of the Indian scripts including Devanagari originated from ancient Brahmi script through various transformations. Nepali Language Devanagari is composed of 13 vowels, and 36 consonants, 10 numerals and 15 modifiers, there are compound (composite) characters in most of Indian scripts including Devanagari, which are formed by combining two or more basic characters. The shape of a compound (composite) character is usually

more complex than its constituent characters. A vowel following a consonant may take a modified shape, which depending on the vowel is placed to the left, right, top, or bottom of the consonant, and are called modifiers or "matras" [4] . There are many different characters in this script which have similar structure differentiated only with presence of dots, lines. The recognition process becomes more complicated due the writing fashion of people.

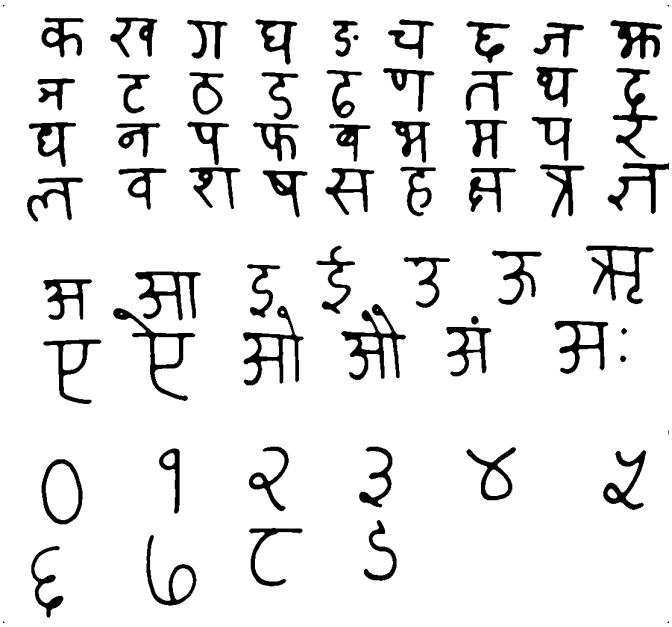


Figure 1.1: Characters used in Devanagari Script

1.2 Problem Statement

In today's digital age, connectivity is pervasive, and as such, handwritten notes and historical literature should also be preserved digitally for global dissemination. The storing of such works solely limits access to those who can understand the language used. However, with the existence of a language-specific script recognition system, like Devanagari, these works could be easily translated into multiple languages, increasing their accessibility to a wider audience. Our proposed system recognizes Devanagari characters and represents the first step toward our ultimate goal. The system is deployed as localhost web application and is available for testing in github [5].

1.3 Objective

The main aim of this project is:

- To detect handwritten Devanagari characters.

1.4 Scope and Application

- Can be used in banks to read amount filled by user and also verify the signature.
- Can be used to read out forms used to collect different information.

Chapter 2

Literature Review

2.1 Historical Background

From literature survey, we found that many researchers have proposed methods for off-line handwritten Devanagari character recognition but many of them used only subset of Devanagari characters to evaluate their methods.
[1] Sharma *et al* [6] proposed a method for recognition of handwritten Devanagari characters using the directional chain code information of the contour points of the characters as features. The characters are segmented into blocks using bounding box and a chain code histogram (CH) is computed in each of the blocks.
Sharma *et al* proposed a quadratic classifier-based scheme for the recognition of handwritten characters using 64-D directional chain codes and obtained 80.36% accuracy with the 11,270 dataset size.
The proposed method in [7] used regular expressions (RE) for handwritten Devanagari character recognition. In proposed work hand-written character encoded based on chain-code features. Then, RE of stored templates is matched with encoded string. Rejected samples are then sent to a minimum edit distance (MED) classifier for recognition. They obtained 82% accuracy on 5000 dataset.
In [8] two stage classification approach for handwritten Devanagari characters. In the first stage using structural properties like shirorekha, spine in character and second stage exploits some intersection features of characters which are fed to a feedforward neural network. To detect shirorekha and vertical bar they designed a differential distance based technique. They achieved 89.12% success rate on 50000 dataset.
Sahare *et al* [9] presented robust algorithms for character segmentation and recognition for multilingual Indian document images of Latin and Devanagari scripts. In this, character segmentation algorithm, primary segmentation paths are obtained using structural property of characters, whereas overlapped and joined characters are separated using graph distance theory. Finally, segmentation results are validated using highly accurate support vector machine classifier. For the proposed character recognition algorithm, three new geometrical shape based features are computed. First and second features are formed with respect to the center pixel of character, whereas neighborhood information of text pixels is used for the calculation of third feature. For recognizing the input character, k-Nearest Neighbor(KNN) classifier is used, as it has intrinsically zero training time.
Hanmandlu *et al* [10] proposed a method for recognition of handwritten Devana-

gari using modified quadratic classifier using features based on directional information obtained from the arc tangent of the gradient character and obtained 94.24% accuracy on 36172 dataset.
S. Arora *et al* [11] presented a elastic matching with PCA based system towards the recognition of off-line isolated uppercase English characters and Devanagari handwritten vowels characters. To improve the recognition performance estimation and the utilization of Eigen deformations were used. 94.91% accuracy obtained on 3600 Devanagari vowels.
Pal [12] proposed combined approach toward the recognition of off-line Devanagari handwritten characters. Two classifiers (MQDF and SVM) based on gradient and curvature features are combined for the recognition. They experimented on 36172 data set and obtained 95.13% accuracy.
Y. Gurav *et al* [13] proposed Devanagari Handwritten Character Recognition using Convolutional Neural Networks with Grey Scale conversion, Edge detection, Noise Removal for image preprocessing on Self-made 34604 handwritten images used for Devanagari script with no header line (Shirorekha) over them and the aaccuracy was 99.65%.
M. Bisht *et al* [3] presented Offline handwritten Devanagari modified character recognition using convolutional neural network using double CNN on hindi consonants and matras dataset with the accuracy of 90.99%.
S. Shitole *et all* [14] proposed Recognition of handwritten Devanagari characters using linear discriminant analysis using SVM classifier on 47 handwritten Devanagari characters of ISI dataset total of 4700 images with 100 images per character and the accuracy was 70.58%.
N. Aneja *et al* [15] presented Transfer learning using CNN for handwritten devanagari character recognition using pretrained model in CNN on 92000 image dataset achieving accuracy of 99%.
R. Sarkhel *et al* [16] proposed multi-scale deep quad tree based feature extraction method for the recognition of isolated handwritten characters of popular indic scripts using softmax classifier and multi-scale convolutional neural networks on HPL offline character database achieving accuracy of 95.18%.
S. Narang *et al* [17] presented Devanagari ancient documents recognition using statistical feature extraction techniques on 6152 pre-segmented samples of Devanagari ancient documents achieving accuracy of 89.95%.
S. Puri *et al* [18] proposed Devanagari character classification in printed and handwritten documents using SVM with Noise Removal, Skew Detection, Normalization, Gray scaling, Binarization applied in image processing on total of 60 documents where 60% used in training and 40% used during testing.

2.2 Review of related literatures

Table 2.1: Review Matrix with Research Papers, Authors and Accuracy.

S.N.	Title	Author/s	Year	Accuracy
1	Recognition of off-line handwritten devnagari characters using quadratic classifier	N. Sharma, U. Pal, F. Kimura, and S. Pal [6]	2006	80.36%
2	Fine classification recognition of hand written Devnagari characters with regular expressions minimum edit distance method	P. Deshpande, L. Malik, and S. Arora [7]	2008	82%
3	Two stage classification approach for handwritten Devanagari characters	S. Arora, D. Bhatcharjee, M. Nasipuri, and L. Malik [8]	2007	89.12%
4	Multilingual character segmentation and recognition schemes for indian document images	P. Sahare and S. B. Dhok [9]	2018	98.86%
5	Fuzzy model based recognition of handwritten hindi characters	M. Hanmandlu, O. R. Murthy, and V. K. Madasu [10]	2007	94.24%
6	Recognition of non-compound handwritten devnagari characters using a combination of mlp and minimum edit distance	S. Arora, D. Bhattacharjee, M. Nasipuri, D. K. Basu, and M. Kundu [11]	2010	94.91%
7	Accuracy improvement of Devanagari character recognition combining SVM and MQDF	U. Pal, S. Chanda, T. Wakabayashi, and F. Kimura [12]	2008	95.13%
8	Devanagari Handwritten Character Recognition using Convolutional Neural Networks	Y. Gurav, P. Bhagat, R. Jadhav and S. Sinha [13]	2020	99.65%
9	Offline handwritten Devanagari modified character recognition using convolutional neural network	M. Bisht and R. Gupta [3]	2021	90.99%
10	Recognition of handwritten Devanagari characters using linear discriminant analysis	S. Shitole and S. Jadhav [14]	2018	70.58%
11	Transfer learning using CNN for handwritten devanagari character recognition	N. Aneja and S. Aneja [15]	2019	99%

12	A multi-scale deep quad tree based feature extraction method for the recognition of isolated handwritten characters of popular indic scripts	R. Sarkhel, N. Das, A. Das, M. Kundu and M. Nasipuri [16]	2017	95.18%
13	Devanagari ancient documents recognition using statistical feature extraction techniques	S. Narang, M. K. Jindal and M. Kumar [17]	2019	88.95%
14	An efficient Devanagari character classification in printed and handwritten documents using SVM	S. Puri and S. P. Singh [18]	2019	99.54% for printed images and 98.35% for handwritten images

Chapter 3

Requirement Analysis

3.1 Hardware Requirements

The project is fully based on software so there are no hardware requirements except laptops for coding as well as preparing documents.

3.2 Software Requirements

Our script recognition system requires Python along with the use of different libraries which are described below;

a. Microsoft Teams

b. Flask

c. Github and Gitlab

d. Google Colaboratory and Kaggle

e. OpenCV

f. Overleaf

g. Python

h. Google Drive

i. Star UML

j. Visual Studio Code

k. Tensorflow and Keras

3.3 Functional Requirements

The main goal of this project is to convert handwritten Devanagari scripts into digital form. So, followings can be listed as the functional requirements of our Devanagari Script Recognition System.

3.3.1 Image Detection

This system takes image as input and recognizes objects or image contain form the image.

3.3.2 Decode Image

This system decodes image and convert it into matrix form for the future mathematical calculation used for character detection.

3.3.3 Character Recognition

This system can detect Devanagari characters with good accuracy.

3.4 Non-functional Requirements

These requirements are needed by the system for the better performance of recognition system. The point below focuses on non-functional requirements of the system.

3.4.1 Reliability

After completion, the project will be extremely dependable. The system will be able to run without fail in a specific environment.

3.4.2 Maintainability

The maintenance of the finished system will be easily feasible. Also we can easily maintain our system for future enhancements. By adding additional examples to the database with which we train our system, the system's ability to identify can be easily upgraded.

3.4.3 Performance

Performance of the system depends on how fast the system gives the result. After its completion, the system will be able to detect fake news in a very less time.

3.4.4 Accuracy

The technology should be able to tell whether the news being read is phony or trustworthy. The system will be capable of accurately detecting fake news.

Chapter 4

Feasibility Study

The following points put feasibility of the project.

4.1 Economic Feasibility

The total expenditure of the project is computational power only. The dataset and computational power required are available easily. Dataset is found from the internet and computational power using our own laptop alongside Google Collaboratory and Kaggle cloud computing service. So, the project is economically feasible.

4.2 Technical Feasibility

All of the data sets that are required for our project are easily available in the internet. We can complete the job using currently available tools. Open sourced and freely available tools will be adequate to meet the system's technical feasibility requirements.

4.3 Operational Feasibility

The systems of Devanagari character recognition is deployed on the localhost with very simple UI. User can run the application and recognize characters by uploading the picture of handwritten character on the white canvas then recognition model runs in the backend and returns required character prediction.

Chapter 5

Methodology

5.1 Software Development Approach

Prototype model is a software development model where instead of freezing the requirements before design or coding can proceed, a throwaway prototype is built to understand the requirements.The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models).This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product. The goal is to provide a system with overall functionality.
Our project is recognition system where design of the system is not rigidly fixed yet. So, with the help of prototype model approach, we can test new ideas in short period of time in order to create system with good accuracy.

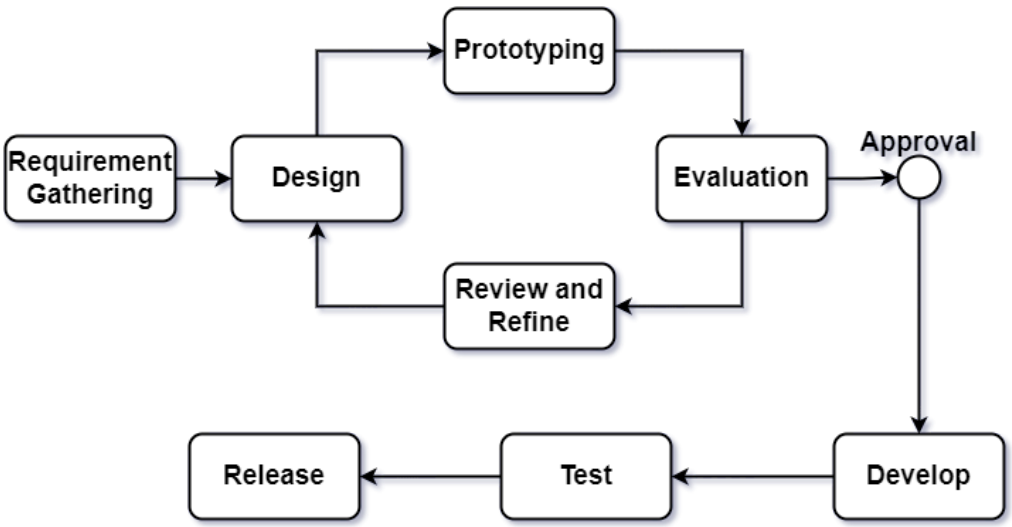


Figure 5.1: Prototype Model for Software Model

5.2 Data Collection and Preparation

We collected dataset for model training from the kaggle website for character recognition model using CNN. We collected datasets from 2 sources. From the first source [19], we collected 2000 images of size 32x32 for each class i.e. from devanagari characters ka to gya and numerals 0 to 9. In second source [20], we had 205 images of size 28x28 per class for Devanagari characters ka to gya and 288 for numerals 0-9. We cut out these to 200 images per class. The dataset from second source had white background and black text whereas dataset from first source had black background and white text. So we inverted color of images from second source i.e. testing data into black background and white text using PIL library function ImageOps.invert().

5.3 Training/Validation/Testing set Split

The dataset from the first source was divided into training and validation with 1700 and 300 images in each class respectively. Datasets from second source was used for testing. We have total 101200 images which were divided into training, validation and testing for each characters as shown in table 5.1.

Table 5.1: Data for Character Detection

Character Detection Dataset	**Type**	**Training**
Validation	**Testing**	
Character Image	73600	18400	9200
Character Image per class	1700	300	200

5.4 Model Implementation

5.4.1 Convolutional Neural Network (CNN)

Convolutional Neural networks(CNN) are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images.CNN is basically used for image classifications and identifying an image.It scans images from left to right and top to bottom to pull out important features from the image and combines the feature to classify images. There are different layers used in convolutional neural network which are:

- **Input Layer** : The input layer is the first layer of the CNN that takes in the input image or data. It typically has a fixed size and shape based on the input data.
- **Convolutional Layer** : The convolutional layer applies a set of learnable filters to the input image, which produces a set of feature maps. Each filter is small and slides over the input image, computing dot products and creating

a feature map. Suppose that we have some $N * N$ square neuron layer which is followed by our convolutional layer. If we use an $m * m$ filter ω, our convolutional layer output will be of size:

$$output - size = (N - m + 1)(N - m + 1) \tag{5.1}$$

In order to compute the pre-nonlinearity input to some unit x_{ij}^l in our layer, we need to sum up the contributions (weighted by the filter components) from the previous layer cells:

$$x_{ij}^l = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{l-1} \tag{5.2}$$

Then, the convolutional layer applies its nonlinearity:

$$y_{ij}^l = \sigma(x_{ij}^l) \tag{5.3}$$

- **Pooling Layer** : The pooling layer reduces the spatial dimensions (width and height) of the feature maps by down-sampling, while retaining the most important information. Max pooling and average pooling are commonly used pooling techniques.
- **Rectified Linear Unit (ReLU) layer** : The Rectified Linear Unit (ReLU) layer is an activation function commonly used in convolutional neural networks (CNNs) to introduce non-linearity. The ReLU activation function is defined as $f(x) = max(0, x)$, where x is the input to the ReLU layer.
- **Flattening Layer** : The Flattening layer in a convolutional neural network (CNN) is a layer that converts the output of the previous convolutional and pooling layers into a one-dimensional vector that can be fed into a fully connected layer for classification or further processing.
- **Fully Connected Layer** : The fully connected layer connects all the neurons in the previous layer to the current layer, performing a matrix multiplication and a bias addition to produce an output. This layer is used for classification tasks.

5.4.1.1 LeNET

LeNET is a convolutional neural network architecture designed by Yann LeCun et al. in the early 1990s, which was one of the earliest successful applications of deep learning to image recognition tasks. It consists of seven layers, including two convolutional layers, two subsampling layers, and three fully connected layers, and was originally designed for character recognition but has also been used successfully for image classification. LeNet was groundbreaking in its use of convolutional layers and subsampling layers to learn hierarchical representations of images, and its success has led to the development of many more advanced architectures for image classification, making it an important milestone in the history of deep learning.

5.4.1.2 AlexNET

AlexNet is a convolutional neural network architecture designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, which revolutionized the field of computer vision by achieving state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) that year. It consists of eight layers, including five convolutional layers and three fully connected layers. AlexNet uses a combination of convolutional layers, max pooling layers, and fully connected layers to learn hierarchical representations of images and classify them into different categories. The architecture introduced several new techniques, such as rectified linear units (ReLU) for activation functions, dropout regularization to prevent overfitting, and data augmentation to increase the size of the training set. AlexNet demonstrated the power of deep learning for image classification and paved the way for the development of many other deep learning architectures for computer vision. Overall, AlexNet represents a major milestone in the history of deep learning and continues to be a widely studied and influential architecture in the field.

5.4.1.3 ZFNET

ZFNet is a convolutional neural network architecture designed by Matthew Zeiler and Rob Fergus in 2013, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in that year. It consists of eight layers, including five convolutional layers and three fully connected layers. The architecture is similar to AlexNet, but with smaller filter sizes and a smaller stride in the first layer. ZFNet uses a combination of convolutional layers, max pooling layers, and fully connected layers to learn hierarchical representations of images and classify them into different categories. The smaller filter sizes allow for more precise feature detection, and the smaller stride in the first layer allows for more detailed information to be preserved. Overall, ZFNet was a significant breakthrough in the field of image classification, demonstrating the power of deep learning and paving the way for future advances in the field.

5.4.1.4 Activation Functions

An activation function is a mathematical function that is applied to the output of a neuron in a neural network. The activation function is used to introduce non-linearity into the output of the neuron, allowing the network to learn more complex and sophisticated patterns in the data.

1. Softmax Activation Function
The softmax activation function is a commonly used activation function in the output layer of a neural network for multi-class classification problems. The softmax function takes a vector of real numbers as input and maps it to a probability distribution over the output classes.The class with the highest probability is then selected as the predicted class for the input. The mathematical expression for the softmax function is:

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{5.4}$$

where, z_j is the input to the j-th neuron in the output layer, K is the total number of neurons in the output layer, and $\sigma(z_j)$ is the output of the j-th neuron.

2. ReLU Activation Function

ReLU (Rectified Linear Unit) is a commonly used activation function in neural networks due to its simplicity and effectiveness in training deep neural networks. The ReLU function returns the input value if it is positive, and returns 0 if it is negative. The ReLU function introduces non-linearity into the output of the neuron, allowing the network to learn more complex and sophisticated patterns in the data.The mathematical expression for the ReLU function is:

$$ReLU(x) = max(0, x) \tag{5.5}$$

3. tanh Activation Function

The tanh (hyperbolic tangent) activation function is a commonly used nonlinear function in neural networks. It is a mathematical function that maps its input to a value between -1 and 1. It is often used as an activation function in the hidden layers of neural networks because it can introduce nonlinearity to the network and can help model complex relationships between inputs and outputs. The formula for the tanh function is:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.6}$$

5.4.2 Implementing LeNET-5 Architecture

We converted 28x28 images seperated for testing into 32x32 using opencv library function resize.

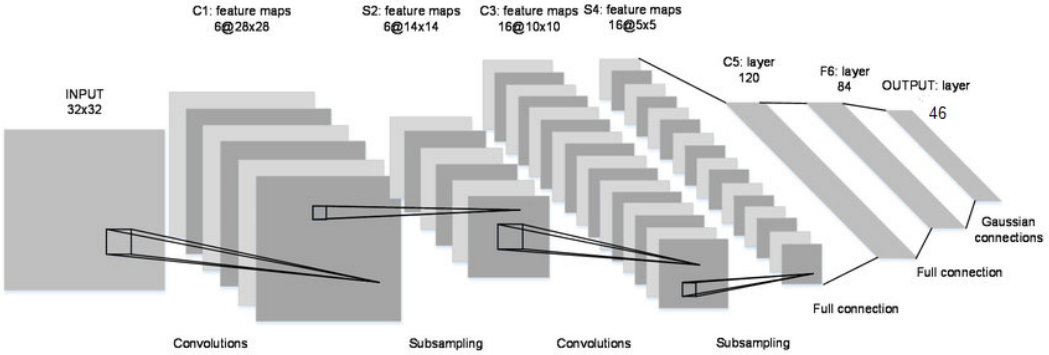


Figure 5.2: Lenet Model Architecture

Source: [21]

S.N.	Layer	Input Shape	Output Shape	param
1	Conv2D	32x32x1	28x28x6	156
2	AveragePooling2D	28x28x6	14x14x6	0
3	Conv2D	14x14x6	10x10x16	2416
4	AveragePooling2D	10x10x16	5x5x16	0
5	Conv2D	5x5x16	1x1x120	48120
6	Flatten	1x1x120	120	0
7	Dense	120	84	10164
8	Dense	84	46	3910

Table 5.2: LeNET Model Architecture

5.4.3 Implementing AlexNET architecture

We converted all 28x28 and 32x32 sized images into 64x64 image as an input in this model. Summary of AlexNET architecture implemented is given in 5.3

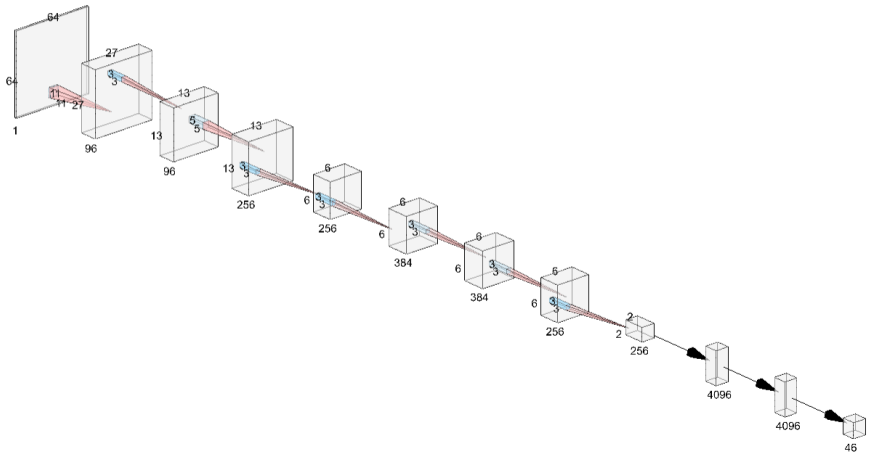


Figure 5.3: AlexNET Model Architecture

S.N.	Layer	Input Shape	Output Shape	param
1	Conv2D	64x64x1	27x27x96	11712
2	MaxPooling2D	27x27x96	13x13x96	0
3	BatchNormalization	13x13x96	13x13x96	52
4	Conv2D	13x13x96	13x13x256	614656
5	MaxPooling2D	13x13x256	6x6x256	0
6	BatchNormalization	6x6x256	6x6x256	24
7	Conv2D	6x6x256	6x6x384	885120
8	BatchNormalization	6x6x384	6x6x384	24
9	Conv2D	6x6x384	6x6x384	1327488
10	BatchNormalization	6x6x384	6x6x384	24
11	Conv2D	6x6x384	6x6x256	884992
12	MaxPooling2D	6x6x256	2x2x256	0
13	BatchNormalization	2x2x256	2x2x256	8
14	Flatten	2x2x256	1024	0
15	Dense	1024	4096	4198400
16	Dropout	4096	4096	0
17	Dense	4096	4096	1678131
18	Dropout	4096	4096	0
19	Dense	4096	46	188462

Table 5.3: AlexNet Model Architecture

5.4.4 Implementing ZFNET architecture

We converted all 28x28 and 32x32 sized images into 64x64 image as an input in this model. Summary of ZFNET architecture implemented is given in table 5.4

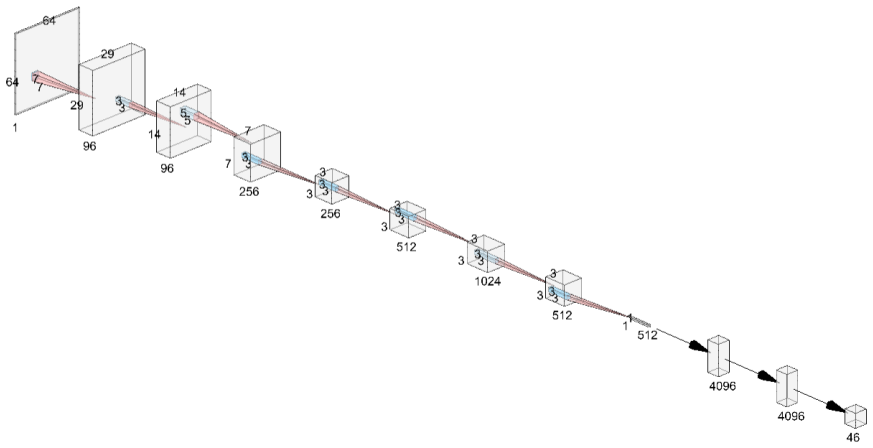


Figure 5.4: ZFNET Model Architecture

S.N.	Layer	Input Shape	Output Shape	param
1	Conv2D	64x64x1	29x29x96	4800
2	MaxPooling2D	29x29x96	14x14x96	0
3	BatchNormalization	14x14x96	14x14x96	56
4	Conv2D	14x14x96	7x7x256	61456
5	MaxPooling2D	7x7x256	3x3x256	0
6	BatchNormalization	3x3x256	3x3x256	12
7	Conv2D	3x3x256	3x3x512	1180160
8	BatchNormalization	3x3x512	3x3x512	12
9	Conv2D	3x3x512	3x3x1024	4719616
10	BatchNormalization	3x3x1024	3x3x1024	12
11	Conv2D	3x3x1024	3x3x512	4719104
12	MaxPooling2D	3x3x512	1x1x512	0
13	BatchNormalization	1x1x512	1x1x512	4
14	Flatten	1x1x512	512	0
15	Dense	512	4096	2101248
16	Dropout	4096	4096	0
17	Dense	4096	4096	16781312
18	Dropout	4096	4096	0
19	Dense	4096	46	188462

Table 5.4: ZFNET Model Architecture

5.5 CNN Modelling

5.5.1 Train the model in training dataset

In this section, we trained 1700 training dataset in each class into all the predefined CNN model implemented. Training was done with maximum of 50 epochs and early stopping was used with $patience = 10$.

Figure 5.5: CNN Modelling

5.5.2 Evaluate the model on validation dataset

While training the model, 300 dataset in each class seperated for validation were evaluated in each epochs. If there was improvement in validation accuracy than the previous best validation accuracy, then that model was saved for future use using model checkpointing.

5.5.3 Pick the model that does best on validation dataset

After the training was done, rather than using the latest model after the last epoch before early stopping, model with highest validation accuracy was picked as the best model.

5.5.4 Confirming the result on testing dataset

After picking the best model, that model was used in order to evaluate the result in the testing dataset with 200 images in each class. This result was used for further evaluation in order to calculate various evaluation metrics such as precision, accuracy, F1 score etc.

5.5.5 Prepare a Proposed Model

After model evaluation on all three predefined model, architecture with best evaluation was tweaked on few parameters in the neural network layers in order to obtain more accurate result in the model evaluation. For this we tweaked few parameters in AlexNET architecture as it produced best result on testing data previously. The architecture of proposed model is:

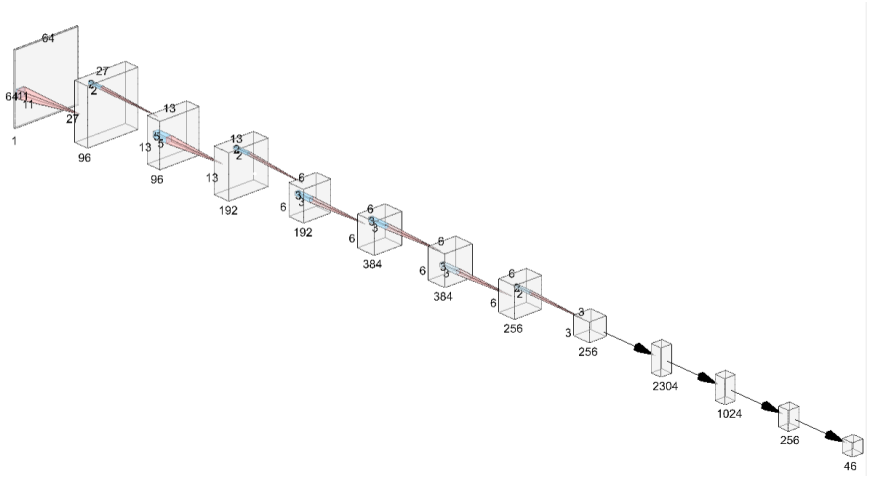


Figure 5.6: Proposed Model Architecture

S.N.	Layer	Input Shape	Output Shape	param
1	Conv2D	64x64x1	27x27x96	11712
2	MaxPooling2D	27x27x96	13x13x96	0
3	BatchNormalization	13x13x96	13x13x96	52
4	Conv2D	13x13x96	13x13x192	460992
5	MaxPooling2D	13x13x192	6x6x192	0
6	BatchNormalization	6x6x192	6x6x192	24
7	Conv2D	6x6x192	6x6x384	663936
8	BatchNormalization	6x6x384	6x6x384	24
9	Conv2D	6x6x384	6x6x384	1327488
10	BatchNormalization	6x6x384	6x6x384	24
11	Conv2D	6x6x384	6x6x256	884992
12	MaxPooling2D	6x6x256	3x3x256	0
13	BatchNormalization	3x3x256	3x3x256	12
14	Flatten	3x3x256	2304	0
15	Dense	2304	1024	2360320
16	Dropout	1024	1024	0
17	Dense	1024	256	262400
18	Dropout	256	256	0
19 s	Dense	256	46	11822

Table 5.5: Proposed Model Architecture

5.6 Image Processing for Character to be predicted

After model training, to use real image into the trained model for recognition, we used image processing. The following steps were used. For Processing following steps from 5.7 is used:

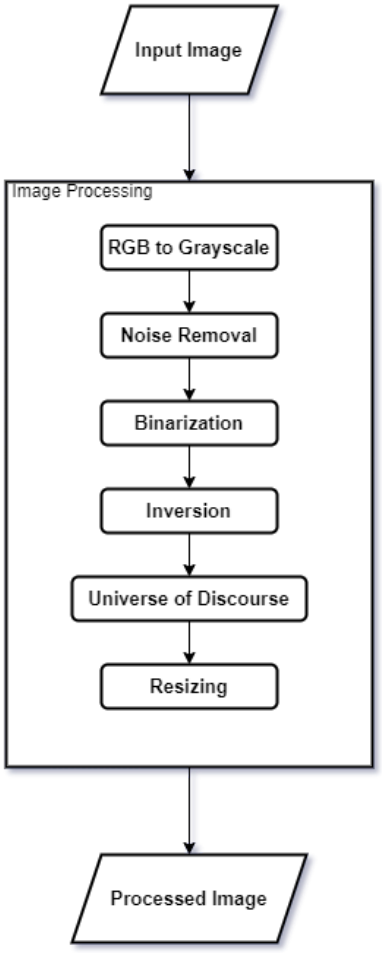


Figure 5.7: Image Processing Steps

The main task of image processing for the character recognition includes:

5.6.1 RGB to gray scale inversion

If the character is written in different color other than black we need to convert this image into a gray scale image.This is done by taking weighted threshold of Red, Green and Blur component of a RGB image. For this we use simple yet popular color to grayscale image conversion approach called as the line projection [22].

$$I = \alpha_r R + \alpha_g G + \alpha_b B \tag{5.7}$$

where α_r, α_g and α_b are non-negative coefficients

$$\alpha_r + \alpha_g + \alpha_b = 1 \tag{5.8}$$

and for this project we use simple average values for $\alpha, i.e.(\alpha_r, \alpha_g, \alpha_b) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$

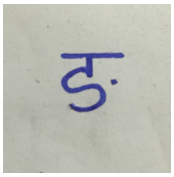


Figure 5.8: Captured Image

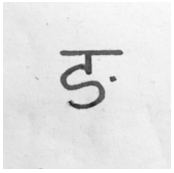


Figure 5.9: Grayscaled Image

5.6.2 Noise Removal

Noise is removed from images with the help of filtering. In our work we use non-linear median filtering. This method is particularly effective at removing salt and pepper pixels, i.e bright regions containing dark pixels and vice versa [23].

$$f'(x, y) = median\{f(x + i, y + j)|(i, j) \in R\} \tag{5.9}$$

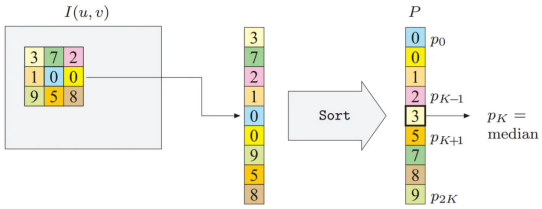


Figure 5.10: Working of non linear median filter

Source: [24]

Figure 5.11: Filtered Image

5.6.3 Segmentation

Segmentation is differentiating object from the background. For the grayscale image f(x,y), the segmented image s(x, y) is obtained by binarization process as given below. The Grayscale value varies from 0 to 255 in which we took 127 as a threshold value. So, all those value below 127 was converted to 0 and above 100 was converted to 255. From this process text is differentiated from the background.

$$S(x,y) = \begin{cases} 1 & \text{if } f(x;y) >= T \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

Figure 5.12: Binarized Image

5.6.4 Inversion

When we write documents we usually write on a white paper with a black/blue pen. But for character recognition we use black pixels as background and white as foreground. Hence, there is a need to invert the images.

$$i(x,y) = 255 - f(x,y) \tag{5.11}$$

Figure 5.13: Inverted Image

5.6.5 Universe of Discourse

It is the smallest area where the character is present, basically the smallest rectangle surrounding the character. It removes the extra unwanted pixels around the image. We cropped the text or character upto its boundary from the image. Extra unwanted pixels around the image are removed. We applied this by scannning the image and assigning the value with first white pixel from top, bottom, left and right as minimum row, maximum row, minimum column and maximum column respectively. After this, image was sliced according to the previously assigned boundary rows and columns.

Figure 5.14: Universe of Discourse

5.6.6 Resizing

Resizing refers to conversion of variable sized input images to a predefined fixed size used for processing. In our work we resize the input images to a fixed size of 64x64 pixels.

Figure 5.15: 64x64 Resized Image

5.6.7 Normalization

In Normalization we convert pixel values which ranges from 0-255 into a standard interval i.e. 0-1 by dividing each pixels by 255.

$$n(x, y) = \frac{f(x, y)}{255} \tag{5.12}$$

5.7 Model Evaluation

Evaluation for the output is done by analysing the F1 score of each of the predefined CNN model implemented. Since this is classification problem, F1 score is sufficient to evaluate and find out the best model.

5.7.1 Precision

Precision measures the accuracy of positive predictions, i.e., the fraction of correctly predicted positive examples out of all positive predictions made by the model. Here, precision is the number prediction character that correctly match actual character, divided by the total number of prediction in the particular class. It is calculated as:

$$Precision = \frac{TP}{TP + FP} \tag{5.13}$$

where, TP = True Positive
FP = False Positive

5.7.2 Recall

Recall is a metric that quantifies the number of correct positive predictions made out of all positive predictions that could have been made. Here, recall is the number prediction character that correctly match actual character, divided by the total number of actual characters in the particular class. It is calculated as :

$$recall = \frac{TP}{TP + FN} \tag{5.14}$$

where, FN = False Negative

5.7.3 F1-score

F1-score is the harmonic mean of precision and recall, and it provides a balanced measure of the overall quality of the character prediction.

$$F1Score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{5.15}$$

5.8 Model Deployment

We used flask for building web app for this project. We have currently deployed our model in localhost. Character Recognition for images is done through the web app by interfacing frontend to the recognition model.

Chapter 6

System Design and Architecture

We developed a Devanagari Character Recognition System which takes image as its input and process it to provide prediction character for the input image.

6.1 Use Case Diagram

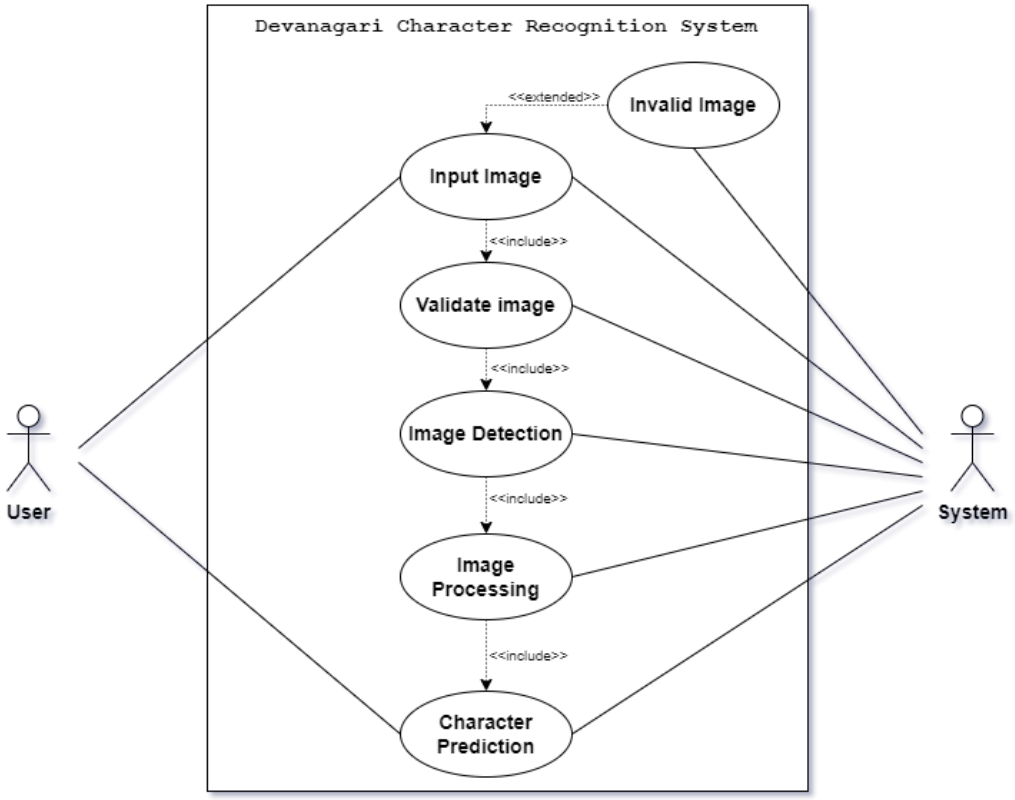


Figure 6.1: Use Case Diagram

6.2 State Chart Diagram

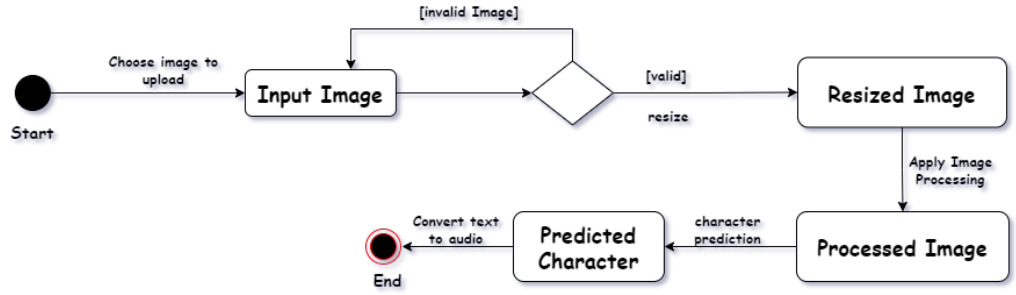


Figure 6.2: State Chart Diagram

6.3 Sequence Diagram

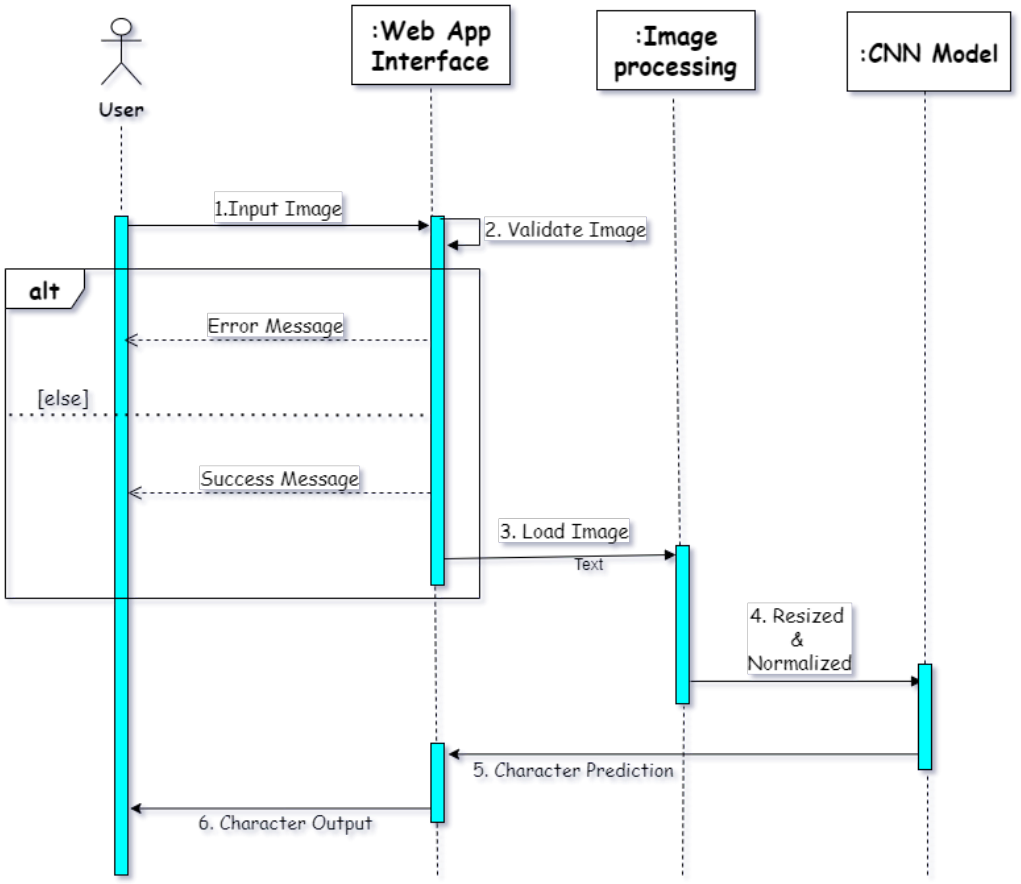


Figure 6.3: Sequence Diagram

6.4 System Block Diagram

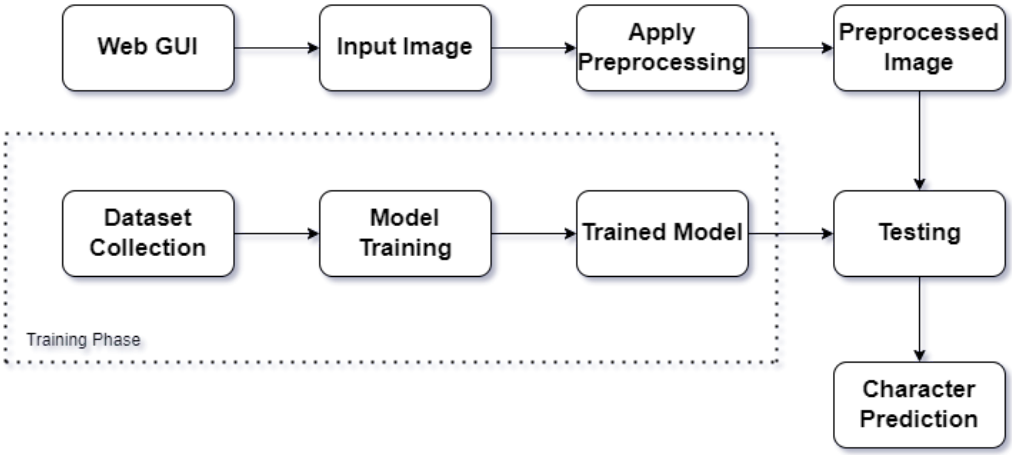


Figure 6.4: System Block Diagram

5

Chapter 7

Results and Discussion

7.1 Model Evaluation Result

7.1.1 Train Validation Loss Graphs

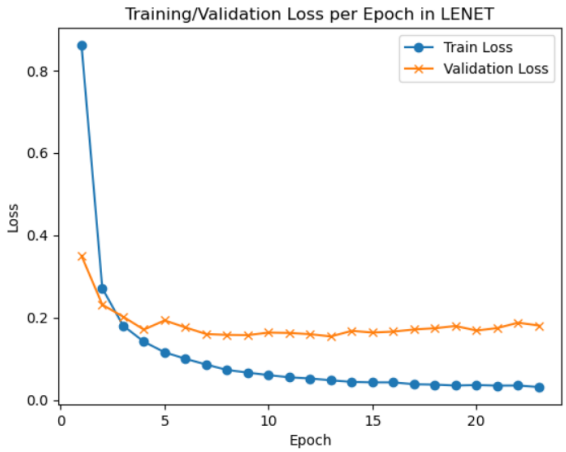


Figure 7.1: Train and Validation Loss Graph for LeNET architecture

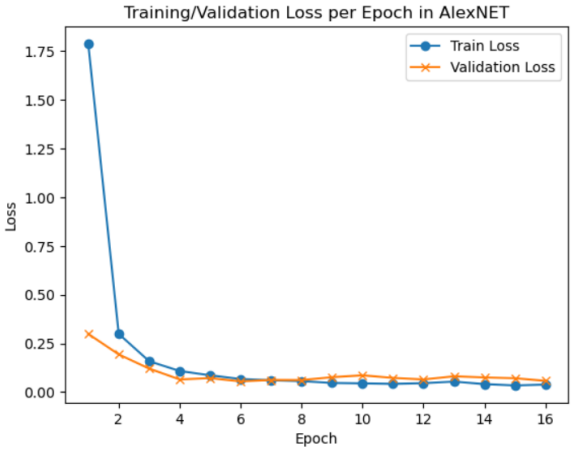


Figure 7.2: Train and Validation Loss Graph for AlexNET architecture

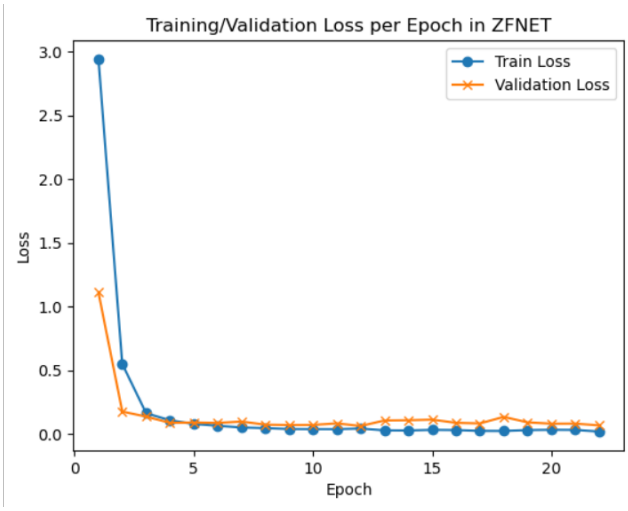


Figure 7.3: Train and Validation Loss Graph for ZFNET architecture

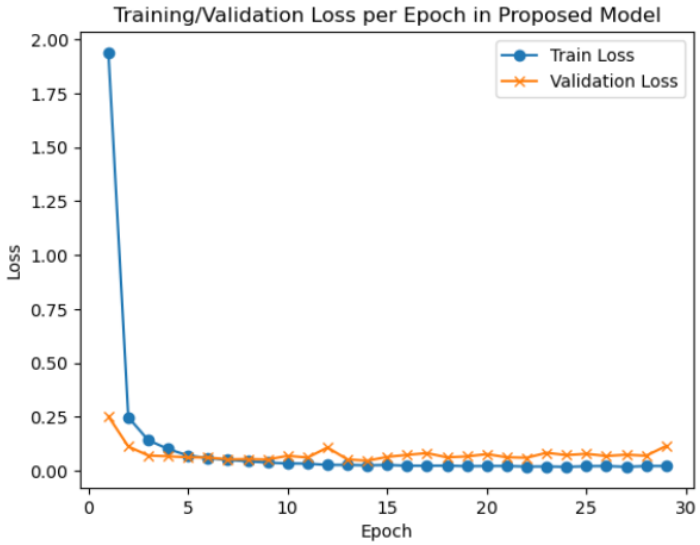


Figure 7.4: Train and Validation Loss Graph for Proposed architecture

From the Train and Validation Loss Graphs of all three architectures, it is found that the loss curve of validation dataset is most similar to that of training datasets in case of AlexNET. In ZFNET, there is a bit more variation between loss curve of training and validation dataset. In the case of LeNET, it has the most variation between training and validation dataset among the three architectures.

7.1.2 Train and Validation Accuracy Graph

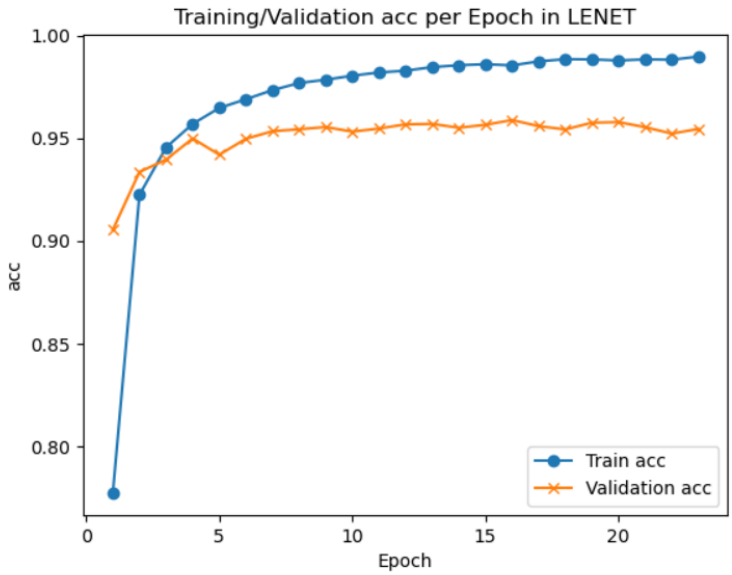


Figure 7.5: Train and Validation Accuracy Graph for LeNET architecture

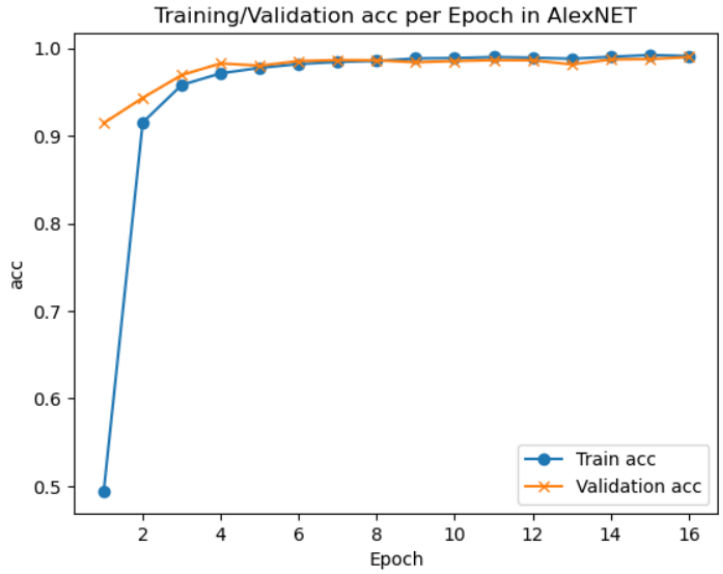


Figure 7.6: Train and Validation Accuracy Graph for AlexNET architecture

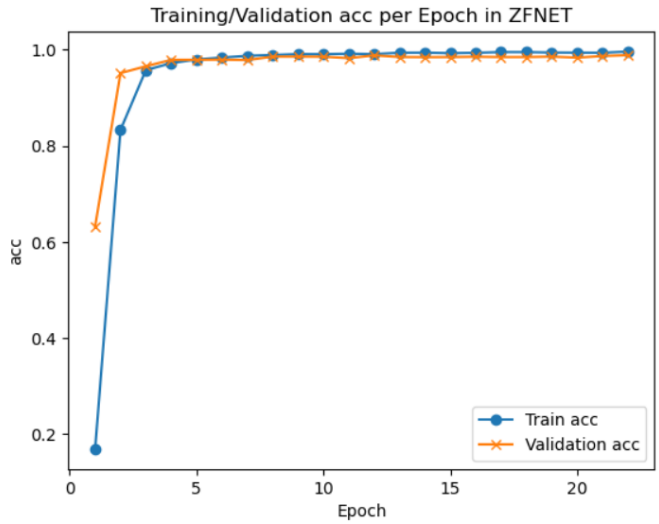


Figure 7.7: Train and Validation Accuracy Graph for ZFNET architecture

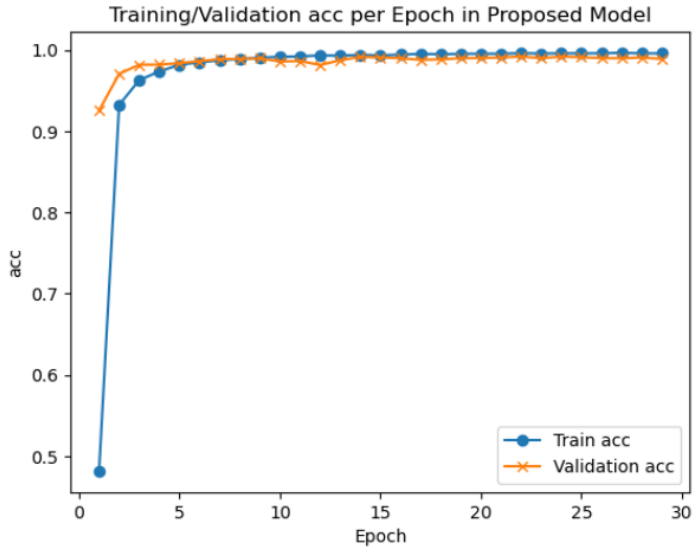


Figure 7.8: Train and Validation Accuracy Graph for Proposed architecture

From the Train and Validation Accuracy Graphs of all three architectures, it is found that the accuracy curve of validation dataset is most similar to that of training datasets in case of AlexNET. In ZFNET, there is a bit more variation between accuracy curve of training and validation dataset. In the case of LeNET, it has the most variation between accuracy of training and validation dataset among the three architectures.

7.1.3 Confusion Matrix

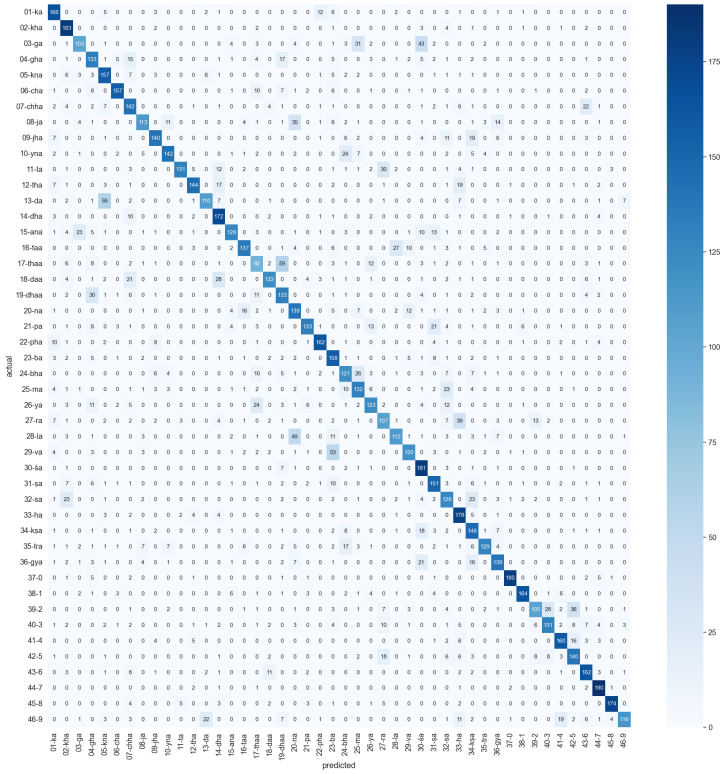


Figure 7.9: Confusion Matrix of LeNET architecture

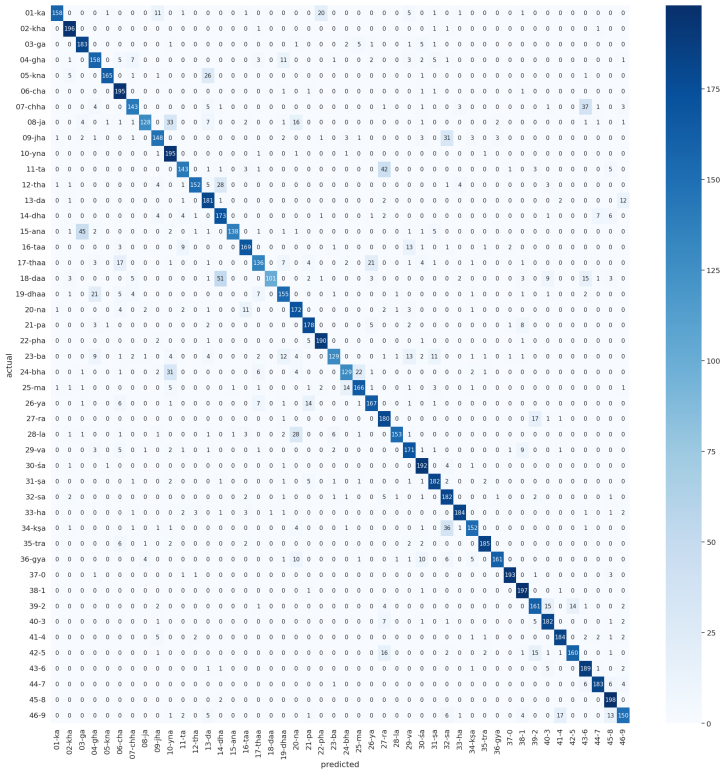


Figure 7.10: Confusion Matrix of AlexNET architecture

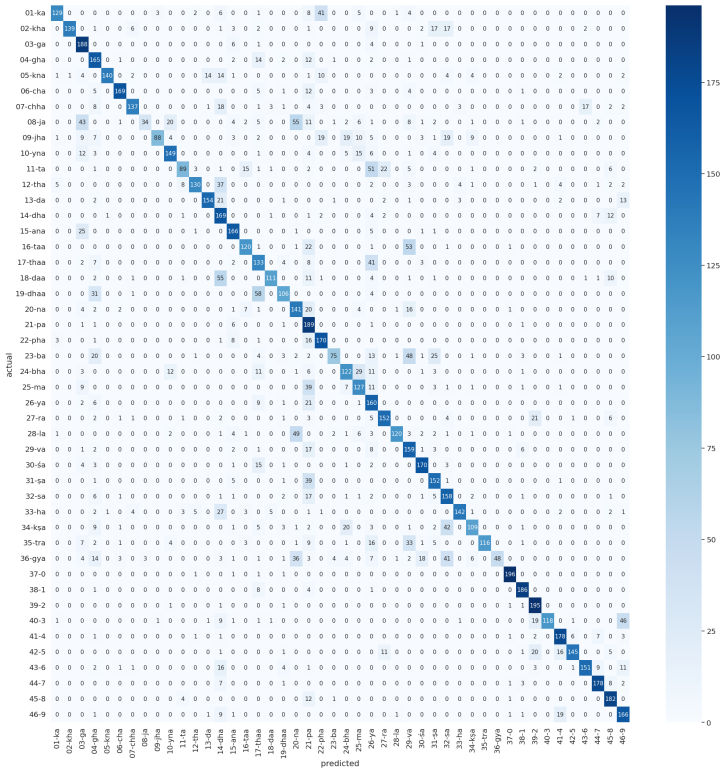


Figure 7.11: Confusion Matrix of ZFNET architecture

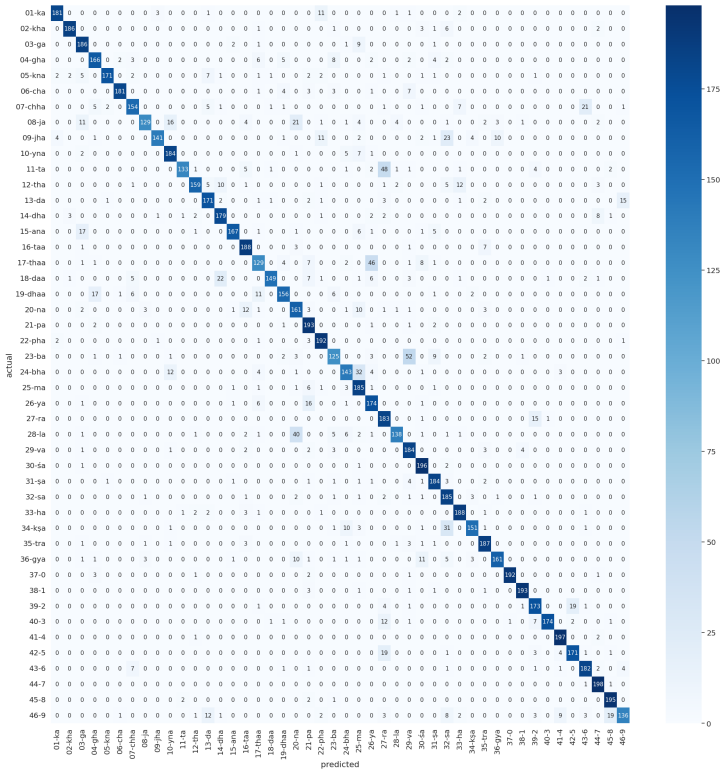


Figure 7.12: Confusion Matrix of Proposed architecture

7.1.4 Evaluation Result

7.1.4.1 Evaluation Result of testing data on LeNET Architecture

SN	Character	Precision	Recall	F1 Score
1	ka	0.74	0.83	0.78
2	kha	0.69	0.92	0.78
3	ga	0.73	0.52	0.60
4	gha	0.56	0.67	0.61
5	kna	0.61	0.79	0.69
6	cha	0.89	0.79	0.83
7	chha	0.59	0.71	0.64
8	ja	0.79	0.56	0.66
9	jha	0.93	0.66	0.77
10	yna	0.85	0.72	0.78
11	ta	0.77	0.55	0.64
12	tha	0.67	0.86	0.75
13	da	0.81	0.64	0.72
14	dha	0.80	0.69	0.74
15	ana	0.52	0.46	0.49
16	taa	0.79	0.61	0.69
17	thaa	0.52	0.67	0.58
18	daa	0.54	0.69	0.61
19	dhaa	0.84	0.67	0.74
20	na	0.86	0.81	0.84
21	pa	0.52	0.79	0.63
22	pha	0.58	0.60	0.59
23	ba	0.59	0.66	0.62
24	bha	0.70	0.61	0.66
25	ma	0.59	0.54	0.56
26	ya	0.73	0.56	0.63
27	ra	0.75	0.60	0.67
28	la	0.58	0.91	0.71
29	va	0.67	0.76	0.71
30	sa	0.53	0.63	0.57
31	sha	0.61	0.89	0.72
32	shha	0.56	0.74	0.64
33	ha	0.84	0.65	0.73
34	ksha	0.71	0.69	0.70
35	tra	0.96	0.90	0.93
36	gya	0.92	0.82	0.87
37	0	0.78	0.53	0.63
38	1	0.77	0.66	0.71
39	2	0.80	0.80	0.80
40	3	0.68	0.70	0.69
41	4	0.70	0.81	0.75
42	5	0.84	0.95	0.89

43	6	0.95	0.87	0.91
44	7	0.90	0.58	0.71
45	8	0.95	0.87	0.91
46	9	0.90	0.58	0.71
	accuracy			0.70
	macro avg	0.73	0.70	0.71
	weighted avg	0.73	0.70	0.71

Table 7.1: Evaluation Result of LeNET Architecture

7.1.4.2 Evaluation Result of testing data on AlexNET Architecture

SN	Character	Precision	Recall	F1 Score
1	ka	0.98	0.79	0.87
2	kha	0.91	0.98	0.94
3	ga	0.77	0.92	0.84
4	gha	0.77	0.79	0.78
5	kna	0.98	0.82	0.89
6	cha	0.78	0.97	0.87
7	chha	0.86	0.71	0.78
8	ja	0.93	0.64	0.76
9	jha	0.80	0.74	0.77
10	yna	0.69	0.97	0.81
11	ta	0.86	0.71	0.78
12	tha	0.95	0.76	0.84
13	da	0.74	0.91	0.82
14	dha	0.67	0.86	0.75
15	ana	0.99	0.69	0.81
16	taa	0.86	0.84	0.85
17	thaa	0.81	0.68	0.74
18	daa	0.99	0.51	0.67
19	dhaa	0.78	0.78	0.78
20	na	0.71	0.86	0.78
21	pa	0.84	0.89	0.86
22	pha	0.88	0.95	0.92
23	ba	0.90	0.65	0.75
24	bha	0.84	0.65	0.73
25	ma	0.84	0.83	0.83
26	ya	0.83	0.83	0.83
27	ra	0.69	0.90	0.78
28	la	0.97	0.77	0.85
29	wa	0.78	0.85	0.81
30	sa	0.82	0.96	0.89
31	sha	0.85	0.91	0.88

32	shha	0.66	0.91	0.77
33	ha	0.94	0.92	0.93
34	ksha	0.89	0.76	0.82
35	tra	0.95	0.93	0.94
36	gya	0.96	0.81	0.88
37	0	0.97	0.96	0.97
38	1	0.86	0.98	0.92
39	2	0.79	0.81	0.80
40	3	0.83	0.91	0.87
41	4	0.89	0.92	0.90
42	5	0.92	0.80	0.86
43	6	0.74	0.94	0.83
44	7	0.93	0.92	0.92
45	8	0.83	0.99	0.90
46	9	0.82	0.75	0.79
	accuracy			0.84
	macro avg	0.85	0.84	0.83
	weighted avg	0.85	0.84	0.83

Table 7.2: Evaluation Result of AlexNET Architecture

7.1.4.3 Evaluation Result of testing data on ZFNET Architecture

SN	Character	Precision	Recall	F1 Score
1	ka	0.91	0.65	0.76
2	kha	0.99	0.65	0.82
3	ga	0.59	0.94	0.73
4	gha	0.54	0.82	0.65
5	kna	0.99	0.70	0.82
6	cha	0.93	0.84	0.89
7	chha	0.90	0.69	0.78
8	ja	0.92	0.17	0.29
9	jha	0.96	0.44	0.60
10	yna	0.78	0.74	0.76
11	ta	0.83	0.45	0.58
12	tha	0.90	0.65	0.76
13	da	0.90	0.77	0.83
14	dha	0.42	0.84	0.56
15	ana	0.75	0.83	0.79
16	taa	0.79	0.60	0.68
17	thaa	0.47	0.67	0.55
18	daa	0.91	0.56	0.69
19	dhaa	0.77	0.53	0.63
20	na	0.49	0.70	0.58

21	pa	0.38	0.94	0.54
22	pha	0.69	0.85	0.76
23	ba	0.88	0.38	0.53
24	bha	0.67	0.61	0.64
25	ma	0.62	0.64	0.63
26	ya	0.42	0.80	0.55
27	ra	0.80	0.76	0.78
28	la	0.96	0.60	0.74
29	wa	0.46	0.80	0.59
30	sa	0.83	0.85	0.84
31	sha	0.67	0.76	0.71
32	shha	0.54	0.79	0.64
33	ha	0.93	0.71	0.80
34	ksha	0.80	0.55	0.64
35	tra	1.00	0.58	0.73
36	gya	1.00	0.24	0.39
37	0	0.96	0.98	0.97
38	1	0.89	0.93	0.91
39	2	0.74	0.97	0.84
40	3	1.00	0.59	0.74
41	4	0.78	0.89	0.83
42	5	0.94	0.72	0.82
43	6	0.88	0.76	0.81
44	7	0.87	0.89	0.88
45	8	0.77	0.91	0.83
46	9	0.67	0.83	0.74
	accuracy			0.71
	macro avg	0.78	0.71	0.71
	weighted avg	0.78	0.71	0.71

Table 7.3: Evaluation Result of ZFNET Architecture

7.1.4.4 Evaluation Result of testing data on proposed Architecture

SN	Character	Precision	Recall	F1 Score
1	ka	0.96	0.91	0.93
2	kha	0.97	0.93	0.95
3	ga	0.81	0.93	0.86
4	gha	0.84	0.83	0.84
5	kna	0.98	0.85	0.91
6	cha	0.97	0.91	0.94
7	chha	0.87	0.77	0.81
8	ja	0.94	0.65	0.77
9	jha	0.97	0.70	0.82

10	yna	0.85	0.92	0.88
11	ta	0.97	0.67	0.79
12	tha	0.94	0.80	0.86
13	da	0.84	0.85	0.85
14	dha	0.83	0.90	0.86
15	ana	0.96	0.83	0.89
16	taa	0.85	0.94	0.89
17	thaa	0.77	0.65	0.70
18	daa	0.97	0.74	0.84
19	dhaa	0.89	0.78	0.83
20	na	0.66	0.81	0.73
21	pa	0.76	0.96	0.85
22	pha	0.85	0.96	0.90
23	ba	0.80	0.62	0.70
24	bha	0.81	0.71	0.76
25	ma	0.70	0.93	0.79
26	ya	0.71	0.89	0.78
27	ra	0.65	0.92	0.76
28	la	0.93	0.69	0.79
29	wa	0.69	0.92	0.79
30	sa	0.85	0.98	0.91
31	sha	0.86	0.92	0.89
32	shha	0.68	0.93	0.78
33	ha	0.87	0.94	0.91
34	ksha	0.93	0.76	0.83
35	tra	0.89	0.94	0.91
36	gya	0.92	0.81	0.86
37	0	0.99	0.96	0.98
38	1	0.96	0.96	0.96
39	2	0.83	0.86	0.85
40	3	0.99	0.87	0.93
41	4	0.92	0.95	0.95
42	5	0.89	0.85	0.87
43	6	0.85	0.91	0.88
44	7	0.90	0.99	0.95
45	8	0.89	0.97	0.93
46	9	0.87	0.68	0.76
	accuracy			0.85
	macro avg	0.87	0.85	0.85
	weighted avg	0.87	0.85	0.85

Table 7.4: Evaluation Result of Proposed Architecture

7.2 Discussion

Based on the train and validation loss and accuracy graphs of all three architectures as well as proposed model, it was observed that the validation curve of AlexNET closely resembled that of its training dataset. For ZFNET, there was slightly more variation between the curves of its training and validation datasets compared to AlexNET. Among the three architectures, LeNET exhibited the greatest variation between the curves of its training and validation datasets. On testing data with 200 images per class average F1 score, precision and recall was found to be highest in proposed model which has the accuracy of 85%, average precision of 0.87, recall 0.85 and f1 score of 0.85 which is significantly more from the three predefined architecture used.

Chapter 8

Conclusion

To conclude, Devanagari character recognition system was completed successfully by making use of predefined CNN architectures like LeNET, AlexNET and ZFNET. We also proposed a model by tweaking few parameters on the best working predefined model for the better result on the used dataset. It outperformed all the predefined models while evaluating on the testing dataset. We used different image processing steps such as RGB to grayscale conversion, noise removal, binarization, inversion, universe of discourse, normalization etc in order to increase the accuracy of the model. We deployed model in a localhost web app which help user to detect and recognize different devanagari handwritten characters.

Chapter 9

Limitation and Future Enhancement

9.1 Limitations

The limitations of our system are:

1. Struggles to recognize the characters that seems similar.
2. Cannot recognize flipped characters.
3. Cannot recognize tilted characters.
4. Cannot recognize characters with modifiers.
5. Cannot recognize compound characters.
6. Cannot recognize words.

9.2 Future Enhancements

For future enhancements for our project we could implement these:

1. Recognition of multiple characters at once.
2. Recognition of characters with different modifiers.

Bibliography

[1] S. Prabhanjan and R. Dinesh, "Deep learning approach for devanagari script recognition," *International Journal of Image and Graphics*, vol. 17, no. 03, p. 1750016, 2017.

[2] B. Dessai and A. Patil, "A deep learning approach for optical character recognition of handwritten devanagari script," in *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, vol. 1, 2019, pp. 1160–1165.

[3] M. Bisht and R. Gupta, "Offline handwritten devanagari modified character recognition using convolutional neural network," *Sādhanā*, vol. 46, no. 1, pp. 1–4, 2021.

[4] R. Jayadevan, S. R. Kolhe, P. M. Patil, and U. Pal, "Offline recognition of devanagari script: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 6, pp. 782–796, 2011.

[5] R. Gosai, S. Suwal, R. Khaitu, and S. Chauguthi, "Romikgosai/devanagari-character-recognition-cnn." [Online]. Available: https://github.com/romikgosai/Devanagari-character-recognition-CNN

[6] N. Sharma, U. Pal, F. Kimura, and S. Pal, "Recognition of off-line handwritten devnagari characters using quadratic classifier," in *Computer Vision, Graphics and Image Processing*, P. K. Kalra and S. Peleg, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 805–816.

[7] P. Deshpande, L. Malik, and S. Arora, "Fine classification recognition of hand written devnagari characters with regular expressions minimum edit distance method," *Journal of Computers*, vol. 3, 05 2008.

[8] S. Arora, D. Bhatcharjee, M. Nasipuri, and L. Malik, "A two stage classification approach for handwritten devnagari characters," in *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*, vol. 2, 2007, pp. 399–403.

[9] P. Sahare and S. B. Dhok, "Multilingual character segmentation and recognition schemes for indian document images," *IEEE Access*, vol. 6, pp. 10 603–10 617, 2018.

[10] M. Hanmandlu, O. R. Murthy, and V. K. Madasu, "Fuzzy model based recognition of handwritten hindi characters," in *9th Biennial Conference of the*

Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications (DICTA 2007), 2007, pp. 454–461.

[11] S. Arora, D. Bhattacharjee, M. Nasipuri, D. K. Basu, and M. Kundu, "Recognition of non-compound handwritten devnagari characters using a combination of mlp and minimum edit distance," *arXiv preprint arXiv:1006.5908*, 2010.

[12] U. Pal, S. Chanda, T. Wakabayashi, and F. Kimura, "Accuracy improvement of devnagari character recognition combining svm and mqdf," in *Proc. 11th Int. Conf. Frontiers Handwrit. Recognit.* Citeseer, 2008, pp. 367–372.

[13] Y. Gurav, P. Bhagat, R. Jadhav, and S. Sinha, "Devanagari handwritten character recognition using convolutional neural networks," in *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 2020, pp. 1–6.

[14] S. Shitole and S. Jadhav, "Recognition of handwritten devanagari characters using linear discriminant analysis," in *2018 2nd International Conference on Inventive Systems and Control (ICISC)*. IEEE, 2018, pp. 100–103.

[15] N. Aneja and S. Aneja, "Transfer learning using cnn for handwritten devanagari character recognition," in *2019 1st International Conference on Advances in Information Technology (ICAIT)*. IEEE, 2019, pp. 293–296.

[16] R. Sarkhel, N. Das, A. Das, M. Kundu, and M. Nasipuri, "A multi-scale deep quad tree based feature extraction method for the recognition of isolated handwritten characters of popular indic scripts," *Pattern Recognition*, vol. 71, pp. 78–93, 2017.

[17] S. Narang, M. Jindal, and M. Kumar, "Devanagari ancient documents recognition using statistical feature extraction techniques," *Sādhanā*, vol. 44, pp. 1–8, 2019.

[18] S. Puri and S. P. Singh, "An efficient devanagari character classification in printed and handwritten documents using svm," *Procedia Computer Science*, vol. 152, pp. 111–121, 2019.

[19] S. Acharya, A. K. Pant, and P. K. Gyawali, "Deep learning based large scale handwritten devanagari character recognition," in *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, Dec 2015, pp. 1–6.

[20] A. K. Pant, S. P. Panday, and S. R. Joshi, "Off-line nepali handwritten character recognition using multilayer perceptron and radial basis function neural networks," in *2012 Third Asian Himalayas International Conference on Internet*. IEEE, 2012, pp. 1–5.

[21] "Lenet model architecture." [Online]. Available: https://www.researchgate.net/profile/Sheraz-Khan-14/publication/321586653/figure/fig4/AS:568546847014912@1512563539828/The-LeNet-5-Architecture-a-convolutional-neural-network.png

[22] Y. Wan and Q. Xie, "A novel framework for optimal rgb to grayscale image conversion," in *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 2. IEEE, 2016, pp. 345–348.

[23] S. Suhas and C. Venugopal, "Mri image preprocessing and noise removal technique using linear and nonlinear filters," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. IEEE, 2017, pp. 1–4.

[24] "Working of non linear median filter." [Online]. Available: https://peeranut435home.files.wordpress.com/2018/10/5-10.png

Appendix

A Assistance for Cloning the Project

A.1 Devanagari Character Recognition System

A.1.1 Clone Git Project

`https://github.com/romikgosai/Devanagari-character-recognition-CNN`

A.1.2 Installation Required

To run the project, Python 3.8 or above is required along with pip3 with the latest version. To upgrade the pip3 version to the latest version, run the following command:

```
sudo -H pip3 install --upgrade pip
```

To install the required libraries, run the install.sh file as follows:

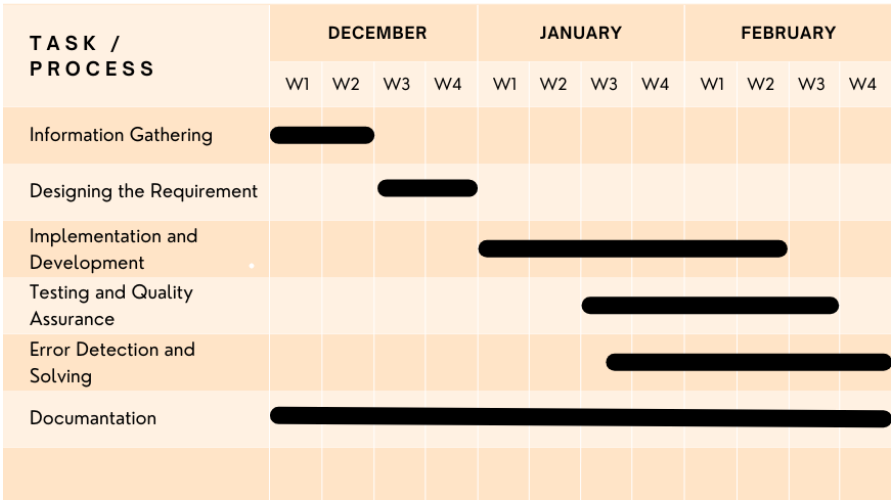
```
./install.sh
```

Then, Open a terminal or command prompt window on your local machine. Navigate to the directory where you want to clone the project. Type "git clone" followed by the URL provide above.

```
git clone https://github.com/romikgosai/Devanagari-character-recognition-CNN
```

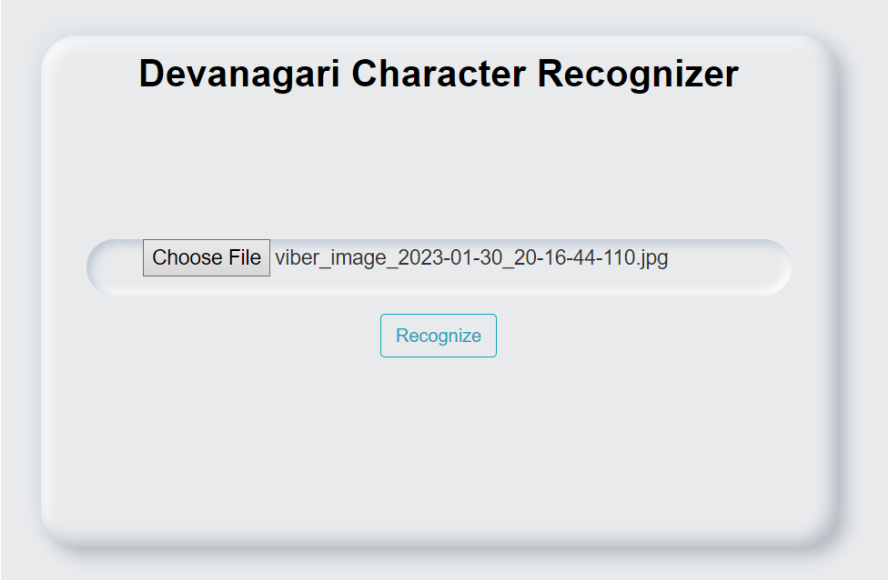
B Snapshot

B.1 Gantt Chart for Project

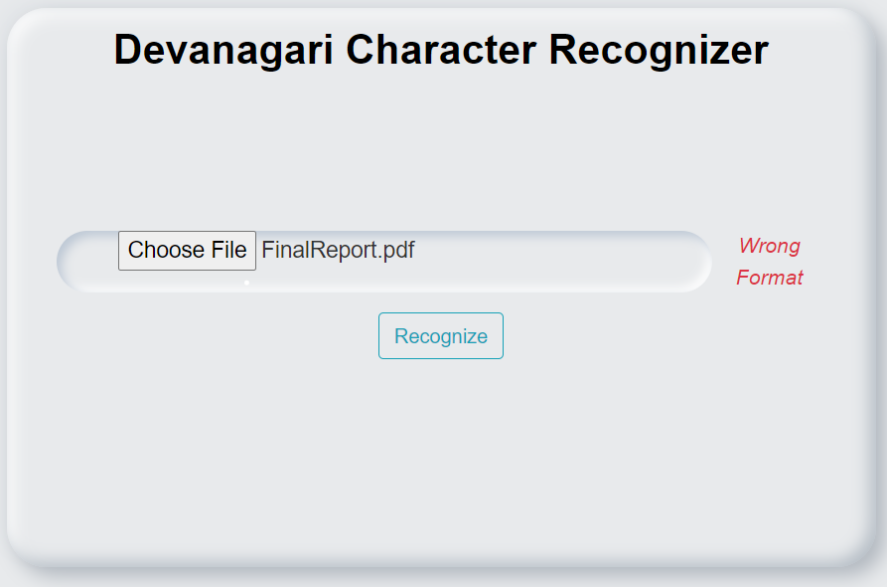


B.2 Outcome of User Interface

B.2.1 Form Page



B.2.2 Invalid Input



B.2.3 Prediction Page