TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
THAPATHALI CAMPUS

A Major Project Final Report
On
Prachalit Lipi Character Recognition using Deep Convolutional Neural Network

Submitted By:

Riwaj Neupane	(THA075BEI034)
Shiv Ranjan Gupta	(THA075BEI039)
Shovit Nepal	(THA075BEI040)
Sushovan Shakya	(THA075BEI046)

Submitted To:
Department of Electronics and Computer Engineering
Thapathali Campus
Kathmandu, Nepal

May, 2023

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
THAPATHALI CAMPUS

A Major Project Final Report
On
Prachalit Lipi Character Recognition using Deep Convolutional Neural Network

Submitted By:

Riwaj Neupane	(THA075BEI034)
Shiv Ranjan Gupta	(THA075BEI039)
Shovit Nepal	(THA075BEI040)
Sushovan Shakya	(THA075BEI046)

Submitted To:
Department of Electronics and Computer Engineering
Thapathali Campus
Kathmandu, Nepal

In partial fulfillment for the award of a Bachelor's degree in Electronics, Communication and Information Engineering

Under the Supervision of
Er. Hasina Shakya

May, 2023

DECLARATION

We hereby certify that the report of the project titled "Prachalit Lipi Character Recognition Using Deep Convolutional Neural Network" that is being submitted to the Department of Electronics and Computer Engineering, IOE, Thapathali Campus, is an accurate description of the work done by us. This certification is made in order to partially fulfill the requirements for the award of the degree of Bachelor of Engineering in Electronics, Communication, and Information Engineering. We are the only authors of this report, and the content has not been presented to a university or other institution in order to receive credit for it. No sources other than those listed here have been used.

Riwaj Neupane (THA075BEI034) _____

Shiv Ranjan Gupta (THA075BEI039) _____

Shovit Nepal (THA075BEI040) _____

Sushovan Shakya (THA075BEI046) _____

DATE: May, 2023

CERTIFICATE OF APPROVAL

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled **'Prachalit Lipi Character Recognition using Deep Convolutional Neural Network'** submitted by **Riwaj Neupane, Shiv Ranjan Gupta, Shovit Nepal and Sushovan Shakya** in partial fulfillment of the requirements for the Bachelor's Degree in Electronics, Communication and Information Engineering.

Supervisor: **Er. Hasina Shakya**
Department of Electronics and Computer Engineering
Institute of Engineering, Thapathali Campus

External Examiner: **Dr. Prakash Poudyal**
Department of Computer Science and Engineering
Kathmandu University, Dhulikhel

Project Co-ordinator: **Er. Umesh Kanta Ghimire**
Department of Electronics and Computer Engineering
Institute of Engineering, Thapathali Campus

Head of Department: **Er. Kiran Chandra Dahal**
Department of Electronics and Computer Engineering
Institute of Engineering, Thapathali Campus

DATE OF APPROVAL: May, 2023

COPYRIGHT

ACKNOWLEDGEMENT

This project is prepared in partial fulfilment of the requirement for for the the bachelor's degree in Electronics and Communication Engineering. We owe the Department of Electronics and Computer Engineering at IOE's Thapathali Campus an expression of appreciation for giving us the chance to complete a major project as part of our educational requirements. We would also like to owe our gratitude to our supervisor, Er. Hasina Shakya, for her guidance. The experience of working on this project will surely enrich our technical knowledge and also gives experience of working on a project and also develop our team works skills to a great extent.

Riwaj Neupane (THA075BEI034)

Shiv Ranjan Gupta (THA075BEI039)

Shovit Nepal (THA075BEI040)

Sushovan Shakya (THA075BEI046)

ABSTRACT

Many manuscripts and inscriptions were written historically, spanning several centuries in Kathmandu valley, by the indigenous Newar people, who developed their own alphabetic writing system. The process of identification and recognition is an difficult task. So, due to recent developments in AI, many languages and scripts have developed the Optical Character Recognition for recognizing the handwritten scripts and digitizing the script through the use of VGG Convolutional Neural Network(CNN) Thus, to develop OCR for Newari language and subsequent scripts used, we have prepared the project ”Prachalit Lipi Character Recognition Using Deep Convolutional Neural Network”.

Keywords: *Deep Learning, Image Processing, Inscription, Optical Character Recognition,VGG,Flask*

TABLE OF CONTENTS

List of Figures

List of Tables

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
CE	Common Era
CSS	Cascading Style Sheet
CUDA	Compute Unified Device Architecture
CV	Computer Vision
CNN	Convolutional Neural Network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
HTML	Hyper Text Markup Language
KNN	K-Nearest Neighbour
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
MNIST	Modified National Institute of Standards and Technology
MP	Megapixel
OCR	Optical Character Recongnition
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SDK	Software Development Kit
SVM	Support Vector Machine
UI	User Interface
VGG	Visual Geometry Group

1. INTRODUCTION

The research aims to identify and categorize the Prachalit Newari script's characters. When a character image is provided, the system identifies it and digitizes the script by romanizing the relevant characters.

1.1. Background

In Nepal, 123 languages are spoken as mother tongue, as reported in 2011 National Census [1]. The majority of the languages spoken in Nepal are members of the Sino-Tibetan and Indo-Aryan language groups. The Newar people are the native inhabitants of Nepal Mandala, which includes the Kathmandu Valley and other areas of Nepal, and they speak Nepal Bhasa, a Sino-Tibetan language.

According to Hemraj Shakya [2], the most ancient script found in Nepal is Brahmi script, also known as Ashokan script, as the script was widely used during the period of Ashoka's reign to make inscriptions. Brahmi script was supposedly in use in 3rd century CE, the earliest recorded instance in Nepal being the Ashoka Pillar in Lumbini, marking the birthplace of Gautama Buddha.

From 4th century CE till 7th century CE, Purva Licchavi was used, and from 7th century till 12th century, Uttara Lichcavi script was widely used. The earliest recorded instance of Purva Licchavi script was found in an inscription in Swoyambhu and in Changunaryan, supposedly written during the reign of King Mandev. An inscription dated Nepal Samvat 82 (961 AD) was found in Patan, written in Uttara Lichhavi script, during the reign of King Narendradev. Ranjana Lipi was developed in 11th century and was prevalently used till 18th century. Earliest recorded instance of Ranjana script was found in Patan, dated Nepal Samvat 2 (881 AD), in a script titled *Kāraṇḍavyūha Sūtra*. Ranjana script also gained popularity in Tibet (called *Lantsa* in Tibetan), China and Japan, as it was widely used to write scripts of Mahayana Buddhism. In Nepal, Ranjana was widely used for writing manuscripts as well as inscriptions, especially Buddhist scriptures.

Nepal Lipi, also known as Prachalit Lipi, came to prominence in 6th century CE, and was widely used to write Nepal Bhasa till 20th century. The earliest recorded instance of Prachalit script dates to 10th century CE, in a manuscript titled *Lankāvatāra Sūtra*, which is dated Nepal Samvat 28 (908 AD). Prachalit script was the most used writing system to write manuscripts and inscriptions, and it is found that majority of manuscripts stored in National Archive in the Department of Archaeology are written in Prachalit Script. Apart from aforementioned scripts, many other writing systems were

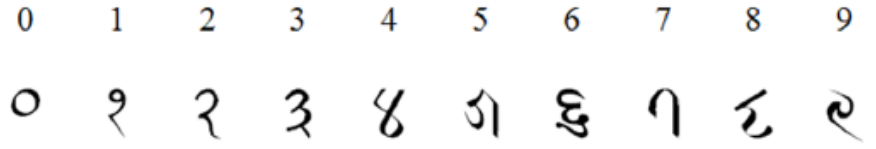
prevalent historically in Nepal. Scripts such as Bhujimol and Litumol were also used for writing Nepal Bhasa historically, particularly from 11th century till 17th century, but has now become extinct. Tibetan script also was seemingly in use in Nepal, starting from 12th century, as inscriptions written in Tibetan script also have been recorded in the National Archive. Manuscripts written in Maithili, Bengali, Gujarati and Tamil scripts also have been found in Nepal, dated 11th century.

While Devanagari script is widely used in Nepal in modern times, the use of Devanagari script in Nepal was seen first in 10th century CE. An inscription dated Nepal Samvat 512 (AD 1390-91), during the reign of Jayasthiti Malla, was found in Kumbheshwor of Patan, which is the first recorded instance of Devanagari in Nepal. The first recorded instance of modern Devanagari however is dated Nepal Samvat 774 (AD 1653), in an inscription in Pashupati, during the reign of Pratap Malla. With Nepali language being the lingua franca of Nepal, Devanagari script become more prevalent, especially from the 20th century, and was eventually adopted by several languages in Nepal, including Nepal Bhasa, for transcribing purposes. However, Ranjana and Prachalit script are prevalent and widely used to write the language in modern times. Spanning several centuries, manuscripts were written in Kathmandu Valley in tremendous amount, many of which are well-preserved within the valley, as well as in museums outside Nepal. However, extraction of information from the manuscript is often a tedious task, as one is expected to be acquainted with both the script and the language, and reading the text line-by-line becomes quite inefficient in this regard.

With the dawn of digital era, many of the documents got digitized, thus enabling safer archiving of information. Manual typing is often the go-to approach for digitization of information; however, development of Optical Character Recognition (OCR) has enabled the end-user to scan the documents and digitize the written or printed texts. OCR is an application of Machine Learning (ML) field, and is being actively developed for several languages of the world. For Newari language, however, digitization is just in infancy. The history of development of fonts was just a matter of recent history, and OCR is still a long way to go.

(a) Vowels

(b) Consonants



(c) Digits

Figure 1.1: Prachalit Script Varṇamālā

1.2. Motivation

Inscriptions and manuscripts tend to degrade over time, due to various environmental factors. As a result, the characters written in the script can get quite daunting to read, and becomes a time-consuming process to understand the character and extract information. The archaeologist or paleographer also has the task of either transcribing the script in written form, or transliterating the extracted information into another system of writing, and then converting it in a digital format for storage. The task involved thus becomes quite lengthy and inefficient, and human error is expected, which can affect the eventual outcome of the extracted information. Thus, this project is aimed at efficient and accurate extraction of information from historical manuscripts and inscriptions, and transliteration into modern writing systems, thus enabling easy digital storage and archiving.

1.3. Problem statement

As previously mentioned, historical manuscripts and inscriptions are susceptible to deterioration over time due to environmental factors, making it challenging for archaeologists or paleographers to decipher and gather information from them. Additionally, understanding the script used in these manuscripts is necessary to comprehend the information they contain. The major difficulty lies in storing the information from historical sources in a digital format. Even if photographs of the manuscripts or inscriptions are taken and saved as digital images, extracting the essential information can be time-consuming as one needs to read and transcribe them line-by-line into a different script before typing them in a textual format for storage. This process is inefficient and consumes a lot of time. To improve this process, a dataset of characters specific to the writing system can be created, and VGG CNN Learning algorithms can be applied to accurately recognize characters and extract information more efficiently.

1.4. Objectives

- To apply VGG16 CNN model to effectively classify Prachalit Script characters
- To convert the recognized characters into Romanized form, enabling their digitization

1.5. Scope of Project

The successful completion of this project will be of great assistance to various professionals such as archaeologists, paleographers, and linguists. Given the vast number of

historical manuscripts archived in museums and historical places, the project's output will help scholars extract valuable information from these manuscripts accurately and efficiently. The digitized information can then be stored and archived for future use. The project can also be useful to students and language enthusiasts interested in learning more about historical writing systems. Additionally, the project's outcomes can aid the archaeology department in conducting research, studying historical information, and archiving data.

1.6. Report Organization

The first chapter of this report gives a brief introduction and historical background of the domain, and explains the major objectives and scope of the project.
Chapter 2 includes the existing systems and different works done in the field.
Chapter 3 explains the system's software requirements and the project feasibility from different aspects.
Chapter 4 includes the theoretical basis of the system, the system architecture used, and the system design of the project.
Chapter 5 discusses the implementation of the system design and architecture for the project.
Chapter 6 includes the obtained result and analysis of the obtained result.
Chapter 7 includes the conclusions derived from the project, and further future enhancements.

2. LITERATURE REVIEW

Previous and current research in the fields of Optical Character Recognition (OCR), Neural Networks, and efforts on OCR of Indic languages were looked to for the project's literature study.

Large-scale data gathering and labeling become more crucial as machine learning becomes more commonplace, especially for cutting-edge neural networks. By concentrating on techniques for data labeling such semi-supervised learning and active learning, researchers in the fields of various machine learning aspects such as: Natural Language Processing, Computer Vision and Deep Learning.

In a recent review, Y. Roh, G. Heo, and S. E. Whang [3] investigated the research landscape of how these strategies complement one another and offered advice on which methodology to apply for certain sub-problems in data gathering, data labeling, and data refining. The study also identified several intriguing issues with data collecting that still need to be resolved.

We might predict the integration of Big data and AI in machine learning in the future, not just in terms of data collecting. This indicates that data-driven methodologies will become more critical to the development of AI.

Y.Roh et al.'s research underscores how vital it is to assess and analyze data effectively; however, determining if you've collected adequate quantities as well as relevant information still eludes researchers today. Selecting suitable datasets becomes essential while working on machine learning assignments since numerous variables must be considered for optimal model training success coupled with efficient efforts such as choosing enough related material while avoiding overwhelming your team with irrelevant or excessive information. It's also noteworthy that dynamic dataset arrays like sensor signal streaming requires changing metadata accordingly.

Without data, machine learning is impossible; without data, AI cannot learn. It has become a labor-intensive task to create a custom dataset. The Prachalit character images are not widely available so manual collection from different schools, language experts, teachers, books, scriptures et al. have been used.

In recent years, data has become increasingly important for algorithm training, as noted by A. Gonfalonieri in their research [4]. To create a dataset that is of high quality and sufficient size, we have employed data generation and augmentation techniques in our model. In addition, we have divided the data into three sets: the training, validation, and

testing set, which are used for various stages of training, tuning, model selection, and testing. The final machine learning model is chosen and fine-tuned using the validation set.

It is essential that the data used for machine learning models is of high quality and correctly integrated. For this reason, we apply data preprocessing techniques such as feature transformation once the data has been gathered. This involves formatting the diverse and essential dataset to files based on the label and parameters of the data. Missing values are added, and unwanted data is removed. We also extract the most important features for prediction and select them to ensure faster computations and lower memory consumption. By preprocessing the data in this way, we are able to ensure that the data used to train our machine learning model is accurate and appropriate for the task at hand.

LeNet is a pioneering convolutional neural network proposed by Y. LeCun, L. Bottou, et. al. [5] in 1998, proposed for recognition of handwritten characters. LeNet was a groundbreaking model, which demonstrated that the convolutional neural networks could be used effectively in recognizing handwritten characters, as it obtained significantly high accuracy in the MNIST dataset. LeNet was developed initially in recognition of handwritten characters for US Postal Service.

The seven layers of the LeNet design include three convolutional layers, two subsampling levels, and two fully interconnected layers. The first layer convolutions the input picture using a collection of gained filters, and the result is then sent via a subsampling layer. The input image's dimension is subsequently decreased by the subsampling layer. The successive placement of the convolutional and subsampling layers, with the process being repeated two or three times, allows for the extraction of ever more complicated characteristics from the input image.

The last subsampling layer's output is flattened and transmitted through two completely connected layers before being converted to the proper classes. The network's top layer, the softmax layer, creates a probability distribution for every potential candidate class. The class picked to represent the predicted class is the one with the highest probability.

LeNet architecture was a pioneering architecture which eventually paved the way for newer CNN architectures, such as AlexNet, ResNet and VGGNet. The aforementioned architectures were build using the principles of LeNet, incorporating additional layers to improve the eventual performance of character recognition.

A. Krizhevsky, I. Sutskever, and G. Hinton [6] The 2012 article "ImageNet Classifica-

tion with Deep Convolutional Neural Networks" used a deep convolutional neural network design and on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) dataset, cutting-edge performance was shown. The term suggested by the CNN model was AlexNet.

With a top-5 error rate of 15.3%, the AlexNet architecture surpassed the previous state-of-the-art models on the ILSVRC dataset. The researchers showed that while the deeper layers of the network learnt more complex features like object components and textures, the network's early levels only learned basic attributes like edges and corners.

Eight layers make up the AlexNet architecture: two fully linked layers, a softmax layer, and five convolutional layers. Following two further convolutional layers with 256 filters each, the initial convolutional layer has 96 filters in it. The fully connected layers each contain 4096 neurons, while the last two convolutional layers each have 384 filters. Each class in the ImageNet dataset is represented by one of the 1000 neurons in the output layer.

The research presented in the AlexNet article, which offered numerous significant breakthroughs in the field of machine learning, was pioneering. Rectified Linear Unit (ReLU) activation functions were suggested by the authors, and they considerably increased output accuracy since they solved the vanishing gradient issues and sped up training. The article also suggested adding procedures like cropping, shearing, flipping, and color shifting to the dataset in order to expand the training set and improve model performance. In order to avoid overfitting of the model, the authors also suggested the use of dropout layers, which involves eliminating a number of nodes from the neural network.

K. Simoyan and A. Zisserman [7] proposed the article on extremely deep convolutional networks for large-scale image recognition, also known as the VGGNet study. The study proposed a deep convolutional neural network architecture that outperformed earlier deep learning models substantially, achieving a top-5 error rate on the ILSVRC dataset of 7.3%

The VGGNet architecture is an extremely deep convolutional neural network design. The original VGGNet design has 19 layers, including 3 fully linked levels and 16 convolutional layers. A max pooling layer follows each convolutional layer in turn after the convolutional layers are arranged in groups of two or three. The first two convolutional layers of VGGNet each have 64 filters, then there are layers with 128, 256, and 512 filters. Each of the dense layers produced after the last max pooling layer's output was sent onto the flatten layer has 4096 neurons, with the final dense layer having 1000 neurons.

A ground-breaking paper in the field of computer vision was the VGGNet study. The research demonstrated that the performance of image recognition was greatly enhanced by increasing the network's depth. The idea of utilizing tiny filters (((3x3)) rather than huge filters was also introduced by VGGNet, which reduced the amount of parameters and made the network easier to train. The study also showed that convolutional layer performance may be considerably enhanced by adopting smaller steps.

The ResNet study, which was suggested by K. He, X. Zhang, S. Ren, and J. Sun [8] in 2016, is titled "Deep Residual Learning for Image Recognition." The research suggested a deep convolutional neural network with a number of significant advancements, drawing inspiration from models like LeNet, AlexNet, and VGGNet.

The ResNet architecture is composed of 152 layers, with majority of the layers being convolutional. The paper proposed the concept of Residual Connections, which enables information to be bypassed through certain layers and passed directly to the deeper layers. The use of residual connections tackles the problem of vanishing gradients which occurs quite commonly in very deep networks, as it allows information to be propagated more easily through the network.

The ResNet paper also introduced several other key innovations, including the use of batch normalizations to normalize the activations of network, speeding up training and thus improving the performance of the network. The paper also introduced bottleneck architecture, which involves reduction of number of parameters in the network while maintaining the expressive power.

In recent years, deep learning architectures have been extensively explored for pattern recognition tasks in various domains, including character recognition. S. Acharya et al. [9] suggested a deep learning-based approach for character recognition in languages like Hindi, Nepali, and Marathi that employ the Devanagari script. Their recommended design made use of deep belief networks (DBNs) and convolutional neural networks (CNNs) to achieve great accuracy in character recognition.

The 92 thousand pictures of 46 distinct classes of Devanagari characters that were carefully separated from handwritten texts made up the dataset utilized in this study. The authors used dataset increment and dropout as two strategies to increase the precision of their model. By using transformations like rotation, scaling, and translation on existing training samples, dataset increment includes creating new training examples from them. Dropout is a common regularization technique used in machine learning which avoids overfitting by randomly dropping mentioned neurons and making sure that the algorithm is able to learn features from training data and apply those knowledege to

identify unseen data

The proposed deep learning architecture achieved an accuracy of 98.47% on the Devanagari character recognition task, which is a highly accurate result. This study demonstrated the effectiveness of using deep learning techniques, along with dataset augmentation and dropout, for character recognition tasks. Such approaches can be useful in various applications, including optical character recognition, document digitization, and natural language processing

Hanmandlu et al. [10] have made an important contribution to the field of Devanagari character recognition by proposing a method that employs a modified exponential membership function fitted to a fuzzy set for the identification of handwritten Devanagari characters for Hindi. This approach was motivated by the observation that Devanagari character recognition is a difficult task, and coarse identification can be helpful in achieving accurate recognition rates. The authors achieved an overall recognition rate of 90.65%, which was a significant improvement at the time when OCR of handwritten Devanagari script was still in its infancy. Their work has paved the way for further research in this area, and it provides a basis for future improvements in the accuracy and robustness of Devanagari character recognition systems.

S. Shelke, et al. [11] proposed a novel approach to the classification of handwritten characters, by collecting a large dataset from over 40 writers.The dataset was created to guarantee that the final classification accuracy was independent of the structural classification results. There are a total of twenty-four classes after the characters were first divided into six classes and then four more classes. Several feature extraction techniques, such as the Fourier descriptor, Zernike moments, and wavelet transform, were used to achieve the classification.. A neural network was used to accomplish the categorization, and a 95.4% overall recognition rate was attained.

The proposed approach has contributed to the development of handwriting recognition systems, especially for Devanagari script. The large dataset collected from multiple writers has ensured the diversity of the samples and has led to the improvement of the recognition accuracy. The use of multiple methods for feature extraction has also contributed to the accuracy, as different methods can capture different aspects of the characters. Overall, this research has paved the way for the development of more accurate and robust systems for Devanagari handwriting recognition

In their study, Shelke et al. [12] proposed a fuzzy-based approach for recognizing handwritten Devanagari characters, which improved upon previous crisp classification methods. They used the fuzzy c-means clustering algorithm to extract features from the

dataset and found that the fuzzy-based approach provided better accuracy and minimized the computation required for identification. The study showed that fuzzy-based classification is a promising approach for handwritten character recognition and has the potential to outperform traditional crisp classification methods.

S. Shelke, et al. [13] in their study have claimed reduction of a multistage feature extraction and classification approach that leads to incorrect categorization of structurally identical properties as proposed in [11] and fuzzy based pre-classifier proposed in [12]. The study concluded that for complex characters, increasing the number of structural classes improved the performance of the system, and optimization improved the recognition rate.

P. K. Sonawane. et al. [14] have employed 16870 pictures of 22 often used Devanagari consonants to train a strong CNN called as AlexNet., achieving a validation accuracy of 94.49% and test accuracy of 95.46%. The study concluded transfer learning as a better alternative due to fast training and only requiring few samples.

T. Rani Das, et al. [15] have proposed a system for recognition of handwritten Bangla characters using Extended Convolutional Neural Network. The Bangla alphabet consists of total 84 classes of letters- 39 consonants, 11 vowels, 10 numerals and 24 combined characters.The model's accuracy rates for Bangla numerals were 99.50%, vowels were 93.18%, consonants were 90%, and mixed characters were 92.25%.

Joshi and Risodkar [16] have proposed a system for recognition of Gujarati Handwritten characters into machine editable format. For classification, K-nearest neighbor (KNN) and NNC classifier were used. The model achieved overall accuracy of 78.6%.

Prashanth, Mehta, et. al [17] have proposed a system for recognizing handwritten Devanagari characters, using modified LeNet and AlexNet CNN architectures. The goal of the experiment was to create a dataset of 38,750 pictures of Devanagari vowels and numerals, which was made available to other academics working in this field. The data was gathered from more than 3000 people of various ages. The segmentation method here was used to extract each character, and the dataset was used for the tests. Three different CNN architectures were experimented on; CNN, modified Lenet CNN (MLCNN) and Alexnet CNN (ACNN). Using CNN, accuracy was 96 percent on training data and 94 percent on unobserved data; MLCNN achieved these accuracy rates at a lower cost while ACNN achieved them at a higher rate. A minimal loss of 0.001 percent was discovered after a series of studies on the data using various combination splits of the data.

M. Mhapsekar, P Mhapsekar, et. al [18] proposed a system for recognizing handwritten Devanagari characters using Residual Neural Network (ResNet) model. The ResNet 34 and ResNet 50 designs were employed, and the outcomes were compared with the most advanced convolutional neural network architecture, which has four and eight layers, respectively.

Darmtasia and M. I. Fanany [19] has suggested a machine learning model based on Support Vector Machines (SVM) as a superior classifier and Convolutional Neural Network (CNN) as an effective feature extraction tool for identifying handwritten characters on form documents. Using L1 loss function and L2 regularization, it combines CNN with linear SVM. On 10 distinct test form documents, the system provides an accuracy rate of 83.37 percent. after combining character recognition, segmentation, and pre-processing into a single system.The system creates an editable text version of its output.

Marcus Liwicki et al. [20] have introduced new connectionist approach to on-line handwriting recognition. According to its the results, Connectionist Temporal Classification (CTC) can immediately train the network to label unsegmented sequence data with a word recognition rate of 74.0%, as opposed to 65.4% when utilizing a recognition method based on a previously built HMM. Long short-term memory blocks in a bidirectional recurrent neural network were utilized.

Tobias, Fischer [21], The online recognition of handwritten characters using capacitive sensors has been evaluated using bidirectional recurrent neural networks with GRU, CNN, HMM, random forests, MLP and SVM. The greatest accuracy, 91.4 percent, was provided by the bi-directional RNN Recurrent Neural Networks. While other approaches did not offer improved accuracy, stochastic weight averaging boosted the model's performance to 95,2% and layer normalization to 91.5%

T. Bluche, J. Louradour, and R. Messina [22] have developed attention-based model for end-to-end handwriting recognition utilizing a multidimensional LSTM network and the deployment of covert and overt attention. The input paragraph does not need to be segmented for the system. Without any preceding line-segmentation, it can read bidirectional scripts.

B. Shi, X. Bai et al. [23] have proposed the Convolutional Recurrent Neural Network (CRNN), a revolutionary neural network design. The CRNN can analyze input images of different sizes and produce predictions of different durations. In the training phase, it is not necessary to provide comprehensive annotations for each individual component (such as characters). Fully linked layers from traditional neural networks are not employed in CRNN. It may be used for tasks like Chinese character recognition and

optical music identification, which both require visual sequence prediction.

Without segmenting handwritten line text image into words or characters, S. Gautam [24] has utilized CRNN to recognize it. The RNN model is trained using LSTM with Alex Graves CTC loss to solve the alignment issue in handwritten data.

B. Liu, et al. [25] have proposed handwritten text-line recognition reference solution using CNN only and CTC method with PyTorch framework and Intel OpenVINO toolkit.The output of CTC based text recognition is further decoded with beam search for high accuracy. This model has metric of character error rate 2.49% for language model and 6.38% for without language model.

3. REQUIREMENT ANALYSIS

3.1. Hardware Requirements

No hardware components were used specifically for the development of this project, as the project is entirely a software-based project that only requires a normal PC as a hardware requirement for development. Machine Learning projects usually need a powerful PC with high-end CPU and GPU for training the models, due to high data processing required. However, the model was trained in Google Colab notebook, which trains the model in the cloud using the GPUs in remote computers, thus only requiring a normal PC for developing the project.

The final build of the project will require any normal PC or handheld device for the character recognition and transliteration.

3.2. Software Requirements

3.2.1. Google Colab Notebook

A Jupyter Notebook environment is provided by Google Colaboratory, also known as Google Colab, so that Python code may be run fully in the cloud through a web browser without the need to install any Python libraries on the local machine. Colab provides GPU for training the ML models, accelerating development by reducing hardware requirements.

3.2.2. Tensorflow

TensorFlow software library for deep learning and machine learning was developed by the Google Brain Team.. TensorFlow's key characteristic is that it offers a flexible, scalable framework for developing and deploying machine learning models, with an emphasis on inference and training on huge datasets. A library of pre-written operations, a mechanism for automatically differentiating those operations to carry out gradient-based optimization, and a computation graph representation of mathematical operations make up the main elements of TensorFlow.

3.2.3. Keras

Keras is a popular option for researchers and practitioners because it offers a straightforward and understandable interface for creating and training neural networks. Different

network topologies, including as feedforward networks, LSTM networks, RNNs, and CNNs are easy to build and experiment with using Keras. Additionally, Keras provides tools, algorithms, and pre-trained models for data preparation and model evaluation.

3.2.4. scikit-learn

It also provides functions for model evaluation, model selection, and feature engineering.One of the strengths of scikit-learn is its focus on consistency and simplicity. All algorithms share a common API for fitting, predicting, and transforming data, making it easy to switch between models and try out different approaches. Additionally, scikit-learn provides extensive documentation, tutorials, and examples, making it a great resource for both beginners and experienced practitioners.

3.2.5. numpy

A well-known Python package for numerical computing and data processing is called NumPy. To execute sophisticated mathematical operations like linear algebra, use Numpy. Large volumes of data may be effectively stored and processed mathematically using NumPy arrays. The arrays can be indexed, sliced, and otherwise modified in a variety of ways, and they can have any number of dimensions. Additionally, NumPy offers a wide range of functions for carrying out mathematical operations on arrays, such as random number generation, linear algebra operations, and mathematical functions.

3.2.6. Pandas

A robust and free Python library for data manipulation and analysis is called Pandas. It offers tools for working with structured data as well as data structures for effectively storing huge datasets. Additionally, it offers a wide range of tools for aggregating, cleaning, and manipulating data.Along with its fundamental data structures and operations, Pandas also works well with Matplotlib and NumPy, two major libraries in the scientific Python environment.

3.2.7. Matplotlib

Python developers frequently utilize Matplotlib, a charting tool, to build static, animated, and interactive visualizations.Line plots, scatter plots, bar plots, histograms, pie charts, error bars, box plots, and more are just a few of the many types of plots you may make with Matplotlib. Matplotlib has a variety of methods for working with photos in addition to its plotting features, such as reading and writing images from different file

formats.

3.2.8. cv2

The OpenCV (Open Source Computer Vision Library) library's Python module is called cv2. A large selection of image-related capabilities are offered by the open-source software library known as OpenCV for computer vision and machine learning. The image processing operations (such as filtering, morphological operations, and color space conversions) as well as utilities for reading and writing pictures are all provided by cv2. The image processing tools offered by cv2 include filtering, morphological operations, color space conversions, and more..

3.2.9. Flask

Flask is a lightweight and flexible framework, providing a simple and intuitive interface for developing web applications and APIs, with features for routing, template rendering, request handling, and support for various database systems. Flask also provides extensions for adding additional functionality such as authentication, database integration, and caching. It also work well while integrating with machine learning models.

3.3. Non Functional Requirements

The non functional requirements are listed below:

- **Performance**: The system is efficient in resource utilization while accurately predicting the characters.
- **Scalability**:The system should be scalable i.e. it should be able to recognize large number of characters as per demand.
- **Reliability**: The system should classify individual characters and use VGG CNN architecture.
- **Usability**: Since this system uses a simple mobile application so any user familiar with mobile apps can run it.

3.4. Feasibility Analysis

With limited time and resources in hand we should also consider the feasibility of the project.

3.4.1. Technical Feasibility

There are a lot of technical difficulties to be dealt with for the system to work properly. The difficulty of no dataset was solved by manually collecting images of characters and performing augmentations for increasing the volume of data upon which the data was trained. Similarly, due to computating devices limitation was solved by using Google colab for training the model. Google colab provides access to GPU which speeds up the training process. Our system is built using python language which is a cross platform language which allows system to be operated in any Os.Thus, the project can be considered technically feasible.

3.4.2. Operational Feasibility

Once the model is trained and targeted accuracy is met. The model is then deployed as a web application through the use of Flask framework. The main advantage of Flask framework in the context of our project is that Flask is a lightweight and flexible framework, and it allows us to build user-friendly web applications that are lightweight and easy on resources.

3.4.3. Economic Feasibility

We are designing our project in such a way that the installation cost will be minimum for the user. As the project is completely software based, so it is cheap economically. There is no need of purchasing any software licenses to run the system making the operation economically feasible.

3.4.4. Functional Feasibility

Although the lack of dataset for the project, dataset was prepared manually by collecting images from manuscripts and handwritten characters. Due to the availability of data augmentation techniques it was not difficult for the creation of dataset. Due to the availability of google colab it was very easy for training the model in rapid manner. The accuracy of the model was also found to be quite satisfactory.

4. SYSTEM ARCHITECTURE AND METHODOLOGY

4.1. Theoretical Background

4.1.1. Convolution Operation

The Convolution method, used to extract distinct features from images, is a key component of the architecture of Convolutional Neural Networks. A narrow window called as a filter or kernel is moved across the picture during convolution. The output feature map, which contains information on the presence or absence of particular input picture attributes, is formed by performing a dot product operation between the filter and the overlapping pixels of the image. The model learns the filter weights during the convolution operation's training phase using backpropagation. In other words, the network will proactively identify the most effective filters for identifying traits in the incoming data. The stride (i.e., the number), input size, and filter size

4.1.2. MaxPooling

Max Pooling is a typical down sampling method, which decreases the spatial dimension of the output feature maps. Max Pooling involves moving a tiny window across the input feature map (usually 2×2 or 3×3) and taking the highest value in each window as the output. By doing so, the feature map's size is decreased by a factor of the window size while still retaining the crucial details on the existence of particular features in the input. In order to improve the network's computational efficiency and lower the likelihood of over-fitting, max pooling is frequently applied after convolutional layers. This reduces the spatial dimensions of the output feature maps. Translation in-variance, which refers to the network's ability to detect specific characteristics even if they are significantly changed in the input.

4.1.3. Activation Function

The activation function in a neural network is a mathematical function that is applied to a network node's output and decides whether or not the output of the node should be activated based on the weighted total of the inputs.

ReLU and softmax were the two activation functions employed in the project's models.

4.1.4. ReLU Activation

The Rectified Linear Unit is a piecewise linear function that, if the input is positive, outputs the value directly; else, the output is zero.

It is mathematically [26] expressed as,

$$f(x) = max(0, x) \tag{4.1}$$

4.1.5. SoftMax Activation

The SoftMax function is a mathematical function that takes in a vector of any arbitrary values and normalizes it within the range of $[0, 1]$, the vectors which all add up to 1. Softmax function converts the vector of arbitrary input values into a probability distribution.

Mathematically, softmax function [5] is representated as,

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4.2}$$

Where $\sigma(z)_i$ is the i^{th} component of output vector $\sigma(z)$, and e is the base of the natural logarithm.

A neural network's output layer employs the SoftMax function, which transforms a model's output into a probability distribution over a range of potential classes. The output with the greatest likelihood is frequently selected as the projected class.

4.1.6. Dropout

Dropout is a regularization technique for neural networks, introduced by Geoffrey Hinton, et.al, in 2012. As illustrated in figure 4.1, dropout involves randomly setting a fraction of activation of neurons to 0. This reduces the amount of information available to each layer, forcing the network to learn multiple independent representations of the same data. This makes the network more robust to overfitting.

In practice, during each forward pass, each activation in the network is set to zero with a certain probability (e.g., 50%), effectively dropping out that activation and its corresponding node in the network. During the backward pass, the gradients are computed normally and then multiplied by a factor that corresponds to the keep probability. This allows the network to learn to "turn on" different nodes and combinations of nodes to

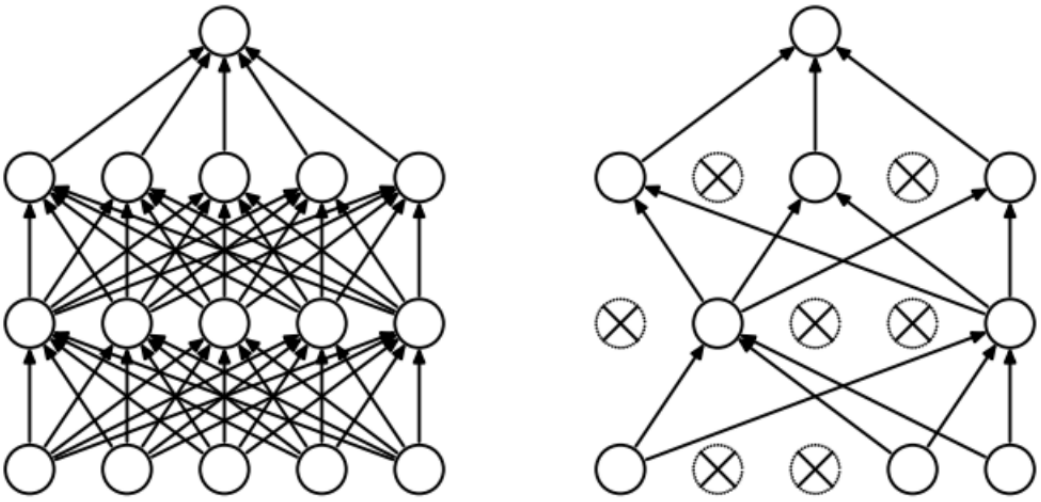


Figure 4.1: Dropout Layer

model the data.

In a deep neural network architecture, dropout layers are inserted between the dense layers or the convolutional layers. The keep probability is typically set to a value between 0.5 and 0.8, depending on the size and complexity of the network and the size of the training data.

4.1.7. Backpropagation

Backpropagation is an algorithm used to train neural networks by adjusting the weights of the neurons.

The network is given an input during training, and its output is compared to the intended output. Calculations are made to determine the error, which is the difference between the desired and actual output. Gradient of the loss function with respect to connection weights is calculated using the error value. The weights of the neural networks are then updated using the gradient in an iterative manner.

The calculation of sparse categorical crossentropy loss occurs when there are more than two label classes. It does not include summation and just computes logarithm once, in contrast to categorical crossentropy loss.

According to Equation (4.3), the loss function is defined as the negative logarithm of the predicted probability of the true class [27].

$$L(w) = -\log(\hat{y}_y) \tag{4.3}$$

The loss function's gradient is calculated for each weight using chain rule and optimized using various optimization algorithms. Because the results of the Adam optimizer are often better than those of other optimization algorithms, they have a faster calculation time, and they need less tuning parameters.

Kingma and Ba [28] proposed the following equations: (4.4),(4.5),(4.6),(4.7),(4.8); for optimizing the weights in Adam optimizer.

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)} \tag{4.4}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + ((1-\beta_2)\nabla_w L^{(t)})^2 \tag{4.5}$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^t} \tag{4.6}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^t} \tag{4.7}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \varepsilon} \tag{4.8}$$

Where, β_1, β_2 and ε are tiny scalars to prevent division by zero, forgetting factors for gradient, and second moments of gradient, respectively. The rate of learning is η.

4.1.8. Confusion Matrix

A confusion matrix is a table or matrix that shows how many of the model's predictions were right and wrong. The main purpose of a confusion matrix is to figure out how well a classification model performs.

The parameters used for the confusion matrix are: True Positives(TP), True Negatives(TN), False positives(FP) and False Negatives(FN). These metrices values can be taken to calculate other parameters such as: accuracy, precision, recall and F-1 score

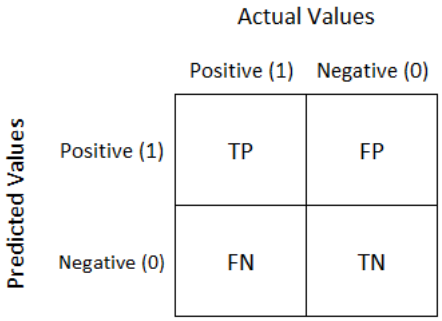


Figure 4.2: Confusion Matrix

True Positives(TP): The number of times the model rightly predicts the positive class.
False Positives(FP): The number of times the model wrongly predicts the positive classes.
True Negatives(TN): The number of times the model rightly predicted the negative classes.
False Negatives(FN): The number of times the model wrongly predicted the negative classes when it is actually positive.

4.1.9. Accuracy

A common statistic used in machine learning to assess a model's performance is accuracy. Out of all the examples, it calculates the percentage of instances that were properly categorized. In other words, it is the proportion of the model's total predictions to those that were made correctly. The formula for accuracy [27] is:

$$accuracy = \frac{\text{total correctly classified instances}}{\text{total number of instances}} \tag{4.9}$$

Accuracy is often used in classification problems where the objective is to predict the class label of an instance based on a set of features identified. For example, in a binary classification problem where the classes are "positive" and "negative", accuracy measures how well the model is able to correctly classify instances as either positive or negative.

4.1.10. Precision

Precision is defined mathematically as the ratio of the model's actual positive predictions to all other positive predictions. In other words, it also indicates the ability of a model to reproduce the same output if tested over the same data.

Mathematically, precision [29] can be represented as:

$$Precision = \frac{TP}{TP+FP} \tag{4.10}$$

4.1.11. Recall

Recall is mathematically defined the number of true positive cases divided by the number of all positive cases. It is a measure of the accuracy of a model in identifying positive cases.

Mathematically, recall [30] is defined as:

$$Recall = \frac{TP}{TP+FN} \tag{4.11}$$

4.1.12. ROC Curve

ROC is short for Receiver Operating Characteristic, is defined as a graphical visualization of the performance of a binary classifier system. This is a common metric to determine the performance of binary classification problem. It presents the True Positive Rate (TPR) oppose to False Positive Rate(FPR), where a threshold value is predefined that acts as a boundary between positive and negative predictions.The formula for TPR and FPR [31] can be represented as:

$$TPR = \frac{TP}{TP+FN} \tag{4.12}$$

$$FPR = \frac{FP}{FP+TN} \tag{4.13}$$

4.1.13. F_1 score

F_1 score is also a commonly used evaluation metric that is used in unseen data, to evaluate its performance. It is mathematically calculated by taking Harmonic Mean between Precision and Recall.It is used in the case when there is imbalance between positive and negative classes, as it gives equal preference to both precision and recall. A high F_1 score will assist to conclude that there is a good balance between precision and recall. An F_1 score of 1 shows that there is perfect precision and recall, whereas F_1

score of 0 will indicate the worst case scenario of precision and recall.

The F_1 Score is calculated using the following formula [32]:

$$F_1 = \frac{2(Precision \times Recall)}{Precision + Recall} \tag{4.14}$$

where eq 4.10 is used to calculate Precision, and eq 4.11 is used to calculate Recall.

4.2. Choice of Supervised CNN for the system

A form of CNN called a supervised Convolutional Neural Network (CNN) trains the model using labeled data. In supervised CNN, a matching label is present for each input picture that the model is attempting to predict. The objective of supervised learning is to develop a mapping that properly connects the input and the output and generalizes well to new data.

In a supervised CNN, the network is trained by providing it with a large dataset of labeled images, where the weights of the model is adjusted based on the error between the predicted and true labels. During the training process, the network learns to recognize different features in the input images and combines them to make accurate predictions.

Once the network is trained, it can then be used to predict the labels of new, unseen images. The performance of a supervised CNN can be evaluated by its accuracy on a validation or test set of labeled images.Supervised CNNs are mostly used in an array of image recognition tasks including character recognition systems. Convolutional Neural Networks (CNNs) apt choice for character recognition systems for several reasons:

- **Translation Invariance**: A CNN can learn to recognize characters regardless of their position within an image, as long as the character is recognizable. This makes them robust to the variability in character placement within an image.
- **Feature Extraction**: CNNs have multiple layers that can automatically extract relevant features from an image, such as lines, edges, and shapes. These features are then used to make a prediction about the character in the image.
- **Robustness to Noise and Degradation**: CNNs can handle some level of noise and degradation in the image, making them more robust compared to traditional image recognition techniques.

- **High Accuracy**: CNNs have achieved high accuracy in character recognition tasks with use of deeper networks and advanced techniques such as transfer learning.
- **End-to-end Learning**: CNNs can learn the complete mapping from input image to output label, making them well-suited for end-to-end character recognition systems.

Unsupervised learning algorithms are those alogrithms that finds undelying pattern in data and cluster them accordingly without defining the labels implicitly. These algorithms are used for clustering, dimensionality reduction. But, in our case of character recognition and classification, our objective is to classify a image into a pre assigned data labels. Without providing implicit data label the model is unable to classify the input images. Using unsupervised learning, the model my determine the underlying features within an image but will not be able to classify them. In comparison, supervised learning algorithms can be used to identify specific features of an image and assign them to one of the predetermined classes.

For our project, four different CNN architectures were used: AlexNet, LeNet, ResNet and VGG-16. Upon training the data over several epochs in different architectures, varying results were obtained, with varying accuracy and precision values. VGG-16 has a deeper architecture compared to AlexNet and LeNet, which allows it to learn more complex and subtle features and patterns in the data more than the aforementioned neural network architectures.

One prime reason why VGG-16 was chosen over AlexNet and LeNet is not only because of the deeper architecture of VGG-16 - which uses 16 layers over 5 layers of AlexNet and 8 layers of LeNet. VGG-16 uses a much smaller filter size like (2×2) or (3×3), compared to larger filter size of LeNet and AlexNet, which is usually either (5×5) or (7×7). Smaller filter size enables a machine learning architecture to extract for finer details from a data, which means the model can recognize subtle differences between the features of similar data, thus increasing the accuracy of recognition.

However, the deciding factor was whether to choose VGG-16 or ResNet, since both the architectures are deep learning architectures, with 16 layers of VGG-16 and 18 layers of ResNet, and both the architectures using (3×3) filter size. VGG-16 was chosen by the virtue of a more accurate result in recognition of finer details, as we observed that ResNet model had higher chances of overfitting and thus producing wrong results in recognition of data.

4.3. The VGGNet Architecture

VGG, which stands for Visual Geometry Group, is a very deep convolutional neural network with several multiple layers; the number of layers vary depending on different VGG architectures. VGG replaces the larger kernel sizes with several layers of small kernel sizes, usually (3×3), one after another, thus making significant improvements over feature extraction compared to other CNN models.

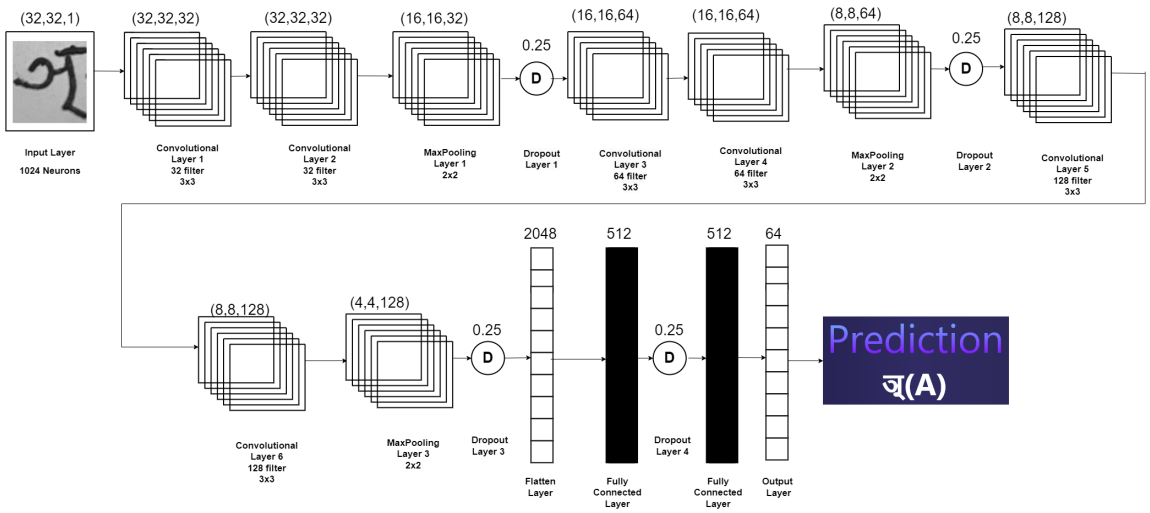


Figure 4.3: The VGG-16 architecture

The VGG network is constructed with very small convolutional layers, and depending on the different convolutional layers involved, the variants of the network exists. In VGG-16 architecture, the model consists of 16 different layers, with 10 convolutional layers and 3 dense layers.

4.3.1. Input Layer

The convolutional layer with 64 filters in the input layer of the VGG architecture has a kernel of (3 times 3), a stride of 1, and padding of 1. This layer's function is to analyze the source picture and extract basic elements like corners and edges.

The input to the VGGNet architecture is usually an RGB image of size 32×32 pixels. The image is preprocessed by subtracting the mean RGB values of the training set from each pixel and then scaling the pixel values to be between 0 and 1.

4.3.2. Convolutional Layers

The prime feature of VGG architecture is the use of Convolutional Layers. VGG uses convolutional layers with the kernel sizes of (3×3). The convolutional layers are each

used in succession with max pooling layer, with varying number of filters applied to the output of each max pooling layer.

- The first and second convolutional layers in VGG applies 64 filters with kernel of size (3×3). The first convolutional layer takes input from the input layer.
- The third and fourth convolutional ayers in VGG applies 128 filters with kernel of size (3×3).
- The fifth and sixth convolutional layers in VGG applies 256 filters with kernel of size (3×3).
- The final four convolutional layers in VGG applies 512 filters with kernel of size (3×3). The layers are passed through max pooling layers in succession.

ReLU activation function is used in every convolutional layers, and the outputs of each convolutional layer is successively passed into the max pooling layer with kernel of size (2×2) and stride of 2. Upon passing through the max pooling layer, the original dimension of image input, which is 32×32, reduces in each convolutional layer, as illustrated in fig 4.3.

4.3.3. Max Pooling Layers

Along with the convolutional layers, multiple max pooling layers are employed in VGGnet. The max pooling layer includes taking the maximum value in each zone as the output after partitioning the input feature map into non-overlapping rectangular sections, typically of size (two by two). In VGGNet, stride 2 is frequently employed by kernels in max pooling.

Max pooling layers help to introduce translational invariance to the network. Translational invariance simply is the capacity of a model to extract the same feature from different location or position of images. For image inputs, translational invariance involves extracting the same feature from different parts of the image, regardless of the position. Additionally, max pooling reduces the number of parameters in the network to learn and avoid overfitting.

4.3.4. Dense Layers

In VGGnet, dense layers are usually implemented towards the end of the network, and is responsible for producing the final output of the network. The purpose of dense

layers is to take the output from convolutional and max pooling layers, which involves the high-level features of the image, and use them to make prediction of the class of the image.

In VGGnet architecture, two dense layers are present, with 4096 neurons each, followed by dropout layers in succession. Dropout layers are use subsequently after dense layers to prevent overfitting of the data. The first dense layer takes input from a flattened layer, which is a 1D vector, and applies ReLU activation function on the vector. The output of this dense layer is then passed through a dropout layer.

A activation function of ReLu is applied to output of the first dropout layer before it passes through the second dense layer. The second dropout layer, which again randomly removes some neurons to avoid overfitting, is then applied to the output of this layer. Finally, a thick layer with as many neurons as classes in the classification issue makes up the output layer. The final predicted probabilities for each class are produced by this layer using a softmax activation function.

4.4. Data Splitting

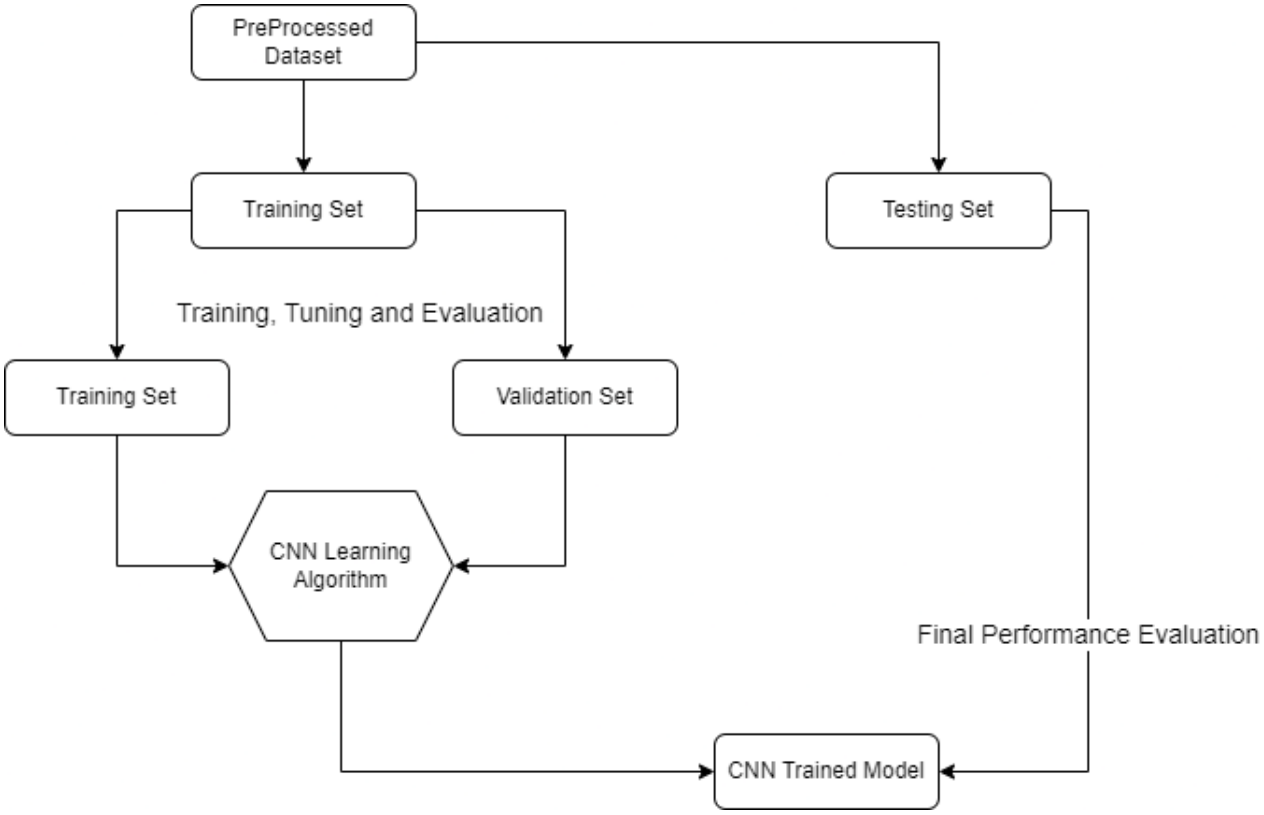


Figure 4.4: DataSplitting Technique for training the model

The original dataset is preprocessed by grayscaling, resizing. The original dataset is firstly split into training and testing dataset in 8:2 ratio. Then the remaining training

dataset is splited into training and validation dataset in the same ratio 8:2.

- **Training Data**: Training data is the data used by machine learning algorithm while learning, it is used to adjust the weights and biases of the network.
- **Validation Data**: Used during hyperparameter tuning as these data are not involved in training phase.
- **Testing Data**: Set of data that is completely unseen during the training process, These data is used to provide unbiased performance evaluation of the model. It can be an indicator for showing how the model will perform on unseen data.

4.5. System Architecture

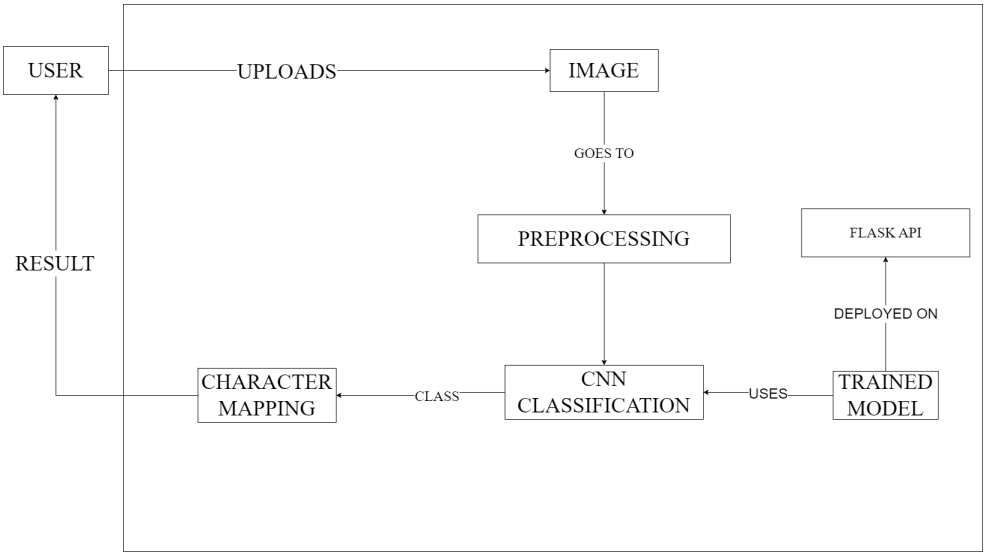


Figure 4.5: System Architecture Block Diagram

The website created using HTML, CSS, and JavaScript allows users to upload an image of an unknown character. The image is then preprocessed by resizing it to 32×32 pixels and converting it to grayscale. The preprocessed image is then fed into the CNN-trained model for character recognition. To train the model, the dataset is first split into training and testing sets. During the testing phase, the accuracy of the model is evaluated. The training data comprises two variables, X and Y, where X contains features of the dataset and Y corresponds to their respective labels. The model is trained based on the recognized features.

The Flask API is used to facilitate communication between the backend and frontend using HTTP POST and GET requests. The HTTP POST request sends the uploaded image file to the backend API by defining the API endpoint. Once uploaded, the image

is preprocessed in the FlaskAPI to make it compatible with the input layer of the trained VGG CNN model. The predict method is then called on the FlaskAPI, which provides the class of the uploaded image. The prediction is contained in the body of a JSON response, which is in key-value pairs. The value is then obtained in the frontend and displayed to the user

4.6. Sequence Diagram

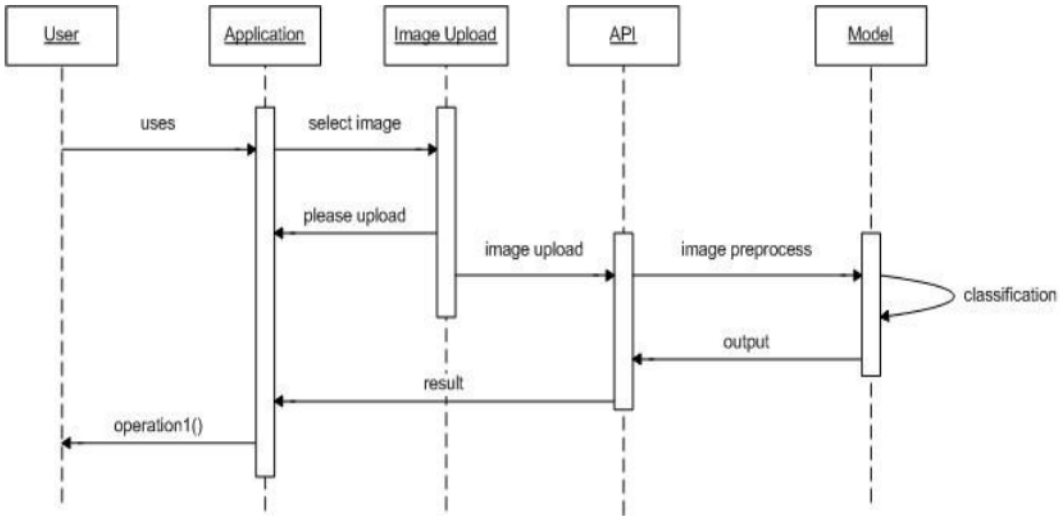


Figure 4.6: Sequence Diagram of the System

The user firstly uses application where they can select image for prediction purpose and if an empty image is tried to be predicted then the app does not allow user to upload the image. Then the image is uploaded into the FLASK API .he obtained image is then pre processed i.e. resizing and filtering and erosion. Then the trained model performs prediction on the unseen data and it provides the prediction to the frontend. The actual prediction is contained inside a JSON response which is then decoded to get the actual result which is then shown to the user.

4.7. Data Flow Diagram

The context contains one user, two processes and a data source. In this DFD the user is prompted to upload a image Then the trained model i.e. Prachalit Charcacter recognition system inputs image and the system then performs prediction. The trained model is run locally as well as the webpage and the prediction is send to the user using the website.

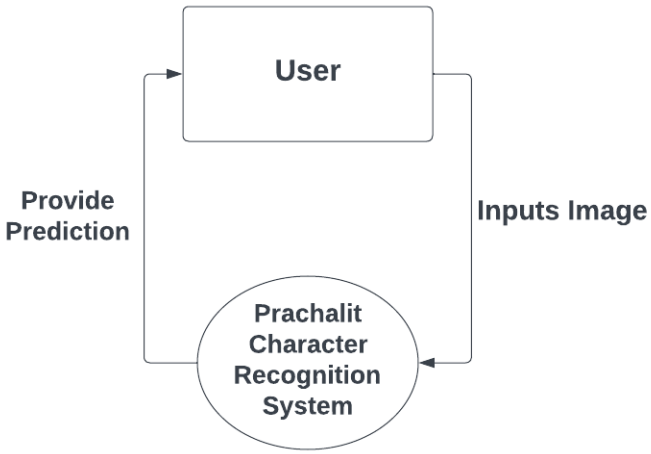


Figure 4.7: 0-Level DFD

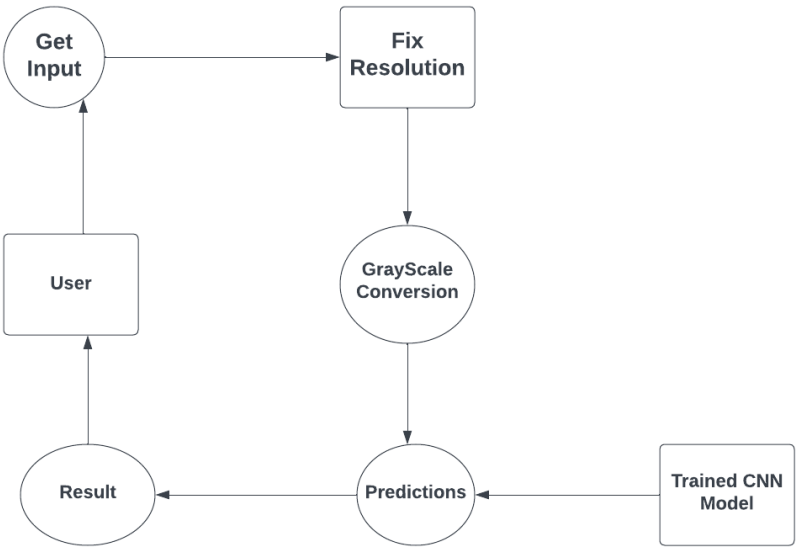


Figure 4.8: 1-Level DFD

4.8. Use Case Diagram

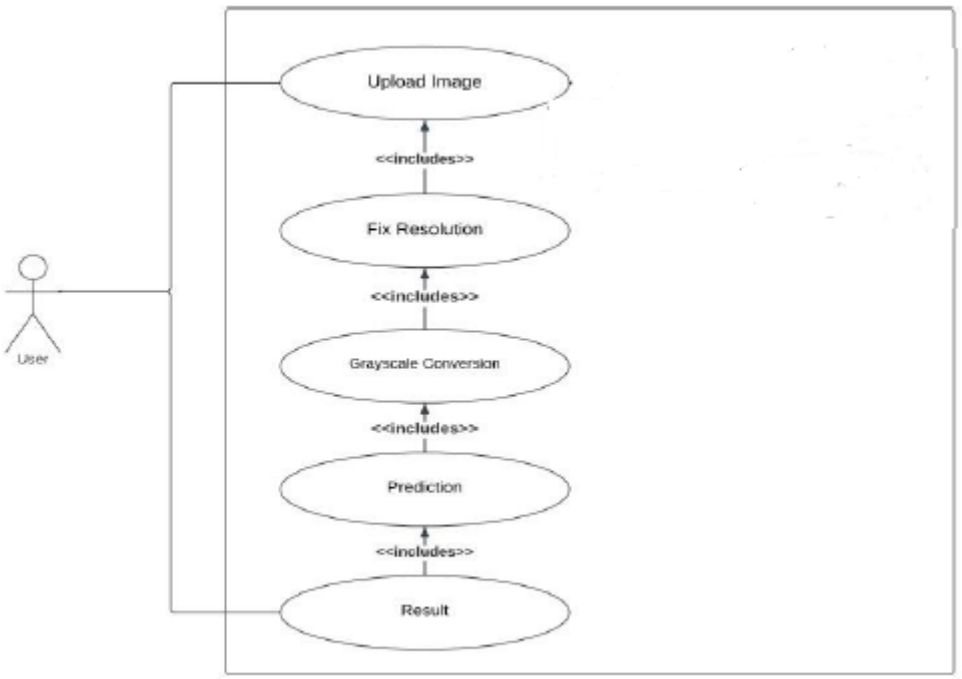


Figure 4.9: Use Case Diagram of the VGG trained model

The use case diagram illustrates the interaction between the user and the system. The user begins by uploading an image of the character they want to recognize using the Flask web application. The preprocessing step, which involves fixing the resolution, grayscaling, and resizing the image, requires the user to upload the image first. The trained model then predicts the uploaded image, and the result is provided to the user in the Flask web app user interface.

4.9. Activity Diagram

4.10. Algorithm

4.10.1. Data Preparation

Step 1: Load the dataset of 51200 images with 64 classes i.e. 800 images each class
Step 2: Preprocess the image and set the size to 32*32
Step 3: Then the resized image is converted to grayscale(1 channel)
Step 4: Split the data into training and testing sets
Step 5: Convert the labels into one-hot encoded vectors

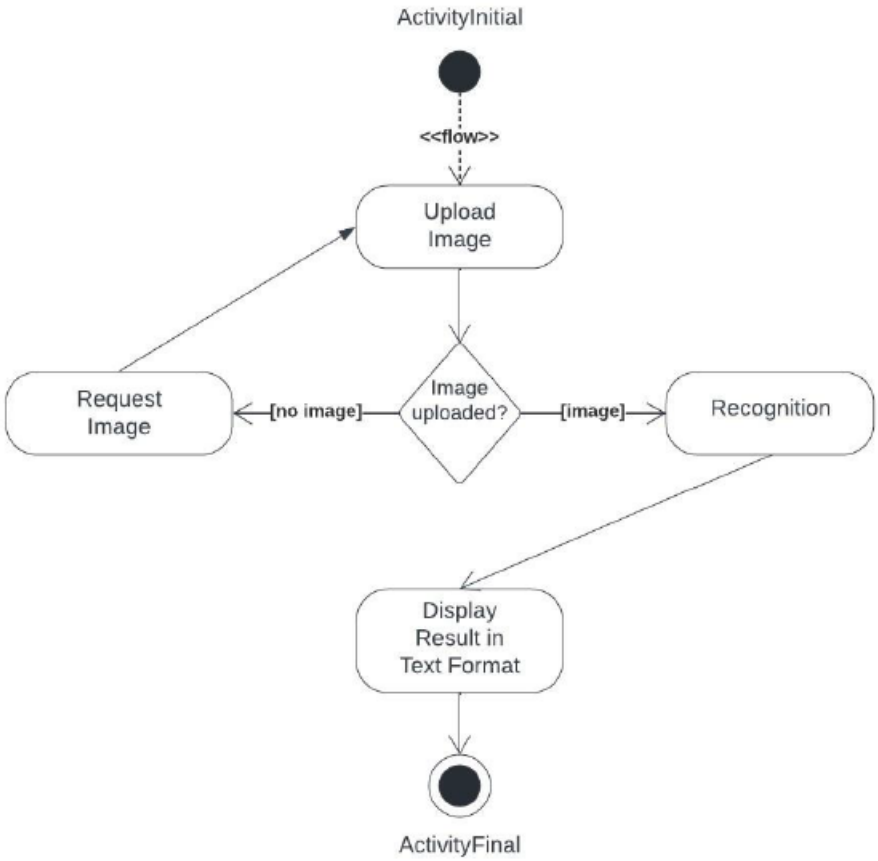


Figure 4.10: Activity Diagram of the System

4.10.2. Model Creation

Step 1: Import the required libaries including tensorflow and keras.
Step 2: Create a instance of model using Sequential.
Step 3: Add Following layers into the model.

- Conv2D layer takes an input of (32,32,1) applies 64 filters having a size of (3,3) with strides 1 and ReLu activation function
- Conv2D layer takes an input of (32,32,1) applies 32 filters having a size of (3,3) with strides 1 and ReLu activation function
- MaxPool2D layer performs max pooling operation on the output of first conv2D Layer. The pool size is (2,2), stride also being (2,2).
- 0.25 dropout rate is used for the dropout layer.
- Conv2D layer takes an input of (32,32,1) applies 32 filters having a size of (3,3) with strides 1 and ReLu activation function
- Conv2D layer takes an input of (32,32,1) applies 64 filters having a size of (3,3) with strides 1 and ReLu activation function.
- axPool2D layer performs max pooling operation on the output of fourth conv2D Layer. The pool size is (2,2), stride also being (2,2).
- 0.25 dropout rate for the dropout layer.
- Conv2D layer of 128 filters, a kernel size of 3, the same padding, and a ”ReLu” activation function.
- Conv2D layer of 128 filters, a kernel size of 3, the same padding, and a ”ReLu” activation function
- MaxPooling2D layer with a pool size of (2,2) and strides of (2,2)
- Dropout layer having a dropping rate of 0.25
- Flatten layer converts the tensor output of the previous layer (2D) to a 1D array, having activation function of ReLU.
- Dense layer with 512 units and ’ReLu’ activation function
- 0.5 dropout rate for the dropout layer.
- 64-unit dense layer with the activation function known as ”softmax”

Step 4: Return the model object

4.10.3. Model Training

Step 1: Import the required libraries for data processing, including numpy and scikit-learn.
Step 2: Using the train test split function, divide the original datasets into training and testing datasets in an 8:2 ratio.
Step 3: Split the remaining training set into training and validation sets using the train test split function in ratio of 8:2
Step 4: Definition of early stopping to prevent over fitting which monitors the validation loss defined by the parameter patience.
Step 5: Compile the model using 'ADAM' optimizer, loss function of 'sparse categorical cross-entropy', and 'accuracy' as the metric.
Step 6: Fit the model on the training set with validation data and early stopping callback using the fit() function of the model object. Save the history object returned by the fit function.
Step 7(a): Prepare plot between validation and training loss against number of epochs.
Step 7(b): Prepare plot between validation and training accuracy against number of epochs.

4.10.4. Model Evaluation

Step 1: Evaluate the model on testing data using metrics such as:
- 1.1: Accuracy
- 1.2: Precision
- 1.3: Loss
- 1.4: Recall
- 1.5: F_1 score
- 1.3: ROC Curve
- 1.4: Precision Recall graph
- 1.5: Confusion Matrix

4.10.5. Model Deployment

Step 1: Save the trained model and its weights to a file.
Step 2: Load the trained model and its weights into the system for deployment.
Step 3: Test the trained model on unseen images and verify its performance.
Step 4: Save the model in local in .h5 file format.
Step 5: Create Flask API and deploy the model.
Step 6: Send post and get request to get responses from Flask API.

5. IMPLEMENTATION DETAILS

5.1. Dataset Acquisition and Creation

For collection of dataset, we had two primary approaches: collecting handwritten data of school students, and collecting data from manuscripts. In the first approach, we collected handwritten data from volunteers. We visited various schools in Kathmandu, where Nepal Bhasa is now a compulsory subject at school level. We distributed the leaflet for writing the alphabets, and we primarily collected the handwriting of students from primary level, as shown in Fig 5-1. Thus obtained data was scanned and cropped for the individual characters of the script. Handwriting of adults were also collected in the same way.

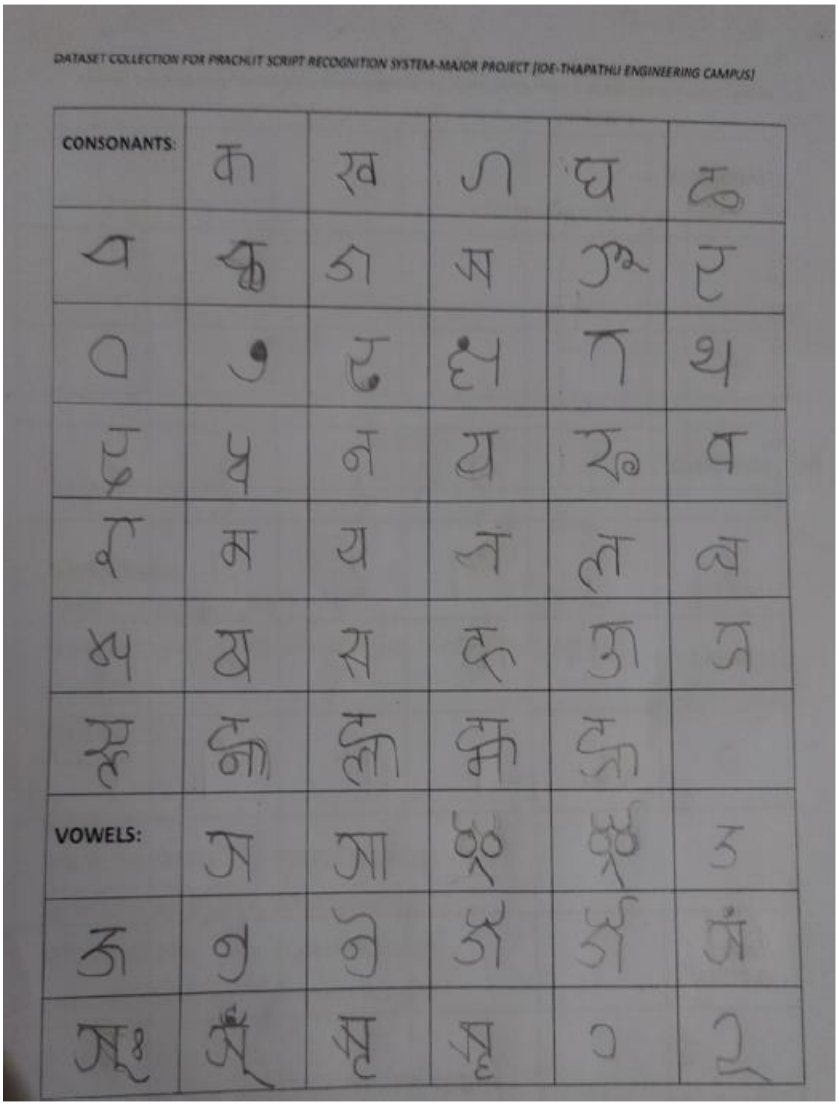


Figure 5.1: Handwriting data collection

In the second approach, we scanned the manuscripts and extracted the individual char-

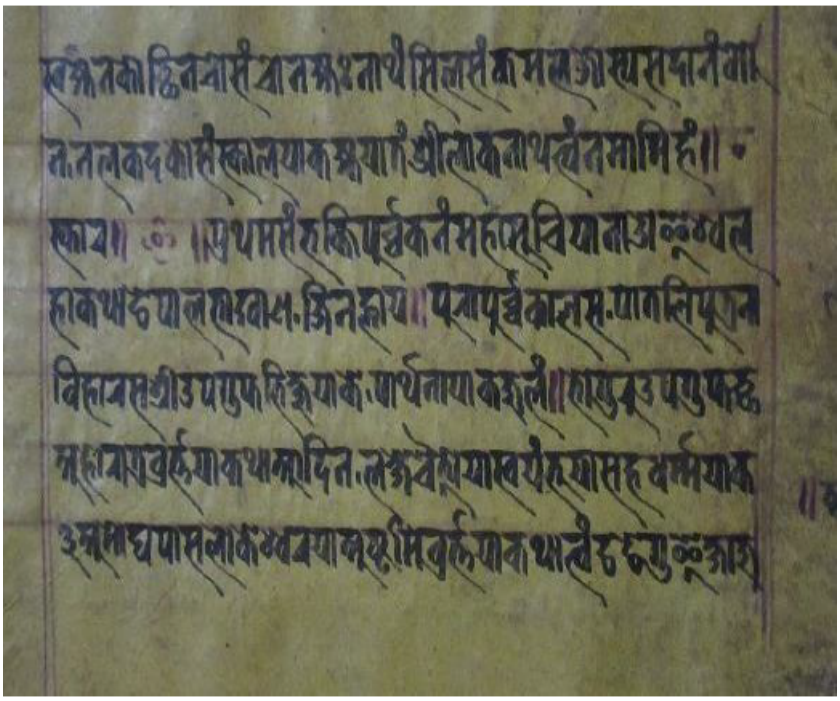


Figure 5.2: Scan of manuscript for data collection

acters from the manuscript by cropping the image. Manuscripts were obtained through both primary and secondary sources. Primary sources of manuscript were the actual physical manuscripts available to us, of which we took photographs and scanned the image. Secondary sources of manuscript were the pre-scanned manuscripts available on the internet, which was simply downloaded and cropped.

Almost 800 images have been added to our dataset for all 64 classes, resulting in a total of 51,200 images. These images have been collected and augmented using appropriate techniques such as rotation, shearing, and adding random noise.

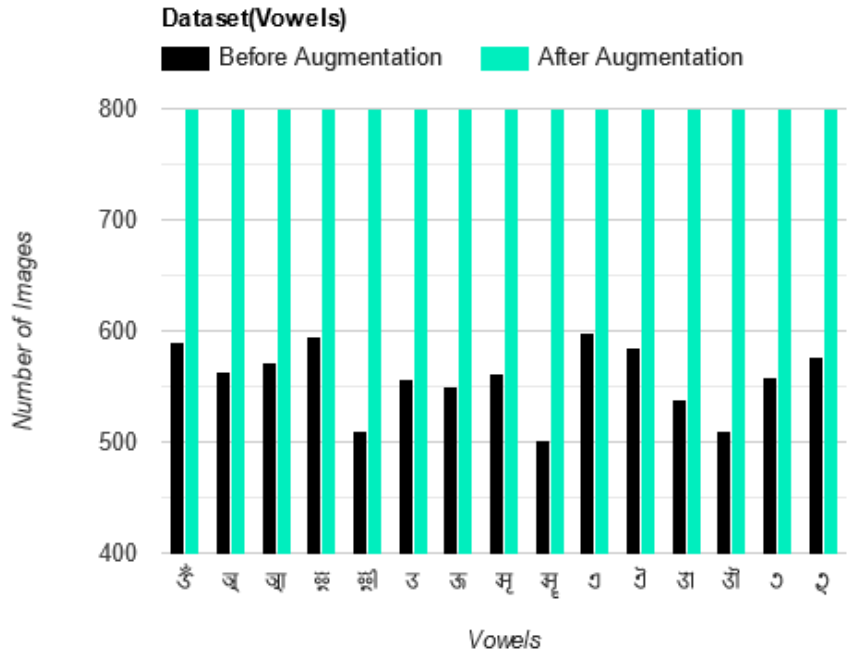
5.2. Image Distribution of Dataset Before and After Augmentation

Initially, the images collected during the dataset were divided into different classes, and grouped under three categories: digits, vowels and consonants. Since the dataset was created primarily by scanning manuscripts, it was observed that some characters were used more frequently than the others, which caused class imbalance, and thus augmentation was necessary before training the model. The image dataset was loaded using the OS and cv2 packages. Access to the directory and files is provided by the operating system library, and the cv2 library may be used to read the photos using their path.

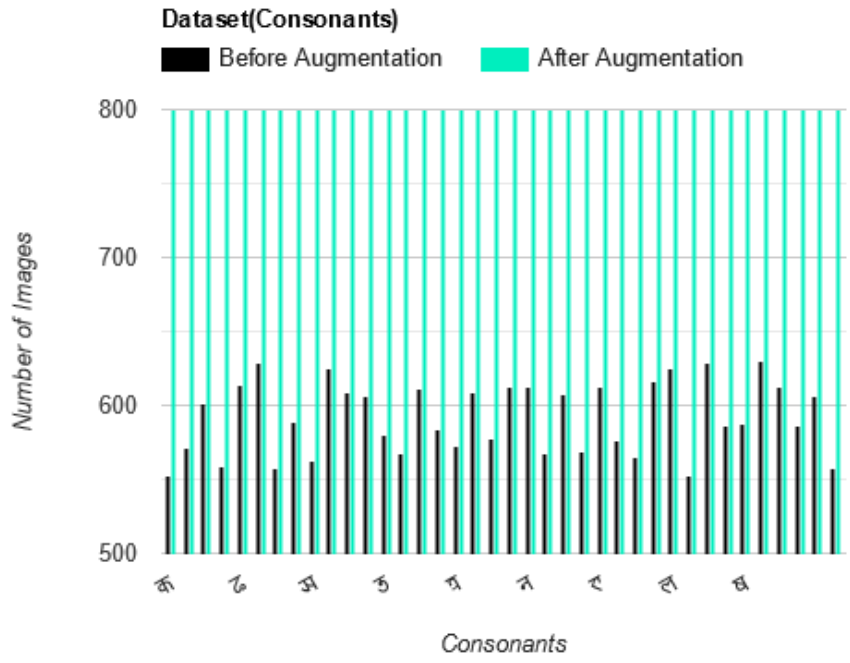
The dataset was later on combined, and the classes were not separated into digits, con-

sonants and vowels. The combined was then augmented, each class containing 800 images.

The distribution of the classes in the aforementioned categories are illustrated in the following bar graphs.

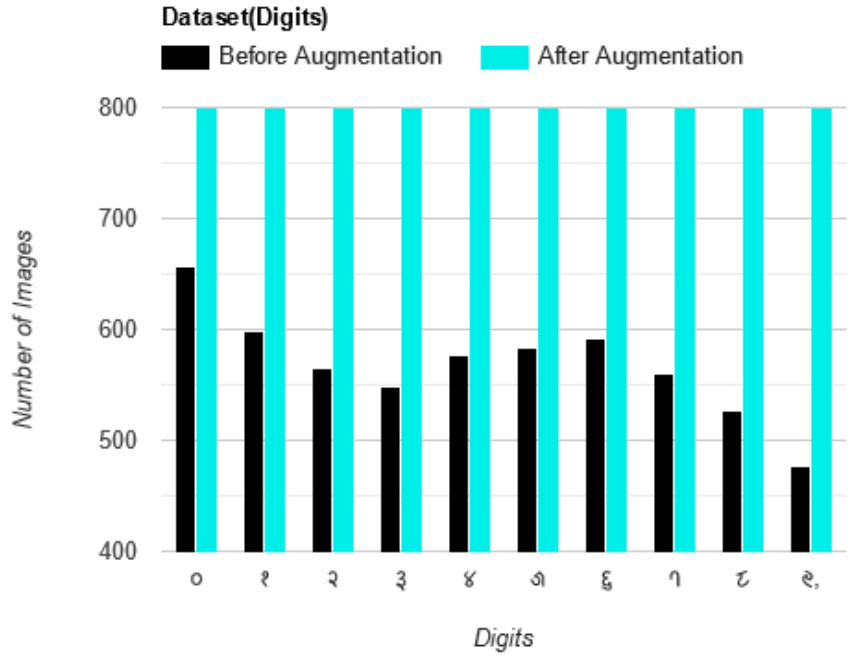


(a) Vowels



(b) Consonants

Figure 5.3: Image distribution among classes before and after augmentation



(c) Digits

Figure 5.3: Image distribution among classes before and after augmentation

5.3. Image Preprocessing

Preprocessing was a necessary process in implementation. This is because the images scanned from the sources were not uniform in nature, while a dataset consisting of uniform images were required for the model to be trained. Image preprocessing was carried out using the OpenCV cv2 library, where the images were converted to grayscale, followed by dimension reduction and resizing into 32×32 pixels resolution.

Figure 5.4: Image preprocessing procedure

5.3.1. Grayscale Conversion

The function for converting image to grayscale is available in the OpenCV cv2 package and turns a 3-channel RGB image into a single grayscale image. The grayscale conversion of the image is done through the following mathematical equation [33]:

$$Gray = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{5.1}$$

Where R, G and B are the red, green and blue colour values of the image, each of which is in a range of $[0, 255]$.

5.3.2. Resizing the image

All the images in the dataset were resized into 32×32 pixel resolution to maintain a uniform size input array. The OpenCV library provided the necessary functions to resize all the images in the dataset to a 32×32 resolution format.

5.3.3. Normalization

In image processing, normalization is often used to adjust the contrast and brightness of an image. It can help to bring out details in the darker or brighter areas of an image, or to make an image look more consistent overall.

Normalization in image processing typically involves scaling the intensity values of the pixels in an image such that they fall within a certain range, usually between 0 and 255 (for 8-bit images) or 0 and 1 (for floating-point images). This can be done by subtracting the minimum intensity value in the image from all pixel values and then dividing by the range of intensity values.

Normalization was carried out by converting the image into a NumPy array, and dividing the array values by 255.

5.4. Need of Image Preprocessing

5.4.1. Resizing of image

Resizing image in image processing involves reducing the images dimensions while maintaining its aspect ratio. Resizing is done in training dataset as well as in input image. The input image obtained can be of different sizes. So, resizing is done in the input image to make it compatible to be fed into the model as an input. Resizing of im-

age to a size of 32×32 reduces the computational need of the classification algorithm. Making the image of a standardized size it can be used to reduce the computational requirements of CNN algorithm to recognize the image. Resizing of image can improve the performance of the recognition algorithm. Smaller size images have smaller features to identify and are less susceptible to noise and degradation,making it easier for recognition.

5.4.2. Grayscaling the image

Grayscaling of image involves conversion of 3 channel into single image(gray). Grayscale conversion involves simplification of data as the image only contains gray colors. Grayscale conversion allows to reduce the amount of features to be processed for character recognition. This can be obtained because in contrast to 3 channel image, one channel image contains simplified features which are easier to extract and recognized by learning algorithm. Grayscale conversion also increases the computational efficiency of the model as it requires less computational resources to process. Grayscale conversion also increases the contrast of the images, color images have higher noise and distortion. Higher quality images of grayscale images provides more distinct features which the model can use to learn more effectively. It also introduces a standard among the unseen data, as different color images tends to have more variation among them. Converting of these images to grayscale allows to eliminate those variations.

5.4.3. Normalization of image

Normalization of images reduces the illumination differences among the input images. Elimination of illumination differences among the image can help machine learning model to extract meaningful features from the images. Normalization of images also helps in standardizing the unseen images. It can be used to scale the images to a specific range of pixel values. Doing so will allow model to recognize the images irrespective of their pixel values. It also helps to improve the image quality of the input. This can be used to reduce the noise and distortion inside a image. It can also be used to enhance the image quality as it increases the contrast of the features within an image. Normalization of image helps to remove the blurry and noisy features of an image.

5.5. Need of data augmentation

Data augmentation refers to the process of increasing the dataset of machine learning algorithm. In data augmentation different image transformation techniques such as: flipping, rotating, shearing, adding noise, changing contrast are applied to increase the dataset amount.

Rotation: Rotation is a technique of data augmentation in which a image is rotated along a certain angle. This operation is done in the dataset images to introduce the various possible configuration of the same image. It increases more variation in the dataset, giving more data points for the models to learn from.

Shearing: It has the process of changing the pixels of images along an axis, keeping another axis fixed. This can help to provide more examples of same data allowing the model to learn from higher amount of data. It helps to recognize images that are little skewed or distorted.

Changing brightness and contrast: Dealing with pixel values, changing the pixel values helps in adjusting the brightness and contrast of images. Allowing for increasing the similar data with varied level of pixel values, which allows the model to have more variations to learn from and make model more efficient in recognizing input of different contrast levels.

Increases the size of dataset: Image augmentation can help to create more sets of data from a preexisting dataset. One image can be augmented several times allowing to capture minute variations in the data. This also provides the model with a varied examples to learn from, making the model more efficient in recognition of the characters.

Increases the robustness of dataset: Having augmented data points make model more robust as it is less sensitive to small variations in the dataset. By allowing the model to learn from more different variations of the characters, the model can identify the variations(rotation, shearing) of these characters.

Prevents Overfitting: Overfitting occurs when the model is not able to generalize well to unseen data. Augmentation can prevent overfitting by providing more diversed dataset to learn from allowing for more feature detection, which it can later apply on unseen data for classification.

5.6. Model Implementation

Four different CNN architectures were tried out in our project: LeNet, AlexNet, VGG and ResNet. This was carried out to test the accuracy for character recognition in different networks, and to choose the best network based on accuracy, loss, precision and F1 score. The network with the best performance was deployed on our project.

5.6.1. VGG-16 architecture

For VGG-16, 17 layers were used to define the model. The model has 1,368,928 total parameters involved. The layers of the model are listed below, and depicted graphically in 5.5.

- Input layer-1
- Convolutional layers-6
- Max Pooling layers-3
- Dropout layers-4
- Flatten layer-1
- Dense layers-2

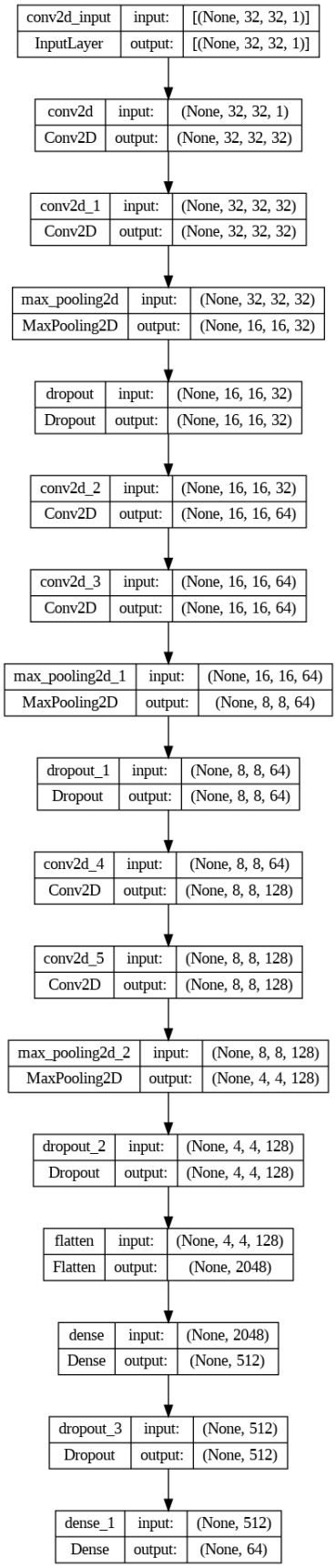


Figure 5.5: VGG-16 layers implementation

- **First Conv2D Layer:** This layer takes input of (32,32,1) and it applies 32 filters of kernel size 3*3 on the received input. After the convolution operation the resulting output is of size 32*32 with 32 channels. Activation function used in this case is ReLU

- **Second Conv2D:** Assuming that the input to this layer is the output of the first Conv2D layer, which has a shape of (32, 32, 32), the second Conv2D layer applies 32 filters of size (3,3) to this input feature map, with a stride of 1 and the same padding. Again, the filters are initialized randomly and learned during training through backpropagation.

- **MaxPooling 2D:** The MaxPooling2D layer applies max pooling to the input feature map with a pool size and stride of (2,2) and a shape of (32, 32, 32).

- **Dropout Layer:** In our specific case, the Dropout layer is applied after each MaxPooling2D layer with a rate of 0.25. The Dropout layer randomly drops out 25% of the neurons in the feature map avoiding overfitting.

- **Third Conv2D Layer:** The output of the preceding layer is subjected to 64 convolutional filters with a size of (3,3) in this layer. And 'same' padding makes sure that the output after convolution has same spatial dimension as the input. ReLU activation is applied element-wise to the output feature maps. Input: 32 feature maps of size 16×16, Output: 64 feature maps of size 16×16.

- **Fourth Conv2d Layer:** The output of the preceding layer is subjected to 64 convolutional filters with a size of (3,3) in this layer. And 'same' padding makes sure that the output after convolution has same spatial dimension as the input. ReLU activation is applied element-wise to the output feature maps. Input: 64 feature maps of size 16×16, Output: 64 feature maps of size 16×16.

- **Second MaxPooling 2D:** The input feature map, with a dimension of (16, 16, 64), is partitioned into non-overlapping 2×2 subregions in the MaxPooling2D layer. The largest value inside each subregion is chosen, and the output is a downsampled feature map with half as many spatial dimensions (8, 8, 64) as the input. This layer downsamples the input by taking the maximum value in each 2×2 region of the feature maps. Input: 64 feature maps of size 16×16, Output: 64 feature maps of size 8×8.

- **DropOut Layer:** This layer drops 25% of the neurons in the training process to avoid overfitting.

- **Fifth Conv2D:** The input to this layer is the output feature maps from the previous Conv2D layer, which would have a shape of (32, 32, 32, 64). Each of the 128

filters would have a size of (3,3) and would be convoluted with the input feature maps to produce 128 output feature maps. Finally, ReLU activation is applied element-wise to these output feature maps, resulting in a rectified version of the output feature maps.

- **Sixth Conv2D:** This layer applies another set of 128 convolutional filters of size (3,3) to the output of the previous layer. The 'same' padding ensures that the output feature maps have the same spatial dimensions as the input. ReLU activation is applied element-wise to the output feature maps.Output being of shape(batch size, 8, 8, 128).

- **Third MaxPooling 2D:** The MaxPooling2D layer down-samples the input feature maps' spatial dimensions by picking the highest value found in each pooling zone. The input feature maps are subjected to a max pooling procedure having pool size (2,2) and stride (2,2) in this particular layer. This layer's output is (batch size, 4, 4, 128).

- **Flatten Layer:** The input tensor is transformed into a 1-dimensional array using Keras' Flatten layer. It takes the output tensor from the previous layer as input and returns a flattened 1D tensor as output. The input shape of (batch size, 4, 4, 128) is converted to (batch size, 2048).

- **Dense Layer:** Dense layer with 512 units and ReLU activation applies a matrix multiplication between the input tensor and a weight matrix of shape (2048, 512). The output of this layer is then the result of the matrix multiplication added to a bias vector of shape (512,) and the ReLU function applied element-wise to this sum. The output shape of this layer is (None, 512).

- **Output Layer:** The Dense layer with 64 units and softmax activation takes as input a flattened feature vector of shape (None, 512) (i.e., a 1D array of length 512 for each input sample), which is the output of the previous Dense layer. The weight matrix has a shape of (512, 64) to allow the layer to learn 64 different output classes. The bias vector has a shape of (64,), with one bias term for each output class. The output of the layer will have a shape of (None, 64),

5.6.2. LeNet Architecture

For LeNet, 11 layers were used to define the model. The model has 484,864 total parameters involved. The layers of the model are listed below, and depicted graphically as:

- Input layer-1
- Convolutional layers-2
- Max Pooling layers-2
- Dropout layers-3
- Flatten layer-1
- Dense layers-2

- **Conv2D Layer:**This layer takes an input of (32,32,1) applies 128 filters having a size of (3,3) with strides 1 and ReLu activation function. This layer provides output of shape(32,32,128).
- **MaxPool2D:**This layer performs max pooling operation on the output of first conv2D Layer. The pool size is (2,2), stride also being (2,2). The output of shape is (16,16,128).
- **Dropout Layers:**It randomly drops 25% of the neurons to avoid overfitting.
- **Conv2D Layer:**It applies 64 filters having a size of (3,3) with strides 1 and ReLu activation function. This layer provides output of shape(16,16,64).
- **MaxPool2D:**This layer performs max pooling operation on the output of second conv2D Layer. The pool size is (2,2), stride also being (2,2). The output of shape is (8,8,64).
- **Flatten Layer:**It flattens the 2D array to 1D array.
- **Dense Layer**: Fully connected Layer having 128 neurons, ReLu activation giving output of (128,)
- **Dropout Layers:**It randomly drops 25% of the neurons to avoid overfitting.
- **Dense Layer**: Fully connected Layer having 64 neurons, Softmax activation giving output probabilities.

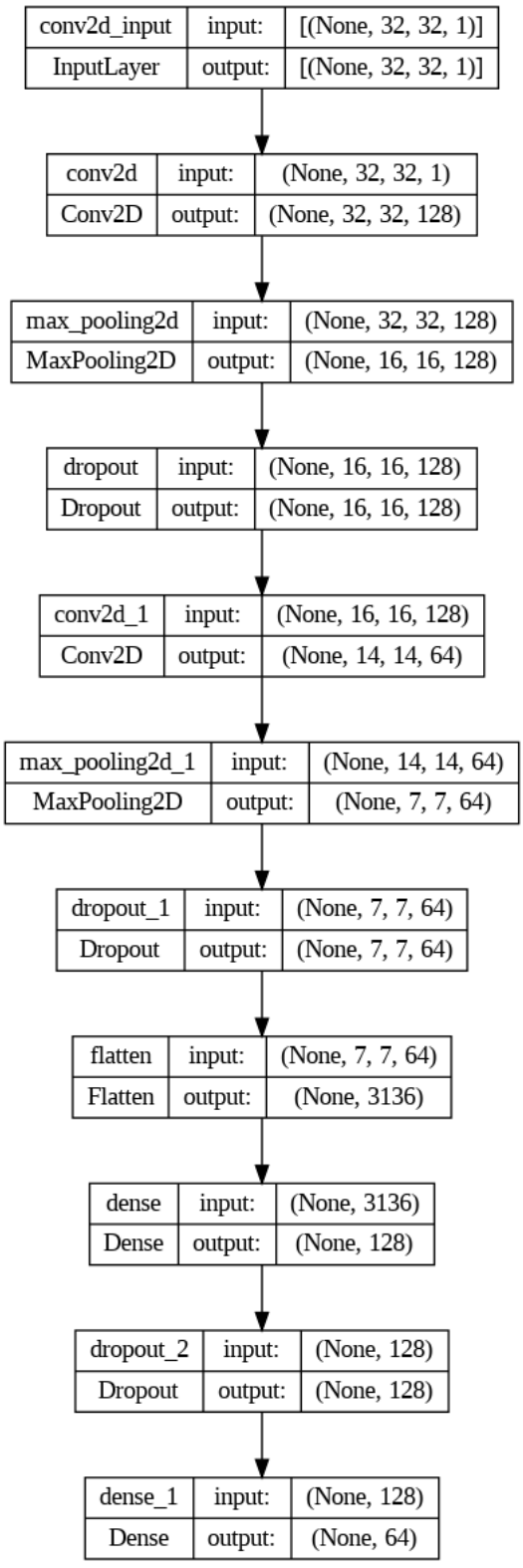


Figure 5.6: LeNet layers implementation

5.6.3. AlexNet architecture

For AlexNet, 16 layers were used to define the model. The model has 1,483,872 total parameters involved. The layers of the model are listed below, and depicted graphically in 5.7.

- Input layer-1
- Convolutional layers-6
- Max Pooling layers-3
- Dropout layers-2
- Flatten layer-1
- Dense layers-3

- **Conv2D Layer:**This layer takes an input of (32,32,1) applies 32 filters having a size of (3,3) with strides 1 and ReLu activation function. This layer provides output of shape(32,32,32).
- **Conv2D Layer:**This layer takes an input of (32,32,32) applies 32 filters having a size of (3,3) and activation function as Relu. This layer provides output of shape(32,32,32).
- **MaxPool2D:**This layer performs max pooling operation on the output of first conv2D Layer. The pool size is (2,2), stride alsi being (2,2). The output of shape id (16,16,32)
- **Conv2D Layer:**This layer applies 64 filters with a size of (3,3) and a ReLu activation function to an input of (16,16,32). This layer outputs a shape of (16,16,64)..
- **Conv2D Layer:**It applies 64 filters having a size of (3,3) with strides 1 and ReLu activation function. This layer provides output of shape(16,16,64).
- **MaxPool2D:**This layer performs max pooling operation on the output of second conv2D Layer. The pool size is (2,2), stride alsi being (2,2). The output of shape id (8,8,64)
- **Conv2D Layer:**It utilizes ReLu activation and 128 filters with a size of (3,3). This layer outputs a shape of (8,8,128).

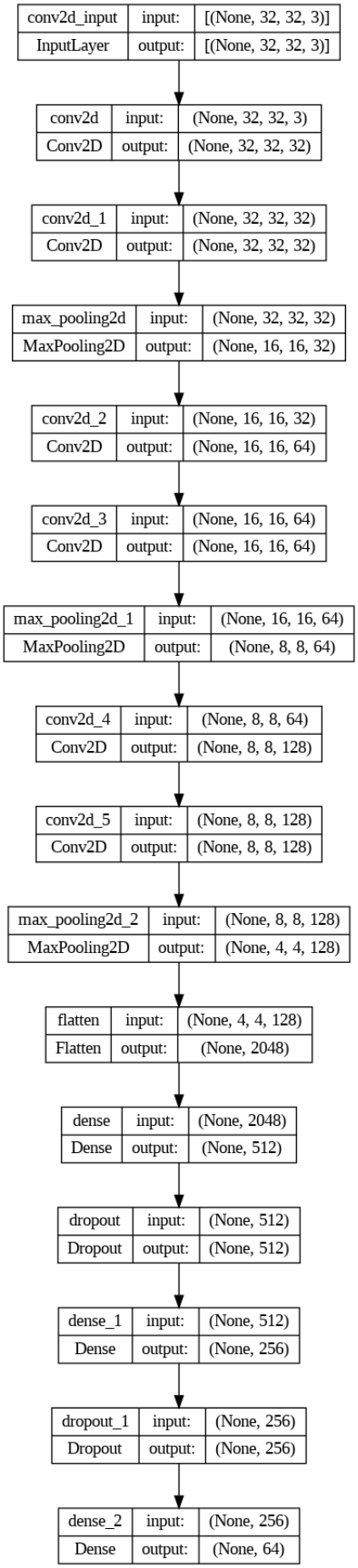


Figure 5.7: AlexNet layers implementation

- **Conv2D Layer:**It applies 128 filters having a size of (3,3) and ReLu activation function. This layer provides output of shape(8,8,128)
- **MaxPool2D:**This layer performs max pooling operation on the output of second conv2D Layer. The pool size is (2,2), stride alsi being (2,2). The output of shape is (4,4,128)
- **Flatten Layer:**It flattens the 2D array to 1D array.
- **Dense Layer**: Fully connected Layer having 128 neurons, ReLu activation giving output of (512,)
- **Dropout Layers:**It randomly drops 25% of the neurons to avoid overfitting.
- **Dense Layer**: Fully connected Layer having 256 neurons, relu activation
- **Dropout Layers:**It randomly drops 25% of the neurons to avoid overfitting
- **Dense Layer**: Fully connected Layer having 64 neurons, softmax activation function to assign probabilities to each target class

5.6.4. ResNet architecture

For ResNet, 15 layers were used to define the model. The model has 9,390,784 total parameters involved. The layers of the model are listed below, and depicted graphically as:

- Input layer-1
- Convolutional layers-6
- Max Pooling layers-2
- Dropout layers-2
- Flatten layer-1
- Dense layer-1

- **Conv2D Layer:**This layer applies 64 filters with a size of (3,3) to an input of (32,32,1) using strides 1 and the ReLu activation function. Shape(32,32,64) is output from this layer.

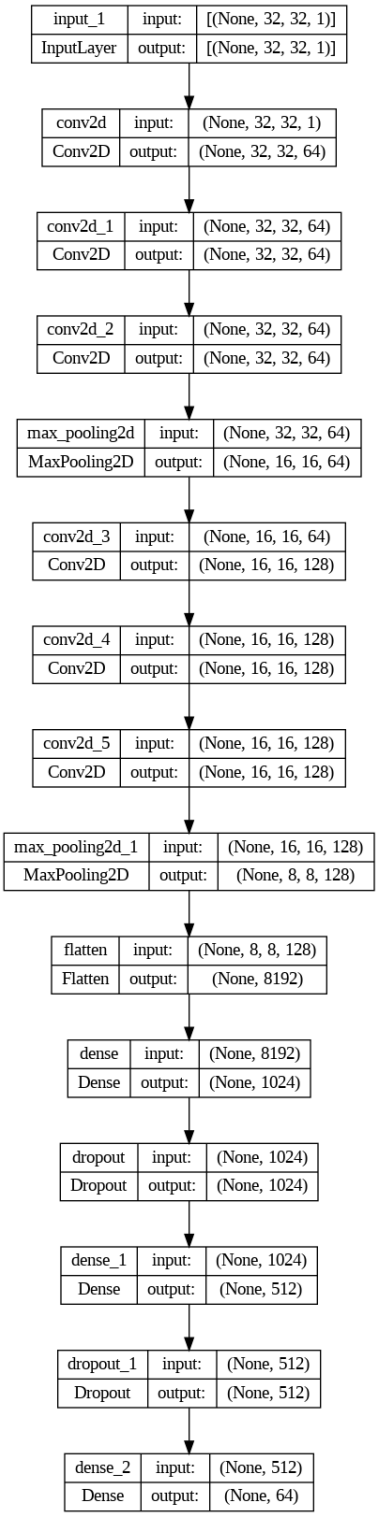


Figure 5.8: ResNet-17 layers implementation

- **Conv2D Layer:**This layer applies 32 filters with a size of (3,3) and a ReLu activation function on an input of (32,32,32). Shape(32,32,64) is output from this layer.
- **Conv2D Layer:**This layer takes an input of (32,32,32) applies 32 filters having a size of (3,3) and ReLu activation function. This layer provides output of shape(32,32,64).
- **MaxPool2D:**This layer performs max pooling operation on the output of first conv2D Layer. The pool size is (2,2), stride alsi being (2,2). The output of shape id (16,16,64)
- **Conv2D Layer:**This layer takes an input of (16,16,64) applies 128 filters having a size of (3,3) and ReLu activation function. This layer provides output of shape(16,16,128).
- **Conv2D Layer:**It applies 64 filters having a size of (3,3) with strides 1 and ReLu activation function. This layer provides output of shape(16,16,128).
- **Conv2D Layer:**It applies 64 filters having a size of (3,3) with strides 1 and ReLu activation function. This layer provides output of shape(16,16,128).
- **MaxPool2D:**This layer performs max pooling operation on the output of second conv2D Layer. The pool size is (2,2), stride also being (2,2). The output of shape id (8,8,128)
- **Flatten Layer:**It flattens the 2D array to 1D array.
- **Dense Layer**: Fully connected Layer having 1024 neurons, ReLu activation giving output of (512,)
- **Dropout Layers:**It randomly drops 25% of the neurons to avoid overfitting.
- **Dense Layer**: Fully connected Layer having 512 neurons, relu activation
- **Dropout Layers:**It randomly drops 50% of the neurons to avoid overfitting
- **Dense Layer**: Fully connected Layer having 64 neurons, softmax activation function to assign probabilities to each target class

5.7. Web-API Interaction

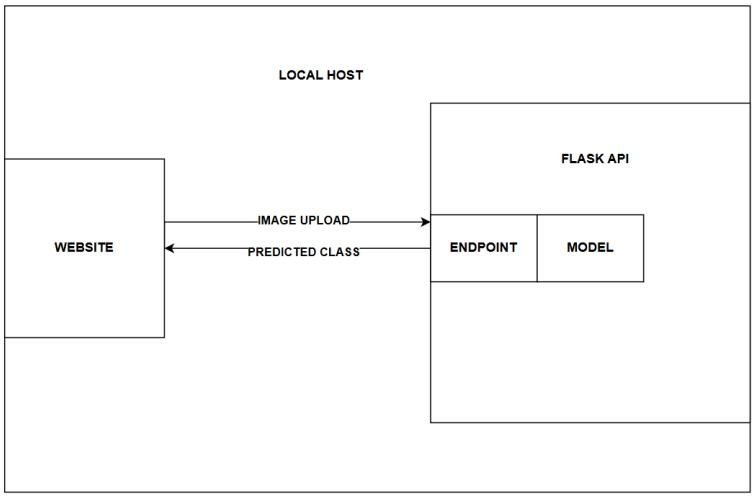


Figure 5.9: System Interaction Diagram

The user uploads an image of the character they want to recognize using the web app created using HTML, CSS, and JavaScript. The image file is sent to the Flask API using an HTTP POST request to the API endpoint. The Flask API preprocesses the image by fixing the resolution, grayscaling, and resizing it to make it compatible with the input layer of the trained VGG CNN model. The Flask API then calls the predict method on the preprocessed image using the trained model to get the predicted class of the uploaded image. The predicted class is returned in the body of a JSON response with key-value pairs. The value of the predicted class is obtained in the frontend and displayed to the user in the Flask web app. Both the Website and Flask API are run on local machine on same port facilitating the interaction among them

Overall, the Flask API app acts as a middleware between the frontend and the backend, facilitating communication and enabling the recognition of characters uploaded by the user.

6. RESULTS AND ANALYSIS

A total of 64 characters, has been taken for the dataset creation: 38 consonants with some combined characters, 15 vowels and 10 numerals. Since there was no dataset available for the Prachalit script, dataset was built from scratch, through collection from various sources, such as manuscripts and hand-written samples. Eventually, a total of 51200 images, of 64 classes, containing 800 images in each class, were trained on 4 different CNN Architectures. The results of different CNN architectures were analyzed and the best architecture was chosen eventually.

6.1. Results

Five different CNN models were trained in Google Colab notebook. The subsequent results obtained were as follows.

Table 6.1: Observations using Different CNN models

Model	**Epoch**	**Early Stopped**	**Batch size**	**Precision**	**Recall**	**F-1**
AlexNet	50	22	32	0.91	0.91	0.91
LeNet	50	50	32	0.86	0.86	0.86
ResNet	50	17	32	0.92	0.92	0.91
VGG	50	34	32	0.96	0.96	0.96

Table 6.2: Observations of Accuracy using Different CNN models

Model	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**	**Testing Loss**	**Testing Accuracy**
AlexNet	0.11	0.96	0.29	0.93	0.30	0.92
LeNet	0.44	0.86	0.49	0.88	0.49	0.88
ResNet	0.11	0.96	0.31	0.93	0.26	0.93
VGG	0.20	0.93	0.18	0.95	0.15	0.95

This table compares four different convolutional neural network models based on their performance on an image classification task. Precision, Recall, F-1 score, Accuracy and Loss metrics have been used.

6.1.1. Alexnet

Classification Report:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	84
1	0.86	0.93	0.90	92
2	0.79	0.87	0.82	114
3	0.95	0.81	0.87	94
4	0.86	0.95	0.90	88
5	0.87	0.93	0.90	84
6	0.93	0.94	0.93	96
7	0.85	0.99	0.91	95
8	0.96	0.84	0.90	112
9	0.94	0.92	0.93	106
10	0.94	0.98	0.96	97
11	0.92	0.96	0.94	113
12	0.84	0.95	0.89	91
13	0.92	0.84	0.88	108
14	0.92	0.95	0.94	88
15	0.99	0.91	0.95	100
16	0.94	0.89	0.91	127
17	0.94	0.93	0.93	112
18	0.97	0.86	0.91	102
19	0.93	0.97	0.95	107
20	0.97	0.91	0.94	110
21	0.94	0.94	0.94	93
22	0.95	0.92	0.93	95
23	0.99	0.89	0.94	95
24	0.97	0.98	0.97	92
25	0.90	0.94	0.92	101
26	0.91	0.97	0.94	109
27	0.99	0.96	0.98	107
28	0.99	0.97	0.98	101
29	0.87	0.88	0.88	86
30	0.94	0.90	0.92	106
31	0.76	0.86	0.81	100
32	0.72	0.86	0.78	95
33	0.97	0.96	0.96	113
34	0.92	0.88	0.90	93
35	0.95	0.96	0.95	97
36	0.83	0.95	0.88	96
37	0.93	0.97	0.95	98
38	0.94	0.93	0.93	100
39	0.94	0.86	0.90	108
40	0.91	0.97	0.94	105
41	0.90	0.91	0.90	95
42	0.91	0.87	0.89	108
43	0.95	0.88	0.92	111
44	0.98	0.81	0.89	100
45	0.91	0.84	0.88	89
46	0.94	0.95	0.95	101
47	0.87	0.95	0.91	92
48	0.86	0.87	0.86	102
49	0.93	0.91	0.92	103
50	0.90	0.89	0.89	100
51	0.91	0.79	0.85	112
52	0.89	0.84	0.87	107
53	0.87	0.92	0.89	95
54	0.95	0.90	0.93	114
55	0.91	0.93	0.92	97
56	0.91	0.87	0.89	97
57	0.95	0.93	0.94	97
58	0.83	0.84	0.84	107
59	0.90	0.88	0.89	97
60	0.93	0.96	0.95	100
61	0.88	0.90	0.89	99
62	0.79	0.91	0.85	80
63	0.91	0.87	0.89	94
accuracy			0.91	6407
macro avg	0.91	0.91	0.91	6407
weighted avg	0.91	0.91	0.91	6407

Figure 6.1: AlexNet Classification report

The AlexNet model has precision, recall and F_1 score of 0.93 each individually indicate that the model is able to classify images with a higher level of confidence. Higher recall number demonstrates the classifier's ability to accurately identify more positive cases, while a higher precision value shows that the classifier is generating fewer incorrect positive predictions. A classifier that performs well in accuracy and recall has a higher F1 score.

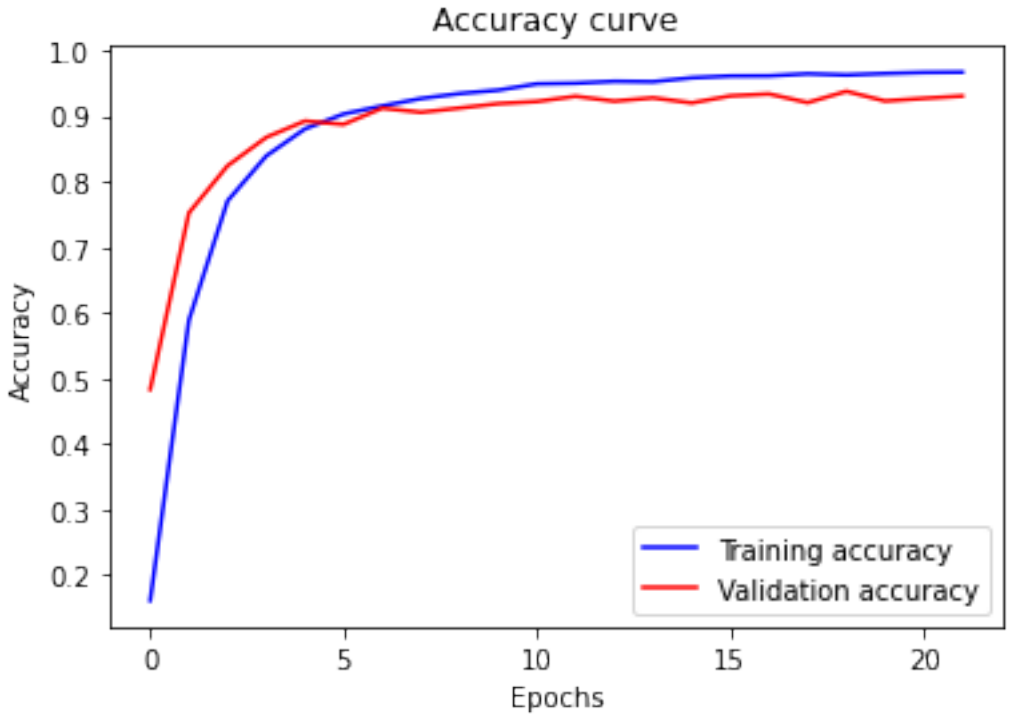


Figure 6.2: AlexNet accuracy Curve

The training accuracy is 0.96 and validation accuracy is 0.93 shows that the model is being able to accurately identify the target variables on both the training and validation data. The testing accuracy of 0.92 is also high which also shows that the model is able to recognize unseen data correctly. The small gap between the validation accuracy and training accuracy suggests that the model is not being overfit to the training data and is being able to generalize well to unseen data.

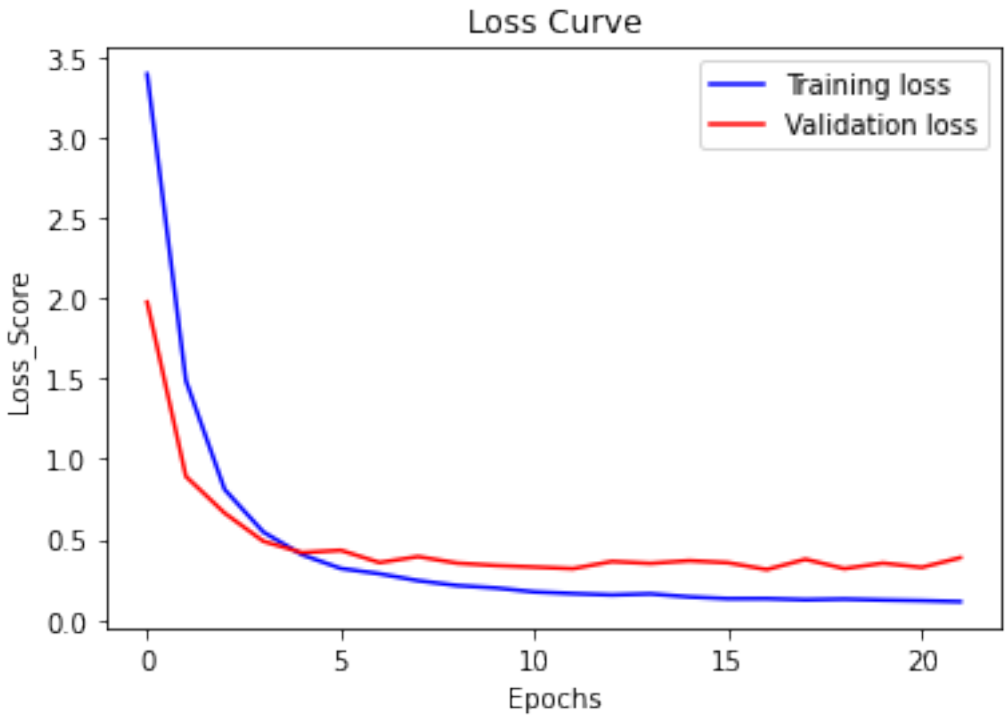


Figure 6.3: AlexNet loss Curve

The training loss is 0.11 indicate that the model is able to classify data with an lower value of error. The validation loss of 0.29 also shows that model is performing well on the validation data-set. The relatively lower value of loss also shows that the model is avoiding overfitting, The training loss of 0.30 also suggest that the model is working well on unseen data with lower error.

6.1.2. Lenet

Classification Report:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	97
1	0.88	0.82	0.85	93
2	0.83	0.75	0.78	102
3	0.85	0.80	0.82	96
4	0.71	0.95	0.81	99
5	0.89	0.83	0.86	102
6	0.88	0.90	0.89	87
7	0.96	0.84	0.90	125
8	0.88	0.92	0.90	106
9	0.85	0.79	0.82	87
10	0.78	1.00	0.88	86
11	0.90	0.84	0.87	107
12	0.89	0.92	0.90	95
13	0.85	0.87	0.86	92
14	0.91	0.94	0.92	98
15	0.95	0.89	0.92	89
16	0.86	0.87	0.86	105
17	0.80	0.91	0.85	86
18	0.68	0.91	0.78	110
19	0.86	0.93	0.89	86
20	0.88	0.79	0.83	103
21	0.87	0.86	0.86	107
22	0.86	0.89	0.88	93
23	0.91	0.92	0.92	112
24	0.91	0.86	0.88	112
25	0.92	0.74	0.82	94
26	0.87	0.87	0.87	92
27	0.88	0.94	0.91	104
28	0.83	0.95	0.89	101
29	0.88	0.87	0.87	106
30	0.82	0.89	0.85	104
31	0.76	0.69	0.73	94
32	0.74	0.75	0.74	119
33	0.93	0.88	0.90	113
34	0.90	0.85	0.87	101
35	0.84	0.93	0.89	105
36	0.92	0.83	0.87	108
37	0.97	0.90	0.94	83
38	0.81	0.83	0.82	92
39	0.84	0.86	0.85	104
40	0.93	0.85	0.89	93
41	0.82	0.86	0.84	102
42	0.83	0.87	0.85	100
43	0.85	0.92	0.88	114
44	0.86	0.81	0.83	77
45	0.93	0.81	0.87	107
46	0.81	0.81	0.81	97
47	0.88	0.90	0.89	109
48	0.75	0.80	0.77	85
49	0.91	0.86	0.88	85
50	0.93	0.86	0.89	93
51	0.76	0.79	0.78	114
52	0.95	0.73	0.82	106
53	0.87	0.74	0.80	98
54	0.88	0.75	0.81	101
55	0.91	0.91	0.91	109
56	0.86	0.86	0.86	98
57	0.85	0.93	0.89	88
58	0.72	0.81	0.76	104
59	0.93	0.87	0.90	111
60	0.87	0.86	0.86	97
61	0.83	0.84	0.84	108
62	0.81	0.83	0.82	103
63	0.92	0.91	0.92	113
accuracy			0.86	6407
macro avg	0.86	0.86	0.86	6407
weighted avg	0.86	0.86	0.86	6407

Figure 6.4: LeNet Classification Report

The Lenet model has precision, recall and F_1 score of 0.86 each individually indicates that the model is able to classify images with a moderate level of accuracy. A moderate level of precision of 86% shows that the out of all the instances that were actually positive, the model correctly identified 86% of them. A recall score of 0.86 indicates that, of all the positive events, 86% of them were correctly detected by the model.While in multiple class classification, the classifier is doing a moderate task of identifying the correct classes and reducing false positives.

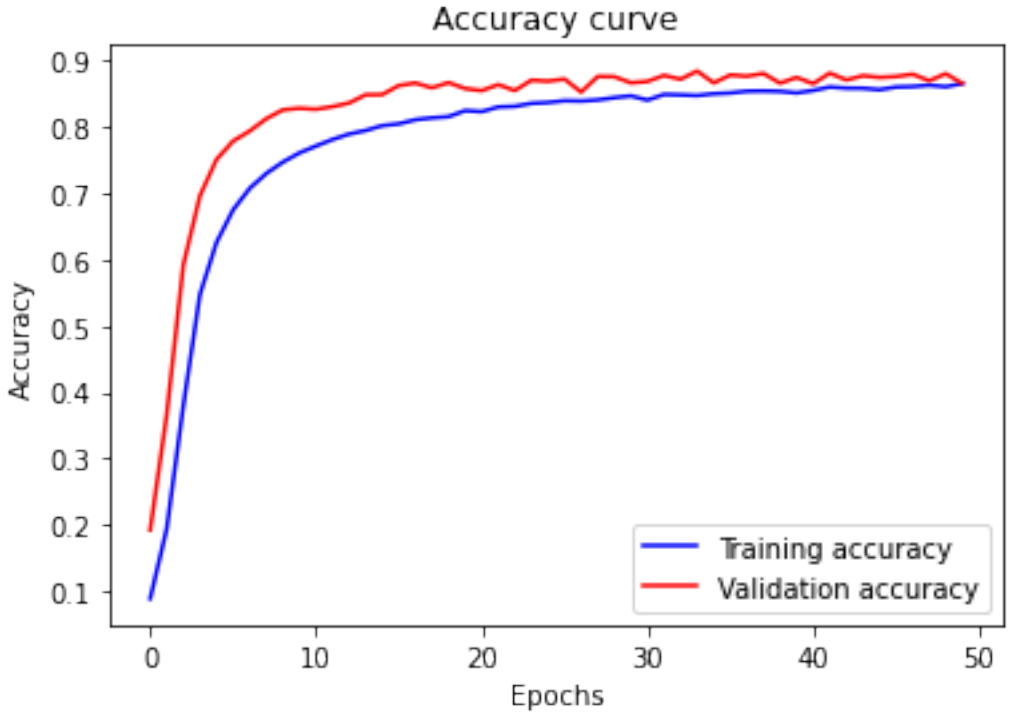


Figure 6.5: LeNet accuracy

The training accuracy is 0.86 , validation accuracy is 0.88 shows that the model is being able to classify target variable on both the training and validation data-set. The higher value of training accuracy suggests that the model is being able to recognize unseen data but the accuracy level was found to be the lowest among the 5 models tried.

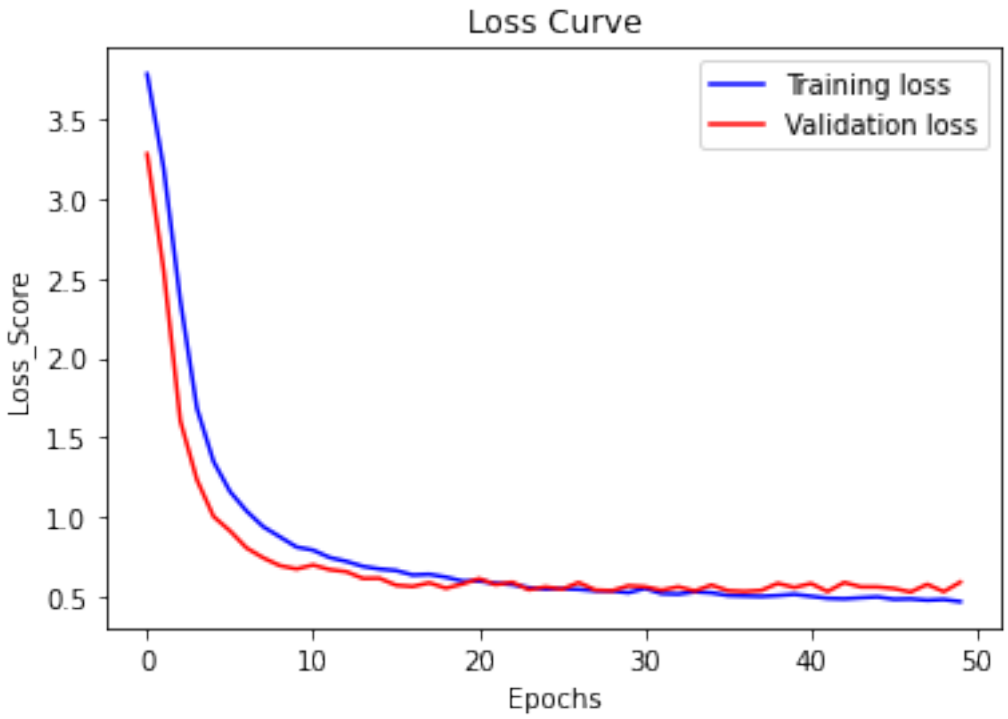


Figure 6.6: LeNet loss

The model cannot effectively classify, as seen by the training loss of 0.44, validation loss of 0.49, and testing accuracy of 0.49.The model cannot be used since the larger value of loss suggests that it is unable to complete the classification objective. The training loss is less than both the validation loss and the testing loss, which indicates that the model is over fitting to the training data.

6.1.3. ResNet

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	94
1	0.95	0.90	0.92	100
2	0.92	0.93	0.92	104
3	0.96	0.90	0.93	99
4	0.95	0.92	0.93	98
5	0.93	0.84	0.88	90
6	0.85	0.97	0.90	92
7	0.84	0.96	0.90	96
8	0.92	0.97	0.94	90
9	0.99	0.92	0.95	90
10	0.95	1.00	0.98	82
11	0.94	0.87	0.90	98
12	0.94	0.96	0.95	108
13	0.95	0.92	0.94	102
14	0.95	0.93	0.94	90
15	0.96	0.92	0.94	96
16	0.97	0.87	0.91	98
17	0.93	0.87	0.90	108
18	0.88	0.96	0.92	93
19	0.97	0.98	0.98	103
20	0.97	0.91	0.94	104
21	0.96	0.93	0.94	107
22	0.93	0.91	0.92	87
23	0.98	0.93	0.95	94
24	0.92	0.94	0.93	82
25	0.91	0.94	0.93	109
26	0.97	1.00	0.98	94
27	0.97	0.99	0.98	89
28	0.99	0.97	0.98	108
29	0.92	0.95	0.94	116
30	0.92	0.92	0.92	100
31	0.88	0.77	0.82	105
32	0.85	0.80	0.83	97
33	0.96	0.97	0.96	116
34	0.82	0.86	0.84	100
35	0.89	0.96	0.92	94
36	0.91	0.89	0.90	82
37	0.93	1.00	0.96	89
38	0.93	0.90	0.92	103
39	0.67	0.89	0.77	84
40	0.85	0.94	0.89	99
41	0.83	0.94	0.88	96
42	0.87	0.92	0.90	102
43	0.93	0.97	0.95	97
44	0.93	0.87	0.90	121
45	0.95	0.93	0.94	94
46	0.92	0.94	0.93	106
47	0.84	0.92	0.88	106
48	0.95	0.87	0.91	109
49	0.93	0.91	0.92	103
50	0.89	0.93	0.91	115
51	0.92	0.78	0.84	113
52	0.89	0.86	0.88	114
53	0.92	0.91	0.92	107
54	0.89	0.87	0.88	92
55	0.95	0.86	0.90	107
56	0.84	0.92	0.88	97
57	0.97	0.92	0.95	103
58	0.88	0.83	0.85	120
59	0.89	0.95	0.92	98
60	0.96	0.92	0.94	111
61	0.84	0.90	0.87	103
62	0.96	0.87	0.91	109
63	0.88	0.89	0.89	94
accuracy			0.91	6407
macro avg	0.92	0.92	0.92	6407
weighted avg	0.92	0.91	0.91	6407

Figure 6.7: ResNet Classification Report

The ResNet model has precision, recall and F_1 score of 0.92 , 0.91 and 0.91 respectively indicate that the model is able to classify images with a higher level of confidence. A greater recall number demonstrates the classifier's ability to accurately identify more positive cases, while a higher precision value shows that the classifier is generating fewer incorrect positive predictions. A classifier that performs well in accuracy and recall has a higher F1 score.

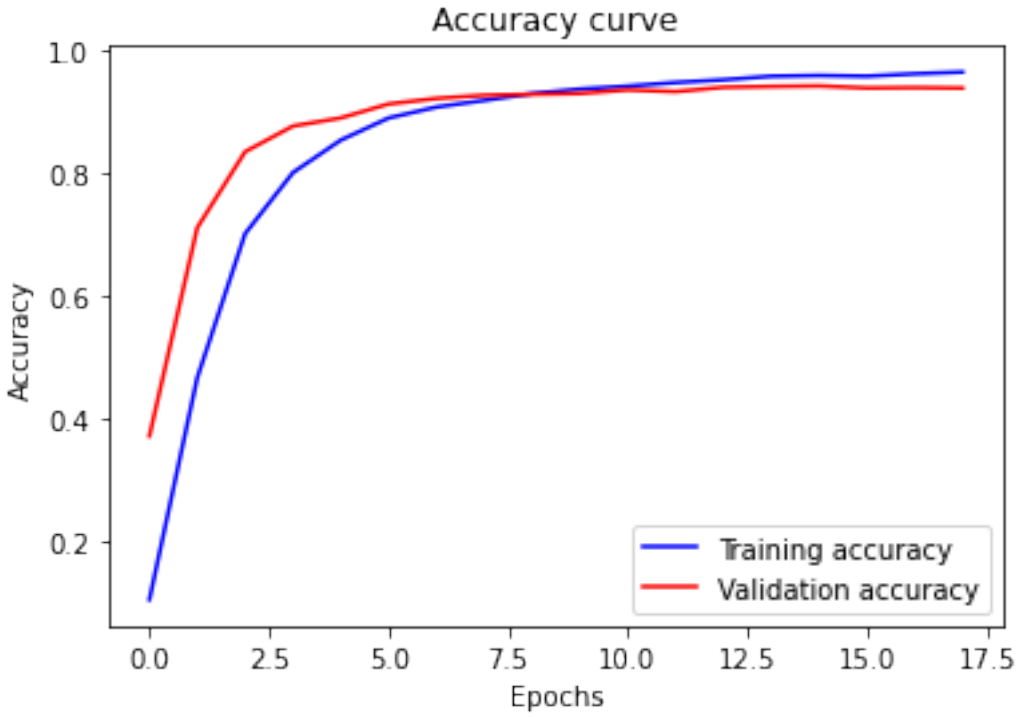


Figure 6.8: ResNet Accuracy Curve

The training accuracy is 0.96, validation accuracy is 0.93 and training accuracy is 0.93. The small gap between the training and validation accuracy also shows that the model is not overfit to the training data. But, the model obtained a lower level of training accuracy then the VGG Model.

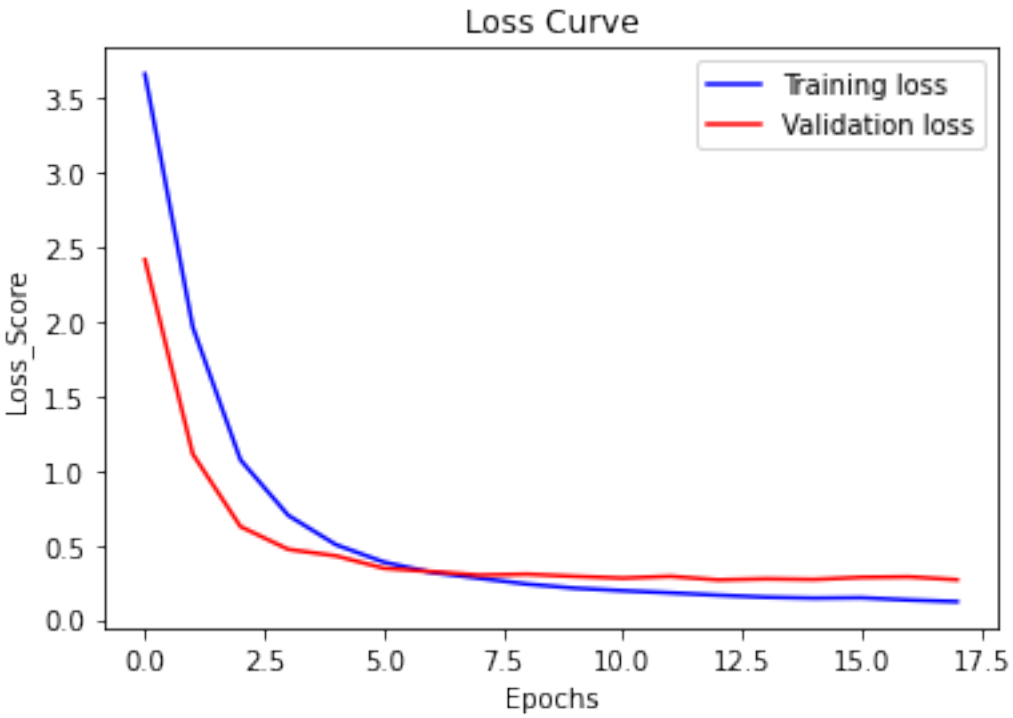


Figure 6.9: ResNet Accuracy Curve

The training loss is 0.11, validation loss is 0.31, and training loss is 0.26. The lower value of training loss shows that the model is being able to learn and adjust its weight on the training data well. The smaller value of validation loss also shows that the model is being able to generalize well to validation data.The not over fitting of the model is also suggested by the lower value of loss in training phase.

6.1.4. VGG

Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	170
1	0.91	0.95	0.93	167
2	0.93	0.96	0.94	158
3	0.97	0.94	0.96	148
4	0.99	0.94	0.96	160
5	0.95	0.98	0.96	166
6	0.94	0.97	0.96	155
7	0.97	0.99	0.98	144
8	0.97	0.96	0.97	167
9	0.99	0.98	0.98	167
10	0.97	0.99	0.98	159
11	0.98	0.93	0.96	176
12	0.98	0.97	0.98	159
13	0.98	0.98	0.98	157
14	0.98	1.00	0.99	168
15	1.00	0.97	0.99	155
16	0.94	0.96	0.95	178
17	0.92	0.96	0.94	160
18	0.98	0.94	0.96	175
19	0.93	0.99	0.96	142
20	0.97	0.99	0.98	173
21	1.00	0.98	0.99	153
22	0.95	0.96	0.95	147
23	0.99	0.95	0.97	179
24	0.97	0.98	0.98	183
25	0.96	0.97	0.97	152
26	0.98	0.98	0.98	174
27	0.99	0.97	0.98	152
28	1.00	1.00	1.00	164
29	0.96	0.96	0.96	165
30	0.97	0.96	0.97	158
31	0.88	0.90	0.89	157
32	0.88	0.86	0.87	170
33	0.98	0.97	0.97	155
34	0.92	0.94	0.93	162
35	0.97	1.00	0.98	152
36	0.94	0.99	0.96	146
37	0.99	0.99	0.99	169
38	0.99	0.93	0.96	163
39	0.92	0.91	0.92	159
40	0.98	0.96	0.97	163
41	0.89	0.93	0.91	153
42	0.96	0.97	0.96	179
43	0.99	0.98	0.98	149
44	0.91	0.96	0.93	142
45	0.99	0.97	0.98	165
46	0.99	0.95	0.97	150
47	0.97	0.93	0.95	150
48	0.95	0.95	0.95	151
49	0.98	0.94	0.96	166
50	0.96	0.97	0.97	153
51	0.96	0.93	0.95	169
52	0.85	0.98	0.91	162
53	0.96	0.92	0.94	160
54	0.96	0.94	0.95	171
55	0.95	0.97	0.96	147
56	0.93	0.94	0.93	157
57	0.99	0.95	0.97	169
58	0.92	0.89	0.90	141
59	0.99	0.98	0.98	139
60	0.95	0.99	0.97	152
61	0.96	0.90	0.93	155
62	0.93	0.90	0.91	175
63	0.90	0.92	0.91	158
accuracy			0.96	10240
macro avg	0.96	0.96	0.96	10240
weighted avg	0.96	0.96	0.96	10240

Figure 6.10: VGG-16 classification report

The VGG model has precision, recall and F_1 score of 0.96 , 0.96 and 0,96 respectively indicate that the model is able to classify images with a higher level of confidence. A greater recall number demonstrates the classifier's ability to accurately identify more positive cases, while a higher precision value shows that the classifier is generating fewer incorrect positive predictions. A classifier that performs well in accuracy and recall has a higher F1 score. On comparison, the precision, recall and f-1 score of the VGG model is found to be highest among the models tried showing that the model is able to identify more positive cases, and fewer incorrect positive predictions.

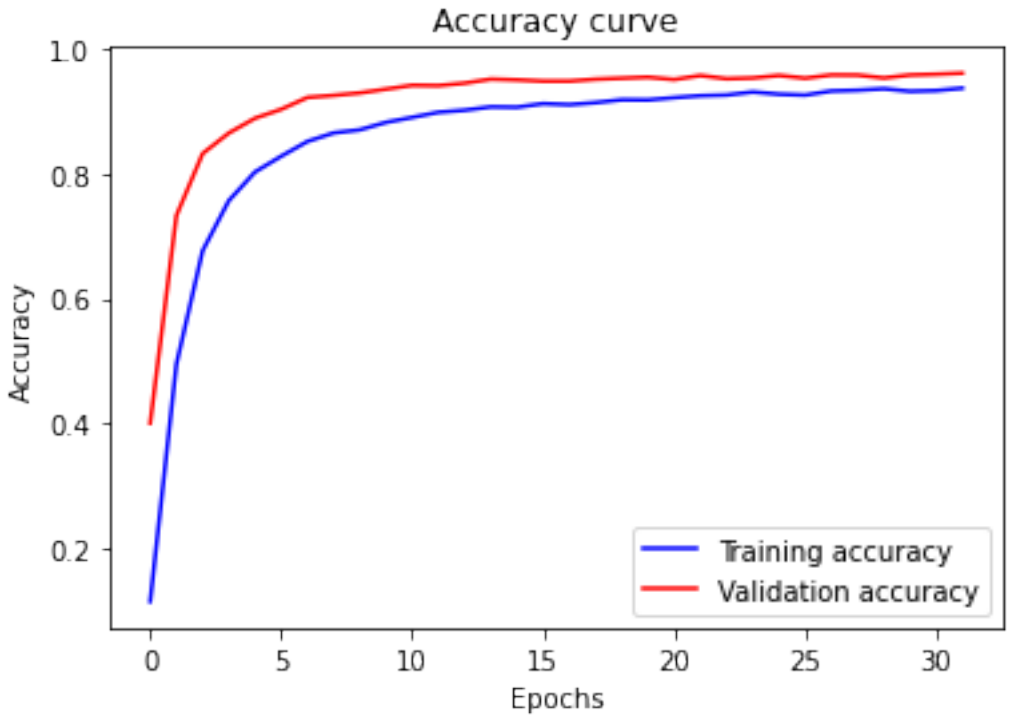


Figure 6.11: VGG-16 accuracy Curve

The training accuracy is 0.93, validation accuracy is 0.95 and training accuracy is 0.95. The small gap between the training and validation accuracy also shows that the model is not over fit to the training data. The higher value of validation and testing accuracy compared to training accuracy shows that the model is being able to recognize and generalize to the unseen data

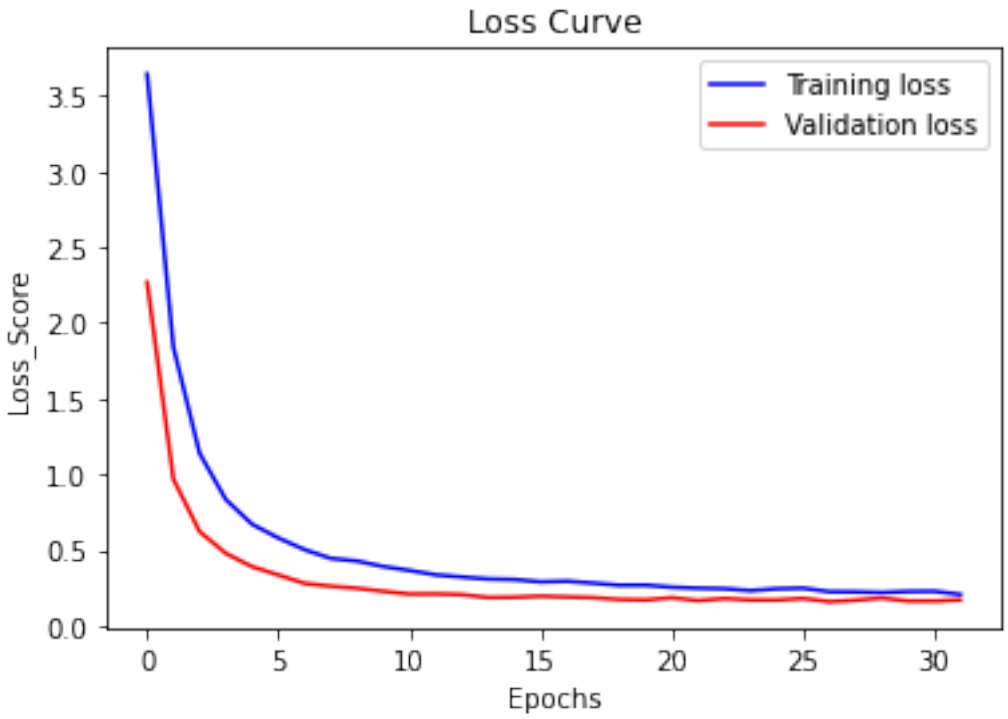


Figure 6.12: VGG-16 loss Curve

The training loss is 0.20, validation loss is 0.18 and testing loss is 0.15. The lower value of the loss value in the training data shows that the model is able to adjust its weights effectively. The validation loss of 0.18 is slightly less than the training loss indicating that the model is able to generalize well to data in validation set. The testing loss of 0.15 is slightly lower than the validation loss which shows that the model is performing well in unseen data.

Upon comparison it was found that the Lenet model was not able to work properly for image classification task as it has higher value of testing loss and lower value of accuracy on unseen data. Whereas, VGG and Resnet and Alexnet had better performance than the Lenet model. But, VGG model was selected for the reason for it having a better performance in the test(unseen) data set. The training loss is lowest among the models tried which indicates that the model is being able to generalize well on unseen data and is not overfitting which is also supported by the highest value of its testing accuracy.

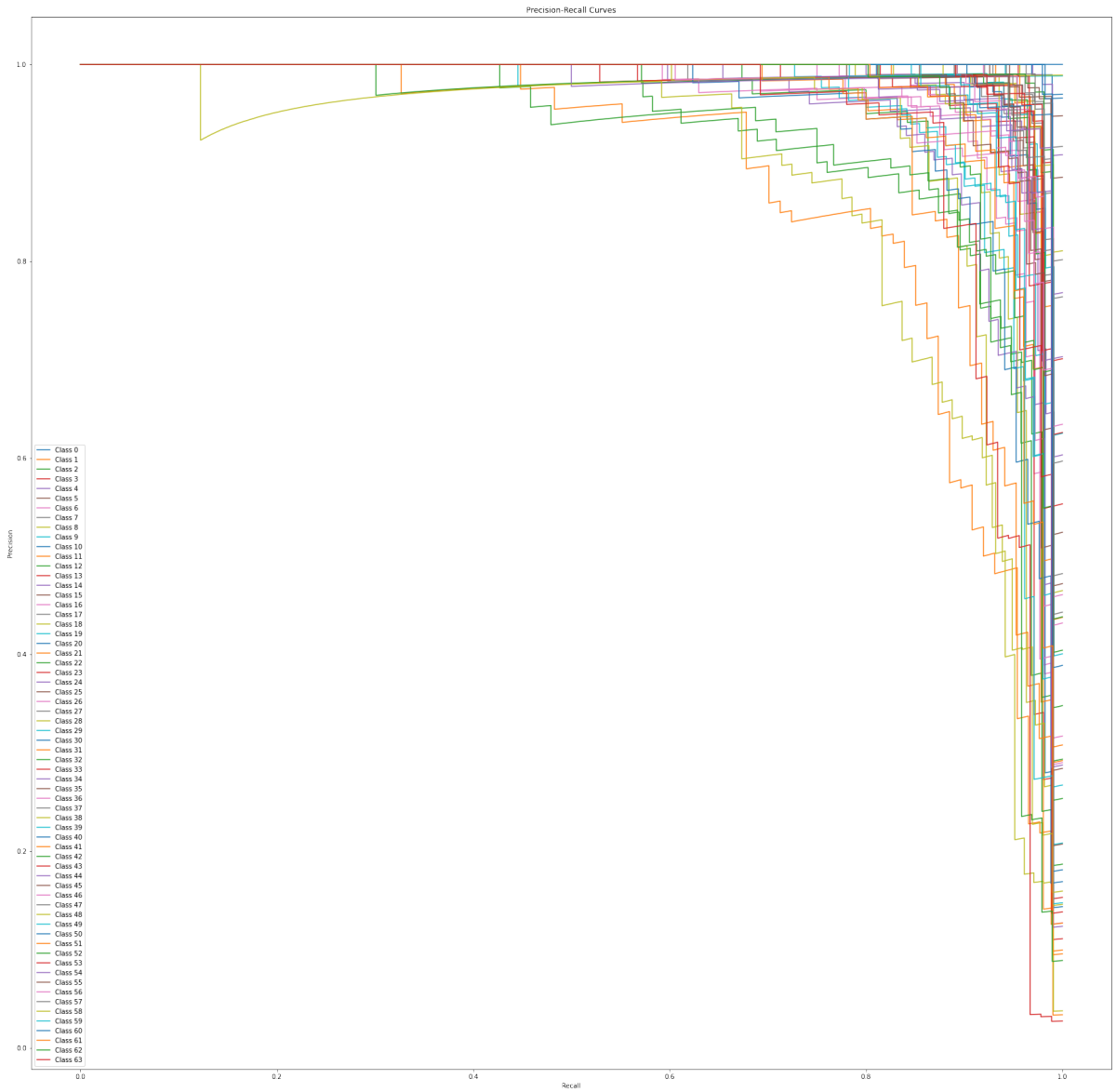


Figure 6.13: VGG-16 precision and recall

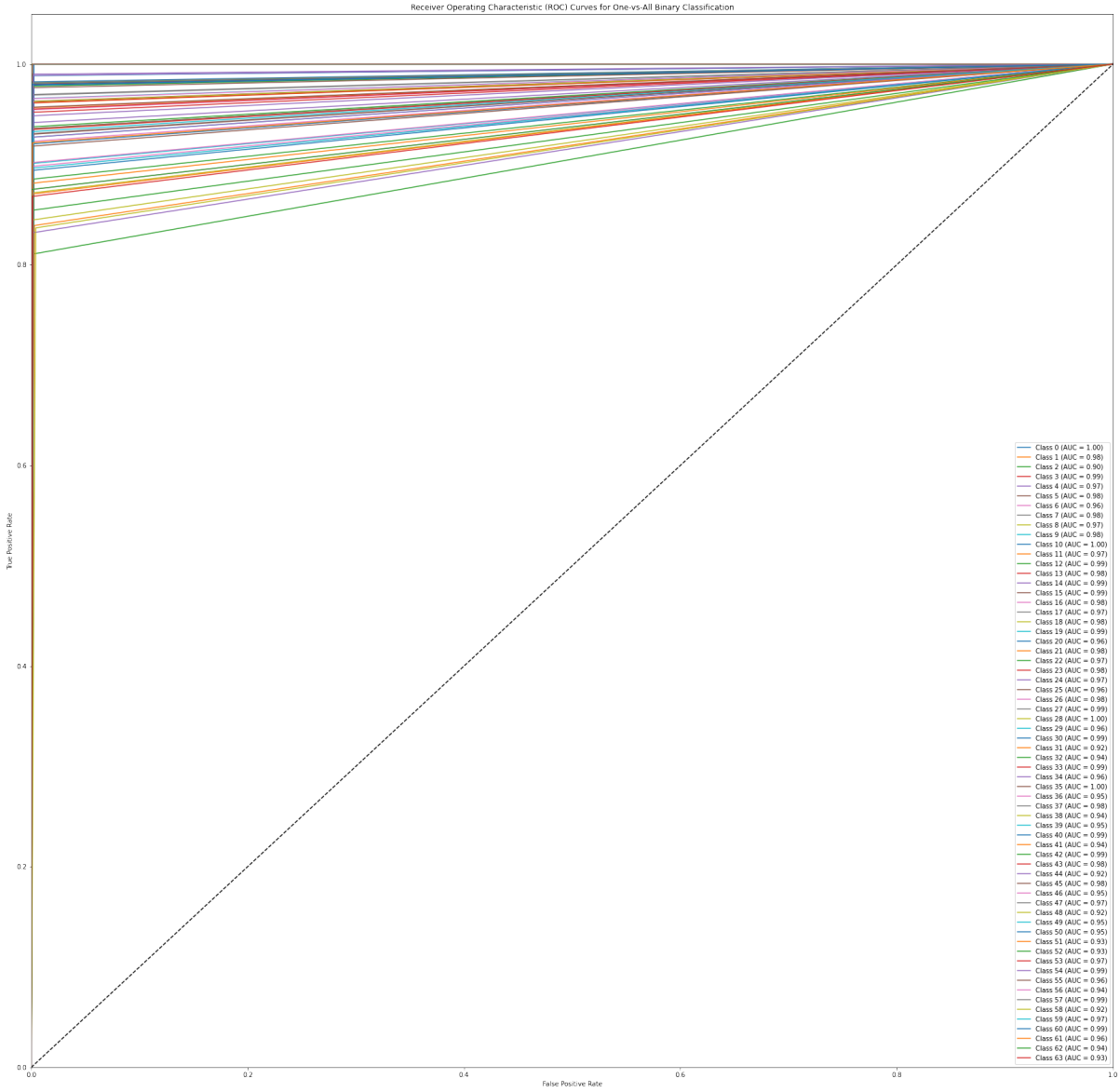


Figure 6.14: VGG-16 ROC

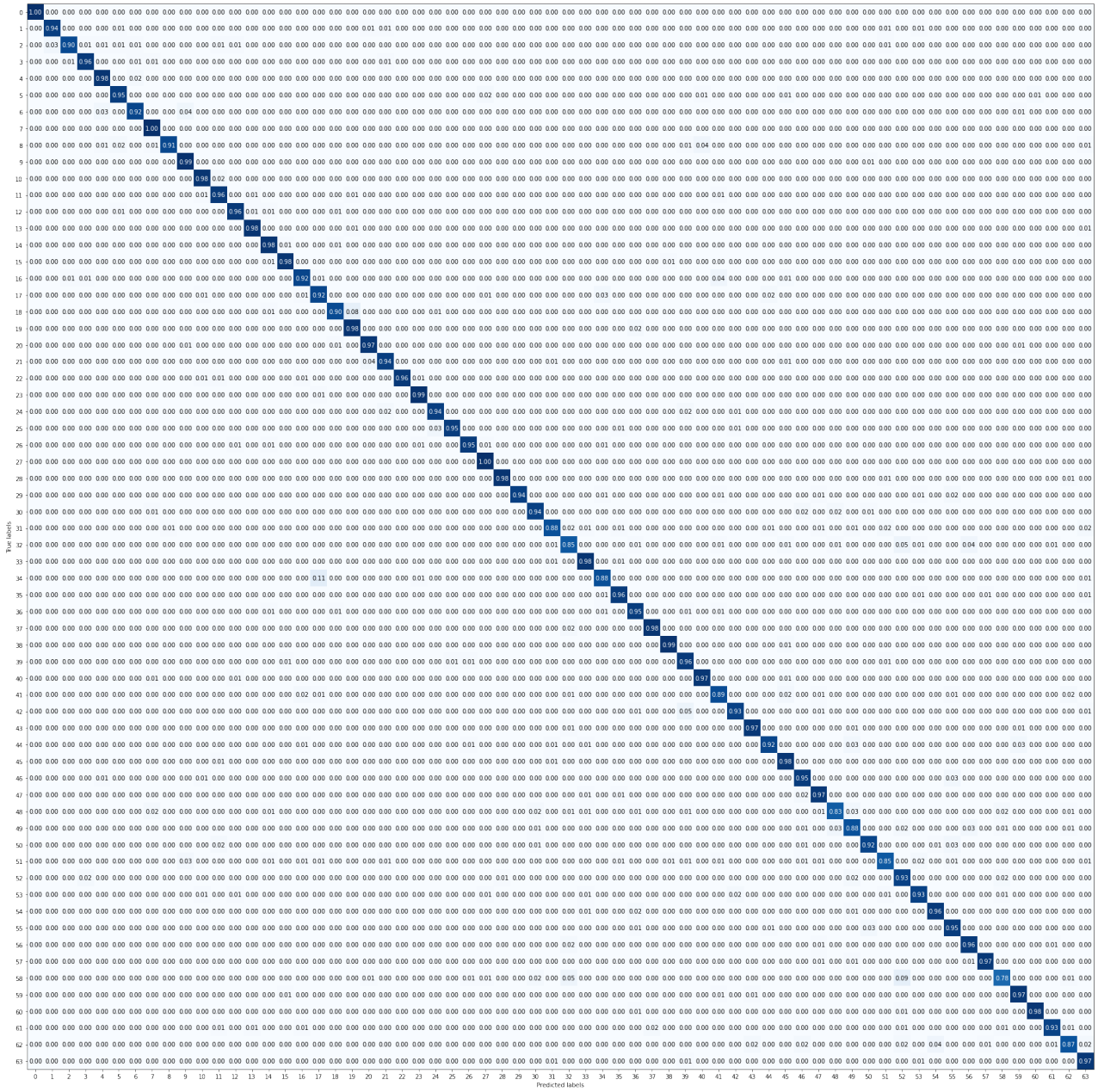


Figure 6.15: VGG-16 normalized confusion matrix

6.2. Recognition in Web App

We have implemented a web application using Flask for our project. The web application consists of a single page, where the user can upload an image of a Prachalit script character through the web app, and the app will predict the character in the image using a pre-trained convolutional neural network (CNN) model. The uploaded image is first checked to see if it has a valid file extension (png, jpg, jpeg, gif). The valid file is then preprocessed accordingly, and passed through the model to predict the class of the image. The predicted class is mapped to the corresponding Prachalit script character using simple dictionary data structure in Python, and thus the predicted class can be mapped to the character. Finally, the predicted character is displayed on a results page using the Flask templating engine. Figure 6.17 shows the character image being uploaded to the web app, and figure 6.18 shows the character being predicted accurately.

Figure 6.21 and 6.22 show the incorrect prediction of the characters in web application. One of the reasons for wrong prediction could be the similarity between characters in the input image. For example, if the model is trained to recognize Devanagari characters, it might get confused between similar-looking characters. Another reason could be the quality of the uploaded image. If the input image is blurry or has low resolution, it might be difficult for the model to accurately recognize the characters. Therefore, it is important to ensure that the input images are of good quality and the characters are clearly visible. Additionally, it might be beneficial to explore techniques like data augmentation or fine-tuning the model to improve its accuracy on similar-looking characters.

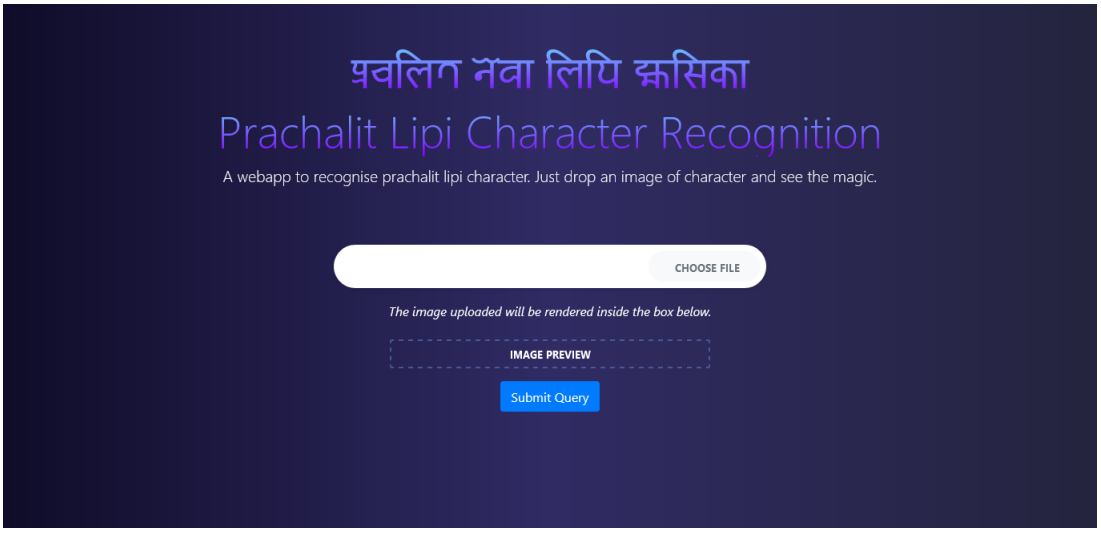


Figure 6.16: Web app home page

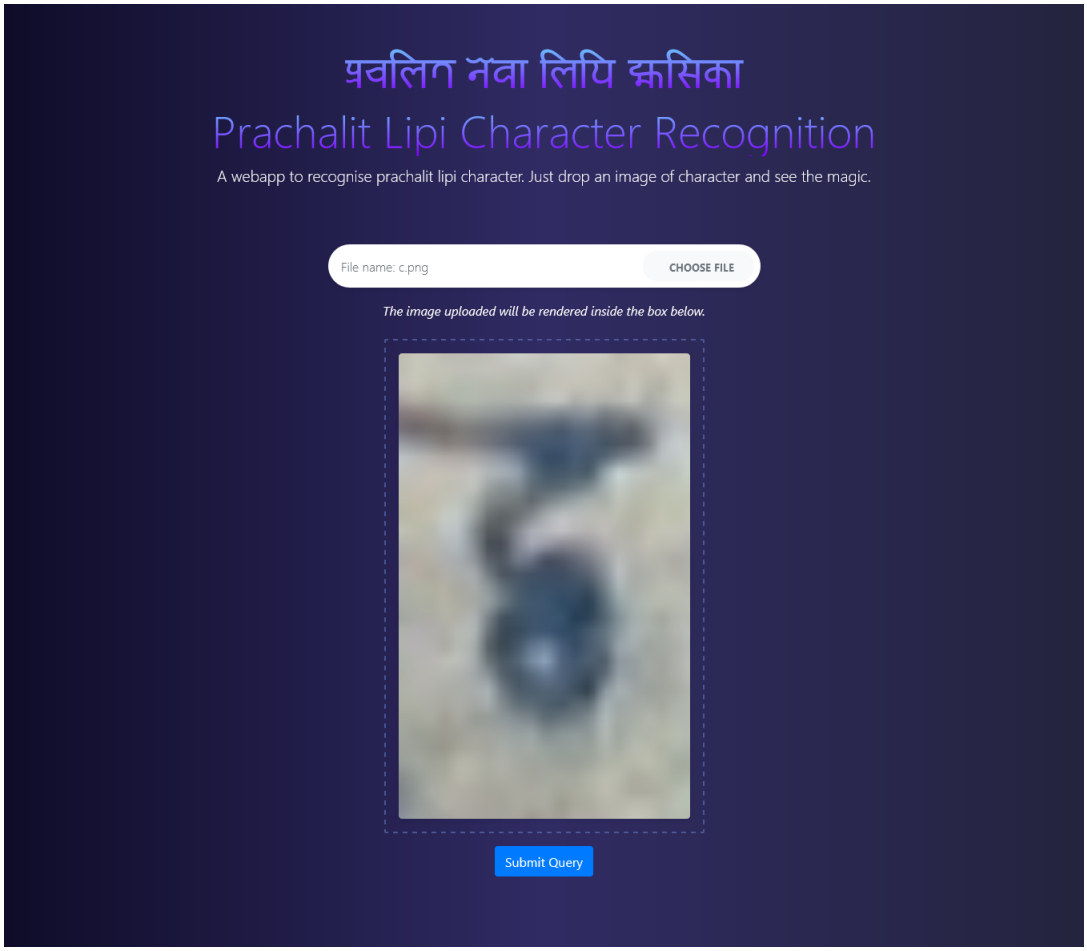


Figure 6.17: Uploading image for prediction(NGA)

Figure 6.18: Correct prediction(NGA)

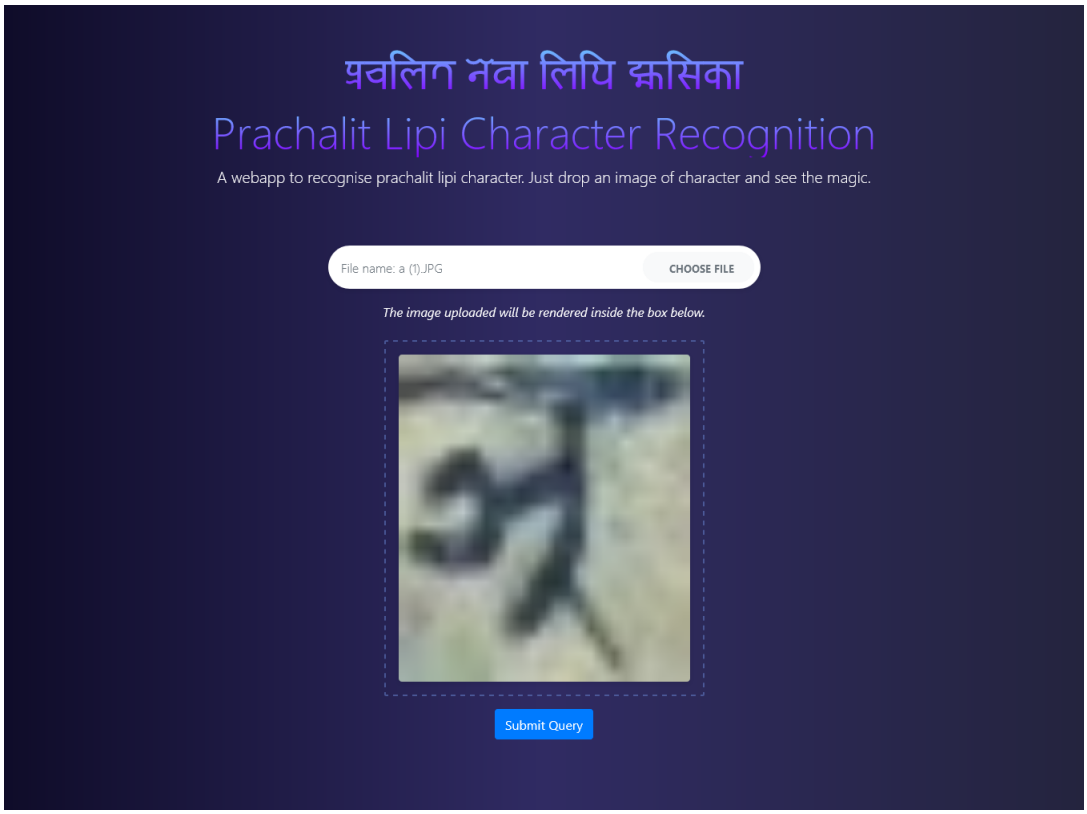


Figure 6.19: Uploading image for prediction(A)

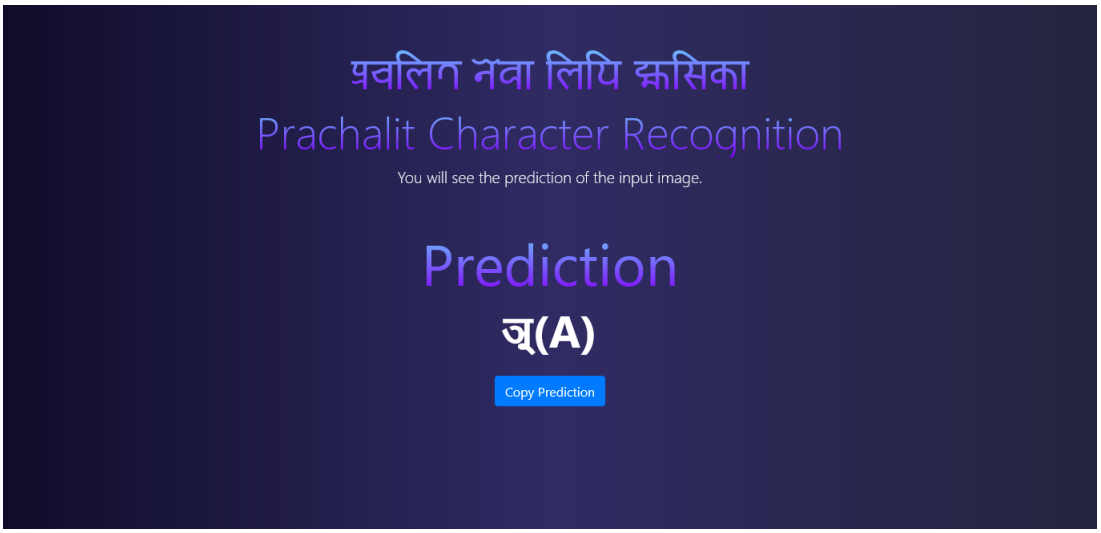


Figure 6.20: Correct prediction(A)

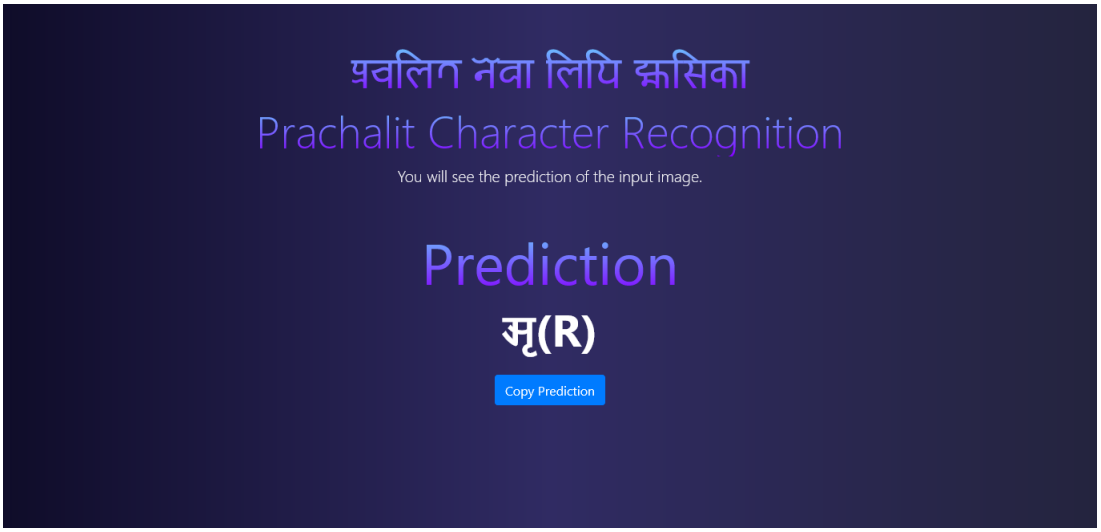


Figure 6.21: Uploading image for prediction(RR)

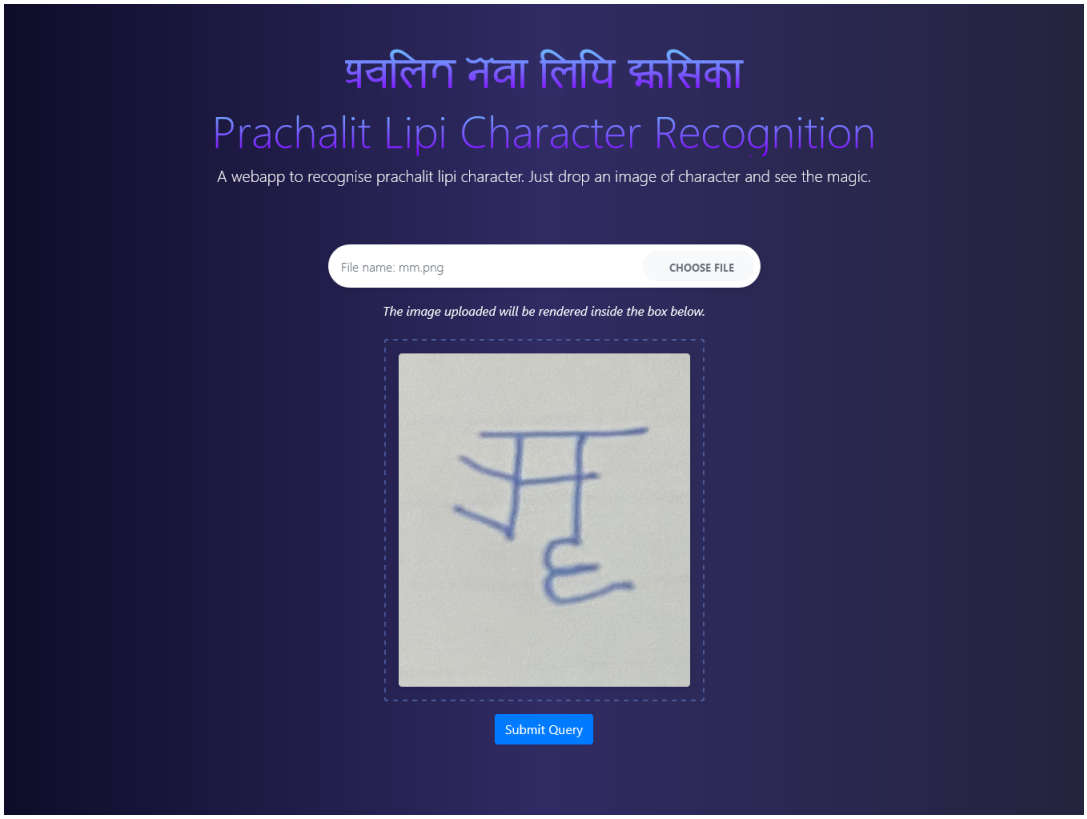


Figure 6.22: Wrong Prediction(R)

6.3. Confused Characters

Character	Character	Reason
ੴ	ੴ	Slightly different in upper modifier
ਸ੍ਰ	ਸ੍ਰ	Slightly different in lower modifier
੭	੭	Slightly different in upper modifier
੦	੦	Very Similar shape wise
੩	੩	Slightly different in upper modifier

Figure 6.23: Most Confused Characters

Testing on the unseen data of the characters, the model found it difficult to distinguish between these above mentioned set of characters due to their identical features and shape.Due to the variation in human handwriting, the model found it difficult to distinguish between remotely similar characters.

7. CONCLUSION

We provided a brand-new Prachalit Script dataset that we created ourselves because it was not already available. This dataset consists of 51200 images divided into 64 classes that include consonants, vowels, and numerals, with 800 images per class. The images was carefully compiled from old manuscripts and handwritten writings. After then, the dataset was expanded utilizing augmentation methods including rotation, skewing, and shearing of the gathered images. The dataset that has been prepared so far may be utilized to create more ML models that need this data. Because CNN possesses the qualities of data invariance, and noise immunity, it was chosen for the project. For the cleaned dataset, four CNN architectures (LeNet, AlexNet, Resnet and VGG16) were trained. VGG was the most accurate of the four and was more accurate at predicting data that had not yet been seen. Characters in the sample that were extremely similar to one another presented a categorization issue. As a result, it was challenging to extract features from characters that appeared similar, which led to mistakes and incorrect predictions. To address the issue of overfitting to the training dataset, we additionally performed early stopping, regularization using dropout layers, and hyperparameter adjustment. The inclusion of a dropout layer can aid in classifying a difficult dataset like ours. The model was then made available through the Flask API, and a website was developed to make it easier for users to provide input images and obtain anticipated results.

8. FUTURE ENHANCEMENTS

The technological advancements that can be further carried out in the project are as follows.

- Recognition of half letters and combined characters and eventual recognition of word.
- Deployment of Model into cloud platforms like: AWS, Azure allowing for the model to run from any device.

A. APPENDIX

A.1. Project Schedule

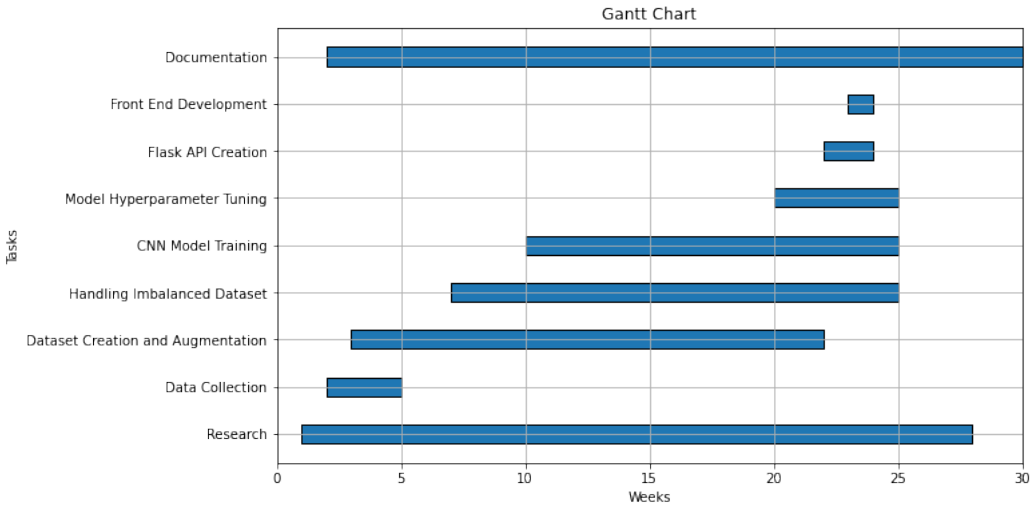


Figure A.1: Project Schedule

A.2. Dataset Description

Dataset has 64 classes with 800 images each i.e. 51200 images in total.

Character	০	১	২	৩	৪	৫	৬	৭	৮	৯
Number of images	800	800	800	800	800	800	800	800	800	800
Total: 8000 images										

Figure A.2: Data Distribution Of Numbers

Character	অ	আ	ই	ঈ	উ	ঊ	ঋ	ৠ	ঌ	ৡ
Number of images	800	800	800	800	800	800	800	800	800	800
Character	এ	ঐ	ও	ঔ						
Number of images	800	800	800	800						
Total: 11200 images										

Figure A.3: Data Distribution Of Vowels

Character	ক	খ	গ	ঘ	ঢ	ব	ছ	জ	ম	ঞ
Number of images	800	800	800	800	800	800	800	800	800	800
Character	ট	ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন
Number of images	800	800	800	800	800	800	800	800	800	800
Character	প	ফ	ব	ভ	ম	য	ন	ল	ক্ষ	ব
Number of images	800	800	800	800	800	800	800	800	800	800
Character	শ	ষ	স	হ						
Number of Images	800	800	800	800						
Total: 27200 images										

Figure A.4: Data Distribution Of Consonants

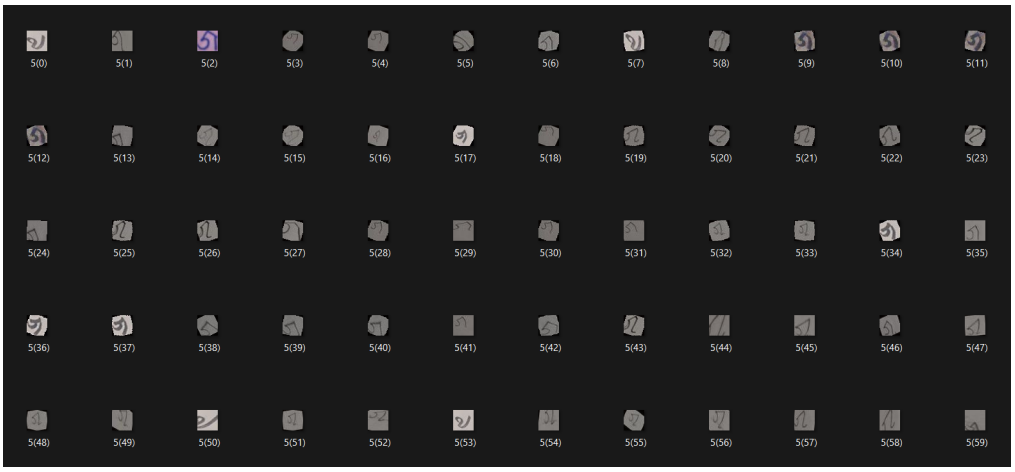


Figure A.5: Dataset Of number

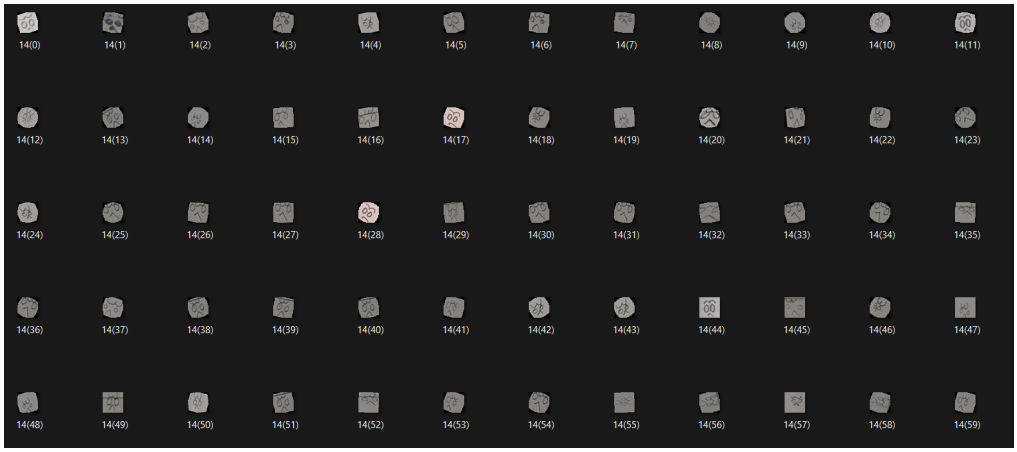


Figure A.6: Dataset Of vowel

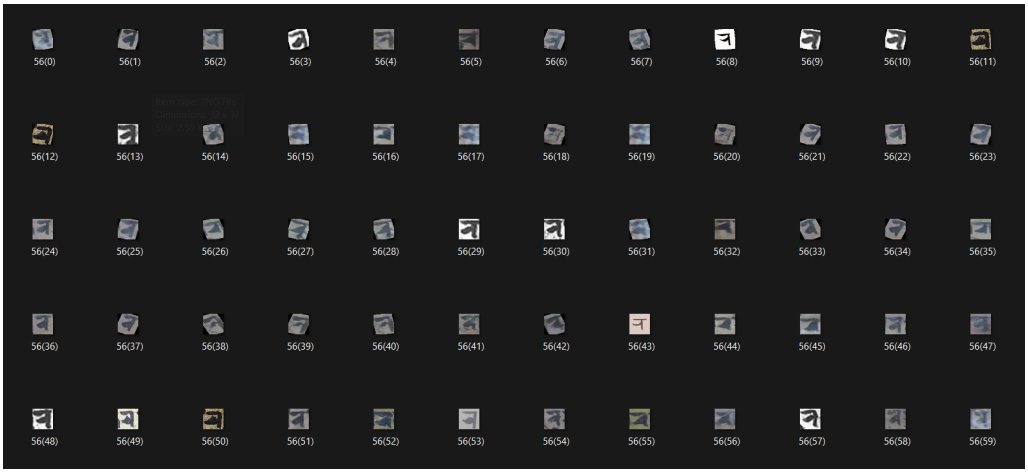


Figure A.7: Dataset Of Consonant

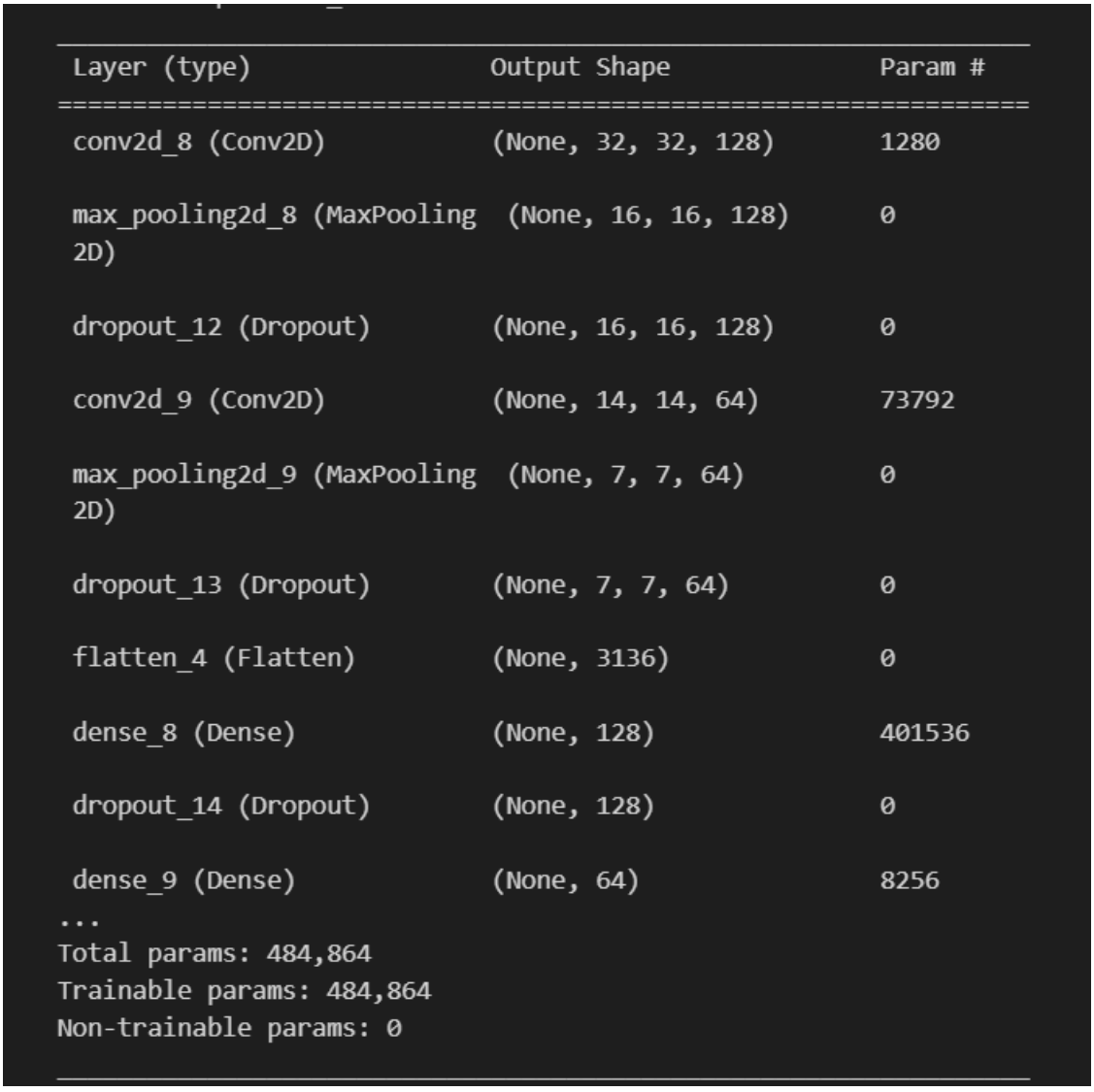
```
Training set shape: (32768, 32, 32, 3) (32768,)
Validation set shape: (8192, 32, 32, 3) (8192,)
Testing set shape: (10240, 32, 32, 3) (10240,)
```

Figure A.8: Dataset Distribution Among Training, Testing and Validation Set

A.3. Models Summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 32)        896

 conv2d_7 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dropout_4 (Dropout)         (None, 16, 16, 32)        0

 conv2d_8 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_9 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_5 (Dropout)         (None, 8, 8, 64)          0

 conv2d_10 (Conv2D)          (None, 8, 8, 128)         73856

 conv2d_11 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_6 (Dropout)         (None, 4, 4, 128)         0

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 512)               1049088

 dropout_7 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 64)                32832

=================================================================
Total params: 1,368,928
Trainable params: 1,368,928
Non-trainable params: 0
```

Figure A.9: VGG Model Summary



Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 32, 32, 128)	1280
max_pooling2d_8 (MaxPooling 2D)	(None, 16, 16, 128)	0
dropout_12 (Dropout)	(None, 16, 16, 128)	0
conv2d_9 (Conv2D)	(None, 14, 14, 64)	73792
max_pooling2d_9 (MaxPooling 2D)	(None, 7, 7, 64)	0
dropout_13 (Dropout)	(None, 7, 7, 64)	0
flatten_4 (Flatten)	(None, 3136)	0
dense_8 (Dense)	(None, 128)	401536
dropout_14 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 64)	8256

...
Total params: 484,864
Trainable params: 484,864
Non-trainable params: 0

Figure A.10: LeNet Model Summary

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 32, 32, 1)]       0

 conv2d_6 (Conv2D)           (None, 32, 32, 64)        640

 conv2d_7 (Conv2D)           (None, 32, 32, 64)        36928

 conv2d_8 (Conv2D)           (None, 32, 32, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_9 (Conv2D)           (None, 16, 16, 128)       73856

 conv2d_10 (Conv2D)          (None, 16, 16, 128)       147584

 conv2d_11 (Conv2D)          (None, 16, 16, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 flatten_1 (Flatten)         (None, 8192)              0
...
Total params: 9,390,784
Trainable params: 9,390,784
Non-trainable params: 0
_________________________________________________________________
```

Figure A.11: ResNet Model Summary

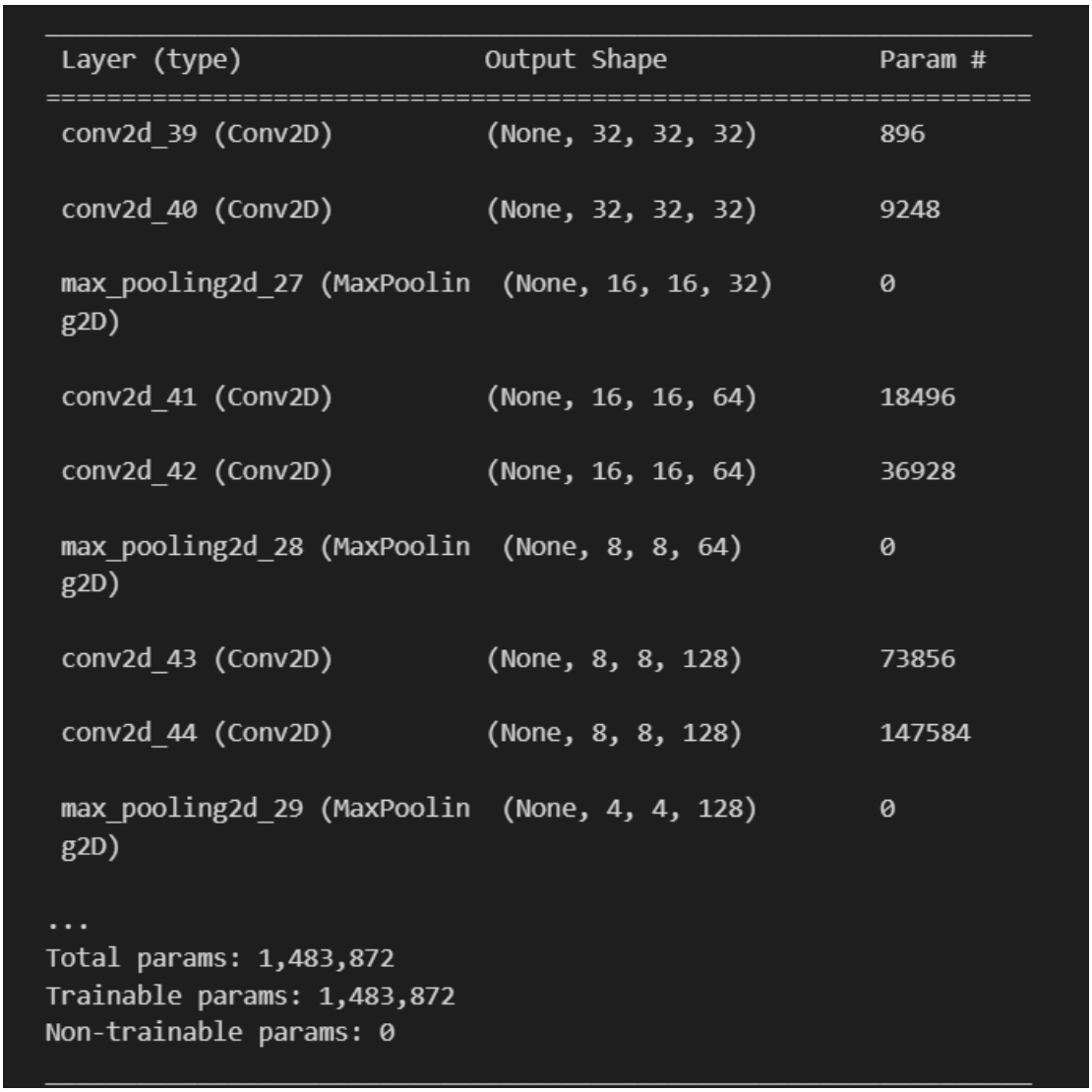


Figure A.12: AlexNet Model Summary

A.4. Models Specifications

Parameters	Values
Batch Size	64
Epochs	50
Image Shape	(32,32,1)
Model	VGG16
Classification classes	64
Learning rate	0.001
Loss Function	Sparse Categorical Cross Entropy
Optimizer	Adam
Filters	32,64,128
Kernel Size	(3,3)
MaxPooling Size	(2,2)
Dropouts	0.25,0.25,0.25,0.50
Activation Functions	Relu and Softmax(Output Layer)
Total Parameters	1,368,928
Trainable Parameters	1,368,928
Non Trainable Parameters	0

Figure A.13: VGG Model Specifications

Parameters	Values
Batch Size	64
Epochs	50
Image Shape	(32,32,1)
Model	AlexNet
Classification classes	32
Learning rate	0.001
Loss Function	Sparse Categorical Cross Entropy
Optimizer	Adam
Filters	128
Kernel Size	(3,3)
MaxPooling Size	(2,2)
Dropouts	0.25,0.25
Activation Functions	Relu and Softmax(Output Layer)
Total Parameters	1,483,872
Trainable Parameters	1,483,872
Non Trainable Parameters	0

Figure A.14: Alexnet Model Specifications

Parameters	Values
Batch Size	64
Epochs	50
Image Shape	(32,32,1)
Model	Resnet
Classification classes	32
Learning rate	0.001
Loss Function	Sparse Categorical Cross Entropy
Optimizer	Adam
Filters	64,128
Kernel Size	(3,3)
MaxPooling Size	(2,2)
Dropouts	0.25,0.50
Activation Functions	Relu and Softmax(Output Layer)
Total Parameters	9,390,784
Trainable Parameters	9,390,784
Non Trainable Parameters	0

Figure A.15: Resnet Model Specifications

Parameters	Values
Batch Size	64
Epochs	50
Image Shape	(32,32,1)
Model	Lenet
Classification classes	32
Learning rate	0.001
Loss Function	Sparse Categorical Cross Entropy
Optimizer	Adam
Filters	64,128
Kernel Size	(3,3)
MaxPooling Size	(2,2)
Dropouts	0.25,0.25
Activation Functions	Relu and Softmax(Output Layer)
Total Parameters	484,564
Trainable Parameters	484,564
Non Trainable Parameters	0

Figure A.16: Lenet Model Specifications

References

[1] Central Bureau of Statistics, Nepal, *Official Summary of Census*. Government of Nepal, 2011.

[2] H. Shakya, *Nepal Lipi Prakash*. Kathmandu: Mandas Lumanti Prakashan, 1993.

[3] Y. Roh, G. Heo, and S. E. Whang, "A survey on data collection for machine learning: a big data-ai integration perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1328–1347, 2019.

[4] A. Gonfalonieri, "How to build a data set for your machine learning project," *Towards data science*, 2019.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] 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)*. IEEE, 2015, pp. 1–6.

[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)*. IEEE, 2007, pp. 454–461.

[11] S. Shelke and S. Apte, "A novel multi-feature multi-classifier scheme for unconstrained handwritten devanagari character recognition," in *Frontiers in Handwriting Recognition, International Conference on*. IEEE Computer Society, 2010, pp. 215–219.

[12] ——, "A fuzzy based classification scheme for unconstrained handwritten devanagari character recognition," in *2015 International Conference on Communication, Information & Computing Technology (ICCICT)*. IEEE, 2015, pp. 1–6.

[13] ——, "Performance optimization and comparative analysis of neural networks for handwritten devanagari character recognition," in *2016 International Conference on Signal and Information Processing (IConSIP)*. IEEE, 2016, pp. 1–5.

[14] P. K. Sonawane and S. Shelke, "Handwritten devanagari character classification using deep learning." in *2018 International Conference on Information, Communication, Engineering and Technology (ICICET)*. IEEE, 2018, pp. 1–4.

[15] T. R. Das, S. Hasan, M. R. Jani, F. Tabassum, and M. I. Islam, "Bangla handwritten character recognition using extended convolutional neural network," *Journal of Computer and Communications*, vol. 9, no. 3, pp. 158–171, 2021.

[16] D. S. Joshi and Y. R. Risodkar, "Deep learning based gujarati handwritten character recognition," in *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*. IEEE, 2018, pp. 563–566.

[17] D. S. Prashanth, R. V. K. Mehta, K. Ramana, and V. Bhaskar, "Handwritten devanagari character recognition using modified lenet and alexnet convolution neural networks," *Wireless Personal Communications*, vol. 122, pp. 349–378, 2022.

[18] M. Mhapsekar, P. Mhapsekar, A. Mhatre, and V. Sawant, "Implementation of residual network (resnet) for devanagari handwritten character recognition," in *Advanced Computing Technologies and Applications: Proceedings of 2nd International Conference on Advanced Computing Technologies and Applications—ICACTA 2020*. Springer, 2020, pp. 137–148.

[19] M. I. Fanany *et al.*, "Handwriting recognition on form document using convolutional neural network and support vector machines (cnn-svm)," in *2017 5th international conference on information and communication technology (ICoIC7)*. IEEE, 2017, pp. 1–6.

[20] M. Liwicki, A. Graves, S. Fernàndez, H. Bunke, and J. Schmidhuber, "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks," in *Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.

[21] T. Fischer, "Online handwritten character recognition with capacitive sensors," 2018.

[22] T. Bluche, J. Louradour, and R. Messina, "Scan, attend and read: End-to-end handwritten paragraph recognition with mdlstm attention," in *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 1050–1055.

[23] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 11, pp. 2298–2304, 2016.

[24] S. Gautam, "Devnagari handwritten word recognition with deep learning," 2019.

[25] B. Liu, X. Xu, and Y. Zhang, "Offline handwritten chinese text recognition with convolutional neural networks," *arXiv preprint arXiv:2006.15619*, 2020.

[26] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. ACM, 2010, pp. 807–814.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] S. Y. Diaba, M. Shafie-Khah, and M. Elmusrati, "Cyber security in power systems using meta-heuristic and deep learning algorithms," *IEEE Access*, vol. 11, pp. 18 660–18 672, 2023.

[30] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[31] T. Fawcett, "Roc graphs: Notes and practical considerations for researchers," *Machine learning*, vol. 31, no. 1, pp. 1–38, 2003.

[32] Y. Sasaki, "The truth of the f-measure," *Journal of Machine Learning Research*, vol. 8, pp. 313–318, 2007.

[33] A. C. Bovik, *Handbook of Image and Video Processing*. Academic Press, 2010.