MVJ COLLEGE OF ENGINEERING, BENGALURU-560067

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF CSE -DATA SCIENCE

CERTIFICATE

Certified that the minor project work titled ***'Image recognition of Devanagiri script using CNN'*** is carried out by **SELVKARTHIK S (1MJ22CD047), N TEJESWAR REDDY (1MJ22CD036), M T VINAY (1MJ22CD027)** and **RAKESH KUMAR PANDEY (1MJ22CD043)** who are confide students of MVJ College of Engineering, Bengaluru, in partial fulfilment for the award of Degree of **Bachelor of Engineering in CSE-Data Science** of the Visvesvaraya Technological University, Belagavi during the year 2024-2025. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the mini project report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of major project work prescribed by the institution for the said Degree.

Signature of Guide	Signature of Head of the Department	Signature of Principal
Prof.Arnab	**Prof.Rekha P.**	**Dr. Ajayan K R**
Assistant Professor	**Assistant Professor**	**Principal**
Dept. of CSE-Data Science	**Dept. of CSE-Data Science**	**MVJ College of Engineering**
MVJ College of Engineering	**MVJ College of Engineering**	**Banglore-560067**
Banglore-560067	**Banglore-560067**	

External Viva

Name of Examiners	Signature with Date
1	
2	

MVJ COLLEGE OF ENGINEERING, BENGALURU-560067

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF CSE (DATA SCIENCE)

DECLARATION

We, SELVKARTHIK S, **N TEJESWAR REDDY**, **M T VINAY**, **RAKESH KUMAR PANDEY** students of fifth semester B.E., Department of **CSE-DATA SCIENCE** MVJ College of Engineering, Bengaluru, hereby declare that the mini project titled ***'Image recognition of Devanagiri script using CNN'*** has been carried out by us and submitted in partial fulfilment for the award of Degree of **Bachelor of Engineering** in **CSE-DATA SCIENCE** during the year 2024-25.

Further we declare that the content of the dissertation has not been submitted previously by anybody for the award of any Degree or Diploma to any other University.

We also declare that any Intellectual Property Rights generated out of this project carried out at MVJCE will be the property of MVJ College of Engineering, Bengaluru and we will be one of the authors of the same.

Place: Bengaluru
Date:

Name **Signature**

1. SELVKARTHIK S (1MJ22CD047)

2. N TEJESWAR REDDY (1MJ22CD036)

3. M T VINAY (1MJ22CD027)

4. RAKESH KUMAR PANDEY (1MJ22CD043)

ACKNOWLEDGEMENT

We are indebted to our guide**, Prof. Arnab,** Professor**, Department of CSE-DATA SCIENCE, MVJ College of Engineering** for her wholehearted support, suggestions and invaluable advice throughout our project work and also for the help in the preparation of this report.

Our sincere thanks to **Prof. Rekha**, Assistant / Associate Professor and Head, Department of CSE-Data Science Engineering, MVJCE for her support and encouragement.

We express sincere gratitude to our beloved Principal, **Dr. Ajayan K R** for all his support towards this project work.

Lastly, we take this opportunity to thank our family members and friends who provided all the back up support throughout the project work.

ABSTRACT

This project focuses on applying Convolutional Neural Networks (CNNs) for recognizing Devanagari script, a widely used writing system in languages like Hindi, Marathi, and Sanskrit. Given its linguistic and cultural significance, the goal is to develop a robust model capable of accurately identifying and classifying characters from images of handwritten and printed Devanagari text. The project starts with curating and preprocessing a diverse dataset, including various handwriting styles, font types, and sizes. This diversity ensures the model's generalizability to real-world variations. Data augmentation techniques, such as rotation, scaling, and noise addition, further enhance the model's robustness against inconsistencies in character representation. A custom CNN architecture is designed to capture the intricate patterns and structures of Devanagari script, leveraging the network's hierarchical feature extraction capabilities. The model's performance is evaluated using accuracy and loss metrics, with results showing significant improvements over traditional optical character recognition (OCR) methods. Key challenges, such as differentiating visually similar characters, are addressed through iterative optimization and fine-tuning. This project demonstrates the potential of deep learning in OCR applications, contributing to the digitization and accessibility of Devanagari script. It holds promise for practical applications like digitizing historical texts, developing educational tools, and enhancing natural language processing (NLP) systems. By advancing the recognition of complex scripts, this work supports broader adoption and integration of Devanagari into modern technologies.

.

TABLE OF CONTENTS

List of Figures

CHAPTER-1

INTRODUCTION

CHAPTER 1

Introduction

1.1 Aim:

Handwriting identification is one of the most challenging research domains in computer vision. This project aims to develop an advanced handwriting recognition system for Devanagari script by leveraging Transfer Learning with Deep Convolutional Neural Networks (DCNNs). The primary goal is to achieve high accuracy, robustness, and efficiency in recognizing Devanagari characters, which exhibit intricate shapes and significant inter-character similarities. By utilizing pre-trained models as a starting point, this project seeks to bridge the gap between traditional handcrafted feature-based methods and modern deep learning techniques, ultimately leading to improved recognition capabilities even under diverse and challenging conditions.

1.2 Motivation:

The motivation behind this project stems from the growing need to address the challenges associated with handwriting recognition in non-English scripts, particularly Devanagari script. Devanagari is used by several Indian languages, including Hindi, Sanskrit, Nepali, Marathi, Sindhi, and Konkani. The recognition of Devanagari script is essential for various applications, such as digital document processing, historical manuscript digitization, and educational tools for learning these languages. Accurate recognition of Devanagari characters can significantly enhance the efficiency and effectiveness of these applications, leading to better data management, preservation of cultural heritage, and improved learning experiences.

1.3 Objective:

The primary objectives of this project are as follows:

- **To Analyse Challenges**: Identify and understand the complexities and challenges involved in recognizing Devanagari characters due to their intricate shapes and similarities.
- **To Explore Deep Learning Approaches**: Investigate various deep learning models and techniques that can be applied to the task of Devanagari script recognition.
- **To Implement Transfer Learning**: Utilize pre-trained deep learning models as a starting point to improve the recognition accuracy and efficiency for Devanagari script.
- **To Evaluate Performance**: Assess the performance of the implemented model through various metrics such as accuracy, precision, recall, and F1-score.
- **To Compare with Traditional Methods**: Compare the results obtained from deep learning approaches with those from traditional handcrafted feature-based methods to highlight the advancements made.

1.4 Existing System:

Existing systems for Devanagari script recognition have made significant strides in recent years, leveraging advanced machine learning techniques to improve accuracy and efficiency. Here are some notable examples:

- **DeepNetDevanagari**
- **Deep Structure Learning of Image Quadrants**
- **Multiclass Recognition of Offline Handwritten Devanagari Characters using CNN**

1.5 Proposed System

The proposed system leverages deep learning techniques, particularly Convolutional Neural Networks (CNNs), to achieve accurate and efficient recognition of handwritten Devanagari script characters. The system is designed to process, classify, and decode handwritten characters into their corresponding Sanskrit letters. Key features of the proposed system include:

- **Real-World Applicability:** The system can be employed in various applications, such as digital document processing, historical manuscript digitization, and educational tools for language learning.
- **Accuracy:** Leveraging CNNs for feature extraction significantly improves recognition accuracy, particularly for complex scripts like Devanagari.
- **Efficiency:** The preprocessing steps ensure that the images are uniform and optimized for training, reducing computational load and enhancing training speed.
- **Scalability:** The system is designed to handle a large number of classes, making it scalable to recognize an extensive set of Devanagari characters.

This proposed system aims to provide a robust solution for recognizing handwritten Devanagari script characters, addressing the unique challenges posed by the script's complexity and variability.

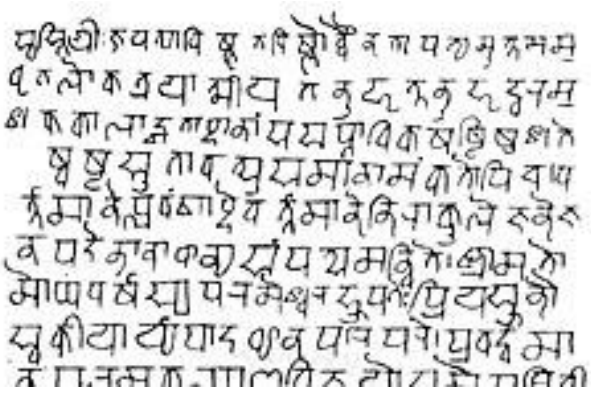

Fig 1. Devanagiri alphabets

CHAPTER-2

LITERATURE SURVEYS

CHAPTER 2

LITERATURE SURVEYS

The following research articles are selected for review, and discussed in detail.

Nagender Aneja∗ and Sandhya Aneja ∗

∗Universiti Brunei Darussalam Brunei Darussalam proposed a "Transfer Learning using CNN for Handwritten Devanagari Character Recognition" an analysis of pre-trained models to recognize handwritten Devanagari alphabets using transfer learning for Deep Convolution Neural Network (DCNN). This research implements AlexNet, DenseNet, Vgg, and Inception ConvNet as a fixed feature extractor. Wthey implemented 15 epochs for each of AlexNet, DenseNet 121, DenseNet 201, Vgg 11, Vgg 16, Vgg 19, and Inception V3. Results show that Inception V3 performs better in terms of accuracy achieving 99% accuracy with average epoch time 16.3 minutes while AlexNet performs fastest with 2.2 minutes per epoch and achieving 98% accuracy.

As per **Prafulla E Ajmire and Shankar Eknath Warkhede.**

Artificial Neural Networks (ANNs) have been widely utilized for Devanagari script recognition due to their ability to handle non-linear patterns and extract features effectively. Their study titled " An Analytical Study of Devanagari Script Recognition. " demonstrated that Convolutional Neural Networks (CNNs), a subclass of ANNs, offer high accuracy but require substantial computational resources and training time. Similarly, Gupta et al. (2018) explored Support Vector Machines (SVMs) in "Efficient SVMs for Devanagari Character Classification," highlighting their efficiency for smaller datasets and robust handling of high-dimensional data. Meanwhile, Roy et al. (2017) implemented Particle Swarm Optimization (PSO) in "Optimized Feature Selection for Devanagari Characters," demonstrating how PSO enhances feature selection, leading to improved recognition performance. Additionally, Verma et al. (2019) applied Genetic Algorithms (GAs) in "Genetic Algorithms for Script Recognition," showcasing their effectiveness in feature optimization and global optima discovery. These methods, while advancing

recognition accuracy, face challenges like computational cost, data variability, and scalability, indicating the need for continued exploration of advanced techniques like deep learning and multilingual system integration.

As per **Holambe et al**. (2018), research on Devanagari script recognition has focused on developing efficient classification techniques for character recognition. Their study titled "A Comprehensive Review of Devanagari Script Recognition" involved the creation and preprocessing of a handwritten character database. They utilized local intensity distribution of gradients for feature extraction, which proved effective in capturing essential character attributes. The K-Nearest Neighbor (KNN) classifier was applied to evaluate recognition performance, highlighting its simplicity and adaptability for small datasets. However, the study identified challenges such as variability in handwriting styles and scalability to larger datasets. The authors emphasized the potential of gradient-based features in enhancing recognition accuracy while noting the limitations of computational efficiency and dataset diversity. Their findings underscore the need for advanced approaches, such as neural networks and deep learning, to overcome these challenges and achieve higher accuracy in Devanagari script recognition.

As per **Prathiba Singh et al**. (2015), the application of mini-batch stochastic gradient descent (SGD) learning was explored for improving the performance of Multilayer Perceptrons (MLPs) in Devanagari handwritten character and numeral recognition. Their study titled "On the Performance Improvement of Devanagari Handwritten Character Recognition" demonstrated that mini-batch SGD effectively reduces variance in gradient estimates and leverages hierarchical memory organization in modern computers. To prevent overfitting, L2-weight decay was integrated into the learning process. Experiments were conducted using direct pixel intensity values as features, followed by a flexible zone-based gradient feature extraction approach. Results highlighted the benefits of SGD in improving recognition accuracy while addressing overfitting. However,

computational demands and the choice of feature extraction techniques remain critical considerations for further optimization in this domain.

Kannuru Padmaja presented a study on "Devanagari Handwritten Character Recognition Using Deep Learning", where she explored the application of deep learning, specifically convolutional neural networks (CNNs), for handwritten character recognition in Devanagari script in her 2025 paper. The research aims to automate the recognition of Devanagari characters, which consist of 12 vowels and 36 consonants, contributing to advancements in various automation systems. The paper outlines a comprehensive framework consisting of five main steps: pre-processing, segmentation, feature extraction, prediction, and post-processing. The study employed CNN-based models to enhance recognition accuracy, coupled with image processing techniques to refine predictions. Results demonstrated a significant improvement in character recognition accuracy, showcasing the effectiveness of CNNs in recognizing complex handwritten characters. While the study's methodology exhibited promising results, it acknowledged the challenges in recognizing distorted or noisy handwriting, suggesting potential areas for improvement. This research highlights the potential of deep learning for character recognition in Indian scripts.

Rahul S. Narsing presented a study on "Devanagari Character Recognition using Image Processing & Machine Learning", where he focused on the recognition of handwritten Devanagari characters in his 2025 paper. This research addresses the gap in character recognition systems for Devanagari script, which is used in languages like Hindi, Marathi, Nepali, and Sanskrit. The paper investigates the recognition of a total of 58 handwritten Devanagari characters, encompassing vowels (220), consonants (2000), and digits (2000), resulting in a dataset of 94,640 images. The approach involves scanning handwritten characters and applying image transformation techniques to decompose them into individual character images. A machine learning-based model is then employed for recognition. The study reports a final accuracy of around 90%, demonstrating the

effectiveness of image processing and machine learning in handwritten character recognition. The work highlights the significance of Devanagari character recognition in reducing manual labor and converting handwritten text into a digital format. However, the paper also notes that further advancements are needed in this field to improve accuracy and handle more complex handwritten forms. This research emphasizes the importance of special attention to Devanagari script for effective analysis and processing.

Shalini Puria and Satya Prakash Singh presented a study on "An Efficient Devanagari Character Classification in Printed and Handwritten Documents using SVM", where they proposed a model for Devanagari character classification using Support Vector Machines (SVM) in their 2025 paper. The research aims to address the challenges in recognizing printed and handwritten Devanagari characters in mono-lingual Hindi, Sanskrit, and Marathi documents. The paper outlines a multi-step approach that involves pre-processing the images, segmenting them using projection profiles, removing the shirorekha (the horizontal line above characters), extracting features, and classifying the characters into predefined categories. The proposed system achieved impressive results, with average classification accuracies of 99.54% for printed images and 98.35% for handwritten images, outperforming other existing recognition techniques. This work highlights the efficacy of SVM in character classification and demonstrates its potential for both printed and handwritten Devanagari documents. The study underlines the robustness of the system and its superiority in terms of accuracy, making it a promising solution for Devanagari character recognition.

Rajiv Kumar and Kiran Kumar Ravulakollu presented a study on "Handwritten Devanagari Digit Recognition: Benchmarking on New Dataset", where they conducted a comprehensive evaluation of handwritten Devanagari digit recognition in their 2025 paper. The research focused on the benchmarking of digit recognition using the CPAR-2012 dataset, which includes a range of handwritten numerals, characters, and text. The paper explores various feature extraction techniques, from simple pixel-based features to

more complex gradient features using Kirsch and wavelet transforms. Several classification schemes were tested, with the combination of gradient and direct pixel features using the KNN classifier achieving the highest recognition accuracy of 95.2%. This accuracy was further improved to 97.87% using a multi-stage classifier ensemble scheme. Additionally, the paper highlights the development of the CPAR-2012 dataset, which contains 35,000 handwritten numerals, 82,609 isolated characters, 4,000 digitized data collection forms, and more. The study provides valuable insights into the performance of different feature extraction methods and classification techniques, contributing to the advancement of Devanagari optical document recognition research.

Shailendra Kumar Shrivastava and Pratibha Chaurasia presented a study on "Handwritten Devanagari Lipi using Support Vector Machine", where they proposed a Support Vector Machine (SVM)-based technique for Devanagari character recognition in their 2025 paper. The research addresses the challenge of recognizing Devanagari characters, which exhibit significant correlation with each other, making their faithful recognition difficult. The paper uses energy features of segmented characters for classification, highlighting the impact of the number of segments on recognition accuracy. The study explores the effect of different segmentation schemes on the recognition rate and also examines the performance of various SVM kernels. The results demonstrate that an increased number of segments improves the recognition rate, with performance varying depending on the kernel used in the SVM model. This work contributes to the development of more effective techniques for recognizing handwritten Devanagari characters and highlights the importance of segmentation and kernel selection in improving classification performance.

CHAPTER-3

SYSTEM REQUIREMENT AND SPECIFICATION

CHAPTER 3

SYSTEM REQUIREMENT AND SPECIFICATION

4.1 HARDWARE :

1. **Processor**:
    - Minimum: Intel i3 or equivalent
    - Recommended: Intel i5/i7/i9 or AMD Ryzen 7/9
2. **Memory (RAM)**:
    - Minimum: 4 GB
    - Recommended: 8GB/16 GB or higher
3. **Storage**:
    - Minimum: 256 GB SSD
    - Recommended: 512 GB SSD or higher (for faster data access and storage of large datasets)
4. **Operating System**:
    - Windows 10/11, Ubuntu 18.04/20.04, macOS (latest version)

Software Requirements:

1. **Programming Language**:
    - Python 3.6 or higher
2. **Libraries and Frameworks**:

- **TensorFlow**: An open-source deep learning framework developed by Google for building, training, and deploying machine learning models.
- **Keras**: A high-level API built on top of TensorFlow for easy and fast prototyping of deep learning models.

  OpenCV: An open-source library providing tools for computer vision tasks such as image processing and analysis.
- **NumPy**: A fundamental package for numerical computing in Python, providing support for arrays and matrices.
- **Pandas**: A data manipulation and analysis library offering data structures like Data Frames for handling structured data.
- **Matplotlib**: A plotting library for creating static, animated, and interactive visualizations in Python.
- **scikit-learn**: A machine learning library providing simple and efficient tools for data mining, analysis, and modeling.

3. **Development Environment**:
   - Jupyter Notebook
   - PyCharm
   - Visual Studio Code

Additional Tools:

- **CUDA Toolkit**: For GPU acceleration (if using an NVIDIA GPU)
- **cuDNN**: GPU-accelerated library for deep neural networks (compatible with TensorFlow)

Specifications:

1. **Data Preprocessing**:

- Image resizing to 200x200 pixels
- Normalization of pixel values to the range [0, 1]
- Label encoding and one-hot encoding for classification

2. **Model Architecture**:
    - Convolutional Neural Network (CNN) with multiple layers
    - Convolutional layers for feature extraction
    - Max-pooling layers for down-sampling
    - Dense layers for classification
    - Softmax activation in the output layer with 174 units (corresponding to 174 classes)
3. **Training Configuration**:
    - Optimizer: Adam
    - Loss Function: Categorical Cross-Entropy
    - Metrics: Accuracy
    - Number of Epochs: 15
    - Batch Size: 32 (adjustable based on available GPU memory)
4. **Evaluation Metrics**:
    - Accuracy
    - Precision
    - Recall
    - F1-Score
5. **Visualization**:

- Training and validation accuracy/loss plots
- Prediction results with decoded Sanskrit letters

By ensuring these system requirements and specifications, you can effectively develop, train, and deploy the handwritten Devanagari script recognition system using advanced deep learning techniques.

CHAPTER 4

SYSTEM DESIGN

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE:

1. User Interface:
    - Image Upload: Allows users to upload images of handwritten Devanagari characters.
    - Results Display: Displays recognized characters and prediction results to the users.
2. Input Layer:
    - Image Acquisition: Receives images from the user interface.
    - Data Validation: Ensures the uploaded images meet required specifications (e.g., image format, resolution).
3. Preprocessing Module:
    - Image Resizing: Resizes images to a uniform dimension (e.g., 200x200 pixels).
    - Normalization: Normalizes pixel values to the range [0, 1] to improve model training efficiency.
    - Label Encoding: Converts class labels into one-hot encoded vectors for multi-class classification.
4. Data Augmentation (Optional):
    - Image Augmentation: Applies transformations such as rotation, scaling, and flipping to increase the diversity of the training dataset and improve model robustness.
5. Data Splitting:

- Training Set: Contains the majority of the data for training the model.
- Validation Set: Used to tune model parameters and prevent overfitting.
- Test Set: Used to evaluate the model's performance on unseen data.

6. Model Architecture:
   - Convolutional Layers: Extracts features from input images using convolutional filters.
   - Activation Functions: Applies non-linear transformations (e.g., ReLU) to introduce non-linearity.
   - Pooling Layers: Reduces spatial dimensions and retains important features (e.g., Max-Pooling).
   - Dropout Layers: Prevents overfitting by randomly setting a fraction of input units to zero during training.
   - Flatten Layer: Converts the 2D feature maps to a 1D vector for input to the dense layers.
   - Dense Layers: Fully connected layers that perform the final classification.
   - Output Layer: Softmax activation to produce probability distributions over 174 classes.
7. Training Module:
   - Loss Function: Categorical Cross-Entropy loss for multi-class classification.
   - Optimizer: Adam optimizer to adjust model parameters and minimize loss.
   - Batch Processing: Trains the model in batches to efficiently utilize computational resources.
   - Epochs: Iterates over the entire training dataset multiple times to improve model performance.

8. Evaluation Module:

    - Model Evaluation: Assesses the model's performance on the test set using metrics such as accuracy, precision, recall, and F1-score.
    - Training History: Records and visualizes training and validation accuracy and loss over epochs.

9. Prediction Module:

    - Inference: Uses the trained model to predict classes for new input images.
    - Decoding: Maps predicted class indices to corresponding Devanagari characters using the class map.

10. Output Layer:

    - Display Results: Shows recognized characters and corresponding class indices to the user.
    - Feedback Loop: Allows users to provide feedback on the recognition results to further refine the model.

4.2 DATA FLOW DIAGRAM:

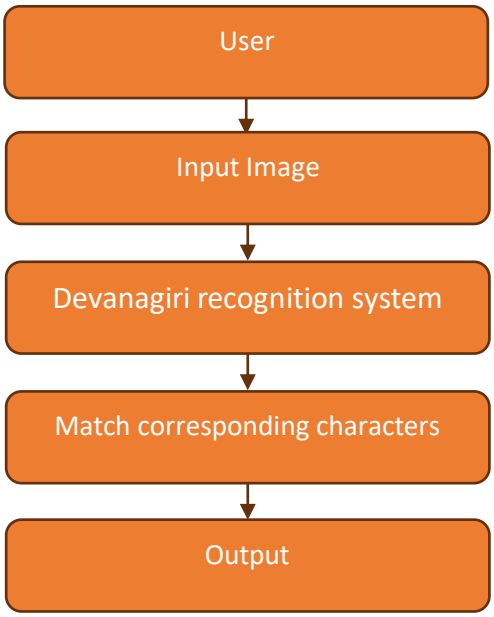

Fig 2.DFD Level 0

DFD Level 0

1. **User**: Uploads images of handwritten Devanagari characters and views the recognized output.

2. **Devanagari Recognition System**: Processes the uploaded images to recognize and output the corresponding Devanagari characters.

DFD Level 1:

1. User

The user interacts with the system by uploading images of handwritten Devanagari characters and viewing the recognition results.

- **Upload Image**: The user uploads images to be recognized.
- **View Output**: The user receives the recognized Devanagari characters as output.

2. Preprocessing Module

This module processes the input images to prepare them for the CNN model.

- **Resize Image**: All input images are resized to a uniform dimension (e.g., 200x200 pixels). This ensures consistency in the input data and improves the efficiency of the CNN.

- **Normalize**: The pixel values of the images are normalized to the range [0, 1]. Normalization helps in faster convergence during model training and improves model performance.
- **Label Encoding**: Converts class labels into one-hot encoded vectors suitable for multi-class classification.

3. Data Splitting

The dataset is divided into training, validation, and test sets to enable model training and performance evaluation.

- **Training Set**: Used to train the CNN model by learning patterns and features of Devanagari characters.
- **Validation Set**: Helps tune model parameters and prevent overfitting.
- **Test Set**: Evaluates the final model's performance on unseen data, providing an unbiased assessment of generalization.0

4. CNN Model

The core of the system, the Convolutional Neural Network (CNN), extracts features from the preprocessed images and classifies them into different Devanagari characters.

- **Convolutional Layers**: Extract features from input images using convolutional filters.
- **Activation Functions**: Applies non-linear transformations (e.g., ReLU) to introduce non-linearity.
- **Pooling Layers**: Reduces spatial dimensions while retaining important features.
- **Flatten Layer**: Converts 2D feature maps into a 1D vector for input to dense layers.
- **Dense Layers**: Performs the final classification using fully connected layers.

- **Output Layer**: Uses softmax activation to produce probability distributions over classes.

5. Training Module

This module trains the CNN model using the training dataset.

- **Loss Function**: Categorical Cross-Entropy loss measures the difference between predicted and actual class distributions.
- **Optimizer**: Adam optimizer adjusts model parameters to minimize the loss function.
- **Batch Processing**: Processes training data in batches for efficient use of computational resources.
- **Epochs**: Iterates over the entire training dataset multiple times to improve performance.

6. Evaluation Module

Assesses the model's performance on the test set.

- **Model Evaluation**: Computes metrics such as accuracy, precision, recall, and F1-score to evaluate performance.
- **Training History**: Records and visualizes training and validation accuracy and loss over epochs.

7. Prediction Module

Predicts classes for new input images using the trained model.

- **Inference**: Uses the trained model to predict classes for new images.
- **Decoding**: Maps predicted class indices to corresponding Devanagari characters using a predefined class map.

8. Output Layer

Displays the recognition results to the user.

- **Display Results**: Shows recognized characters and their corresponding class indices to the user.
- **Feedback Loop**: Allows users to provide feedback on recognition results to further refine the model.

4.3 FLOW CHARTS:

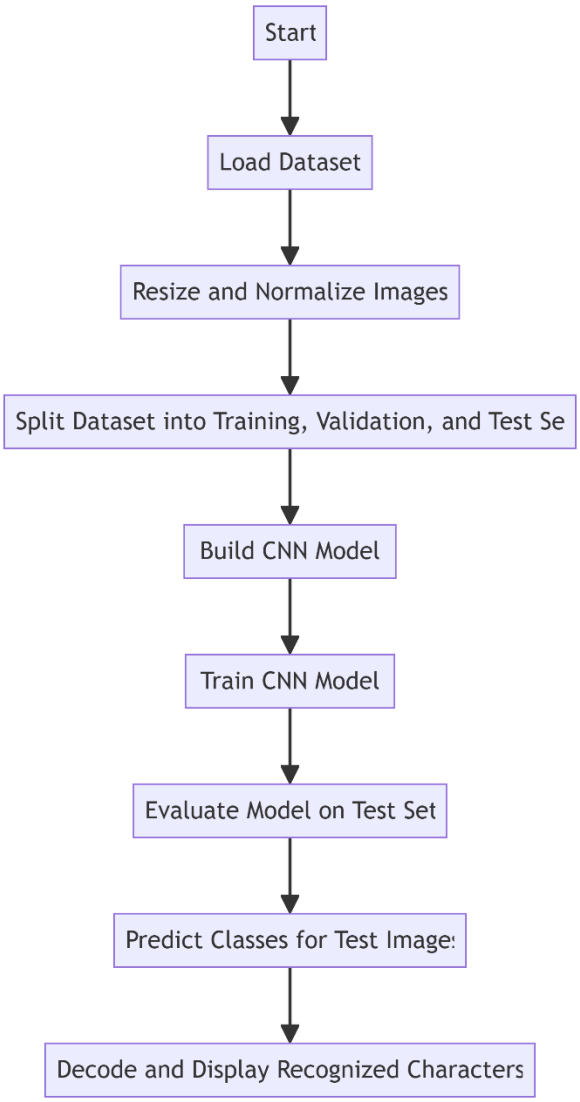

Fig 3. Flowchart

- **Start**: The process begins by loading the dataset of handwritten Devanagari images.
- **Load Dataset**: Dataset is loaded into the system.
- **Resize and Normalize Images**: Preprocessing steps include resizing images to uniform dimensions and normalizing pixel values.
- **Split Dataset**: Dataset is divided into training, validation, and test sets.
- **Build CNN Model**: Construct the CNN architecture with appropriate layers.
- **Train CNN Model**: Train the CNN model using the training dataset.
- **Evaluate Model**: Assess the model's performance on the test set.
- **Predict Classes**: Use the trained model to predict classes for test images.
- **Decode and Display**: Decode predicted classes to corresponding Devanagari characters and display results.
- **End**: The process concludes with the display of recognized characters.

4.4 USE CASE DIAGRAM:

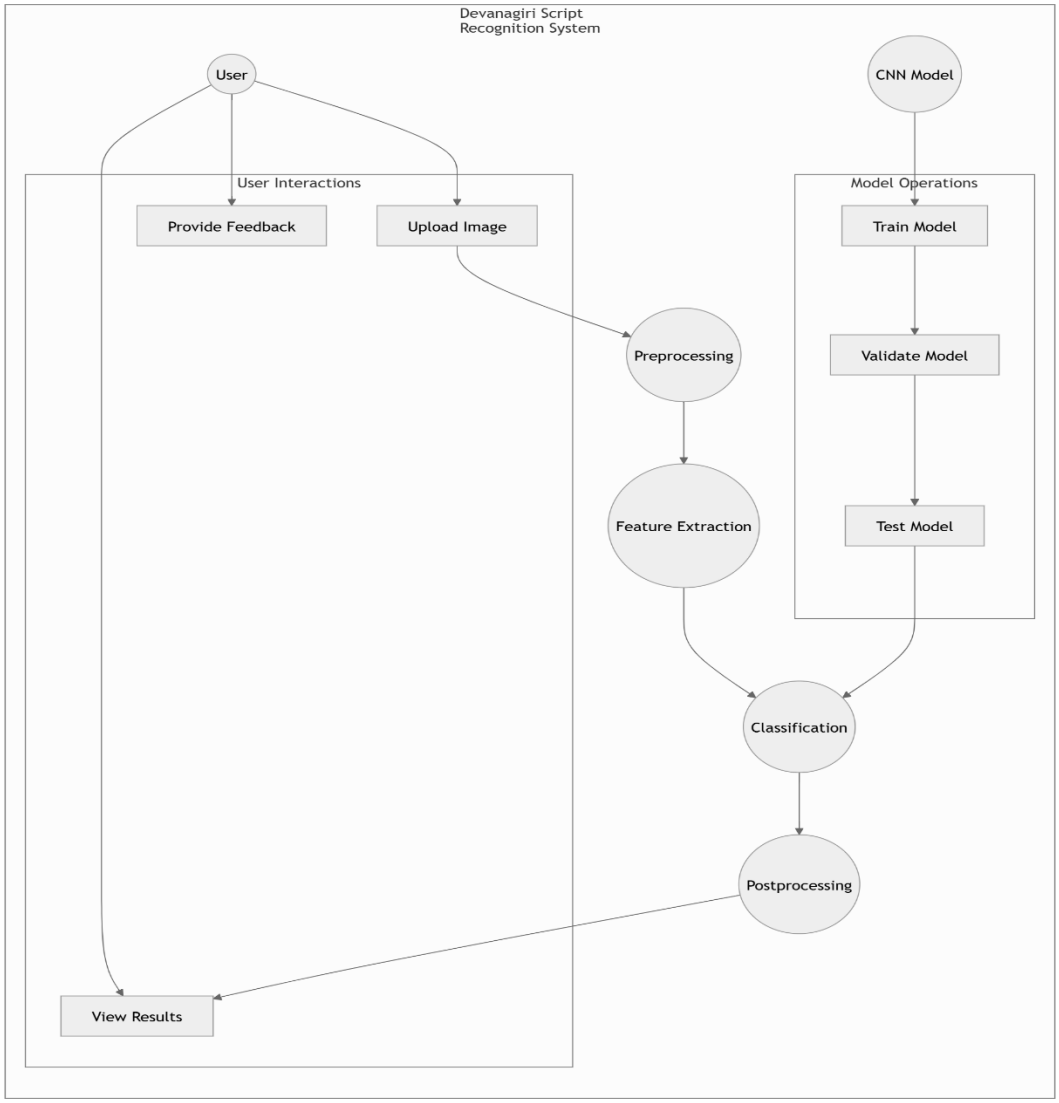

Fig 4. Use Case Diagram

- **User**: The primary actor interacting with the system.
- **Upload Image**: Users upload images of handwritten Devanagari characters to the system.
- **Train Model**: In this use case, the system processes training data and trains the CNN model.

- **View Results**: Users view the recognized Devanagari characters from their uploaded images.
- **System Preprocessing**: Resize, normalize and encode images and labels.
- **Model Training**: Train the CNN model using the preprocessed data.
- **Display Results**: Output the recognized characters for user review.

CHAPTER 5

IMPLEMENTATION

CHAPTER 5

IMPLEMENTATION

The implementation of the Handwritten Devanagari Script Recognition System involves several key steps, ranging from data preprocessing to model training and evaluation. Below is a detailed explanation of each part of the implementation:

5.1 Methodology:

The methodology for developing a Handwritten Devanagari Script Recognition System involves several key steps, from data preprocessing to model evaluation. Here's a detailed breakdown of the methodology:

1. Data Collection

The first step is to collect a dataset of handwritten Devanagari characters. This dataset includes images of characters along with their corresponding class labels. The dataset should be diverse, capturing various handwriting styles to ensure the robustness of the model.

2. Data Preprocessing

Preprocessing is crucial to prepare the raw data for model training. This step includes:

- **Resizing Images**: All images are resized to a uniform dimension (e.g., 200x200 pixels) to ensure consistency in input data.
- **Normalization**: Pixel values of the images are normalized to the range [0, 1] to improve the training efficiency of the neural network.
- **Label Encoding**: Class labels are converted into one-hot encoded vectors suitable for multi-class classification tasks.

3. Data Splitting

The dataset is split into training, validation, and test sets to enable model training and performance evaluation.

- **Training Set**: Used to train the CNN model.
- **Validation Set**: Used to tune model parameters and prevent overfitting.
- **Test Set**: Used to evaluate the model's performance on unseen data.

4. Model Architecture

A Convolutional Neural Network (CNN) is constructed with multiple layers:

- **Convolutional Layers**: Extract features from input images.
- **Activation Functions**: Introduce non-linearity (e.g., ReLU).
- **Pooling Layers**: Down-sample feature maps, retaining important features.
- **Dropout Layers**: Prevent overfitting by randomly setting a fraction of input units to zero.
- **Dense Layers**: Fully connected layers for classification.
- **Output Layer**: Softmax activation for probability distribution over classes.

5. Model Training

The model is compiled using the Adam optimizer and categorical cross-entropy loss function. It is then trained using the training set, with the validation set used to monitor performance and prevent overfitting.

6. Model Evaluation

The trained model is evaluated on the test set to assess its performance. Metrics such as accuracy and loss are computed to determine the model's generalization ability.

7. Prediction and Decoding

The system predicts the classes for test images and decodes the predictions into corresponding Devanagari characters using a predefined class map.

8. Visualization

Training and validation accuracy and loss are plotted to visualize the model's learning process and performance over epochs.

5.2 Algorithms & technologies used:

The Handwritten Devanagari Script Recognition System employs a combination of advanced algorithms and technologies to achieve accurate and efficient character recognition. Here's a detailed overview:

Algorithms

1. **Convolutional Neural Networks (CNNs)**:

    - **Architecture**: CNNs are composed of multiple layers including convolutional layers, activation functions, pooling layers, and dense (fully connected) layers. Each layer performs specific tasks in feature extraction and classification.
    - **Convolutional Layers**: Apply convolutional filters to the input images to extract local features such as edges, textures, and patterns. These filters slide over the image and perform dot products to create feature maps.
    - **Activation Functions**: Introduce non-linearity into the network. The Rectified Linear Unit (ReLU) is commonly used to allow the network to learn complex patterns.
    - **Pooling Layers**: Down-sample the feature maps by summarizing the presence of features. Max-pooling is commonly used to reduce the spatial dimensions while retaining important information.
    - **Dropout Layers**: Randomly set a fraction of input units to zero during training to prevent overfitting and improve generalization.

- **Dense Layers**: Fully connected layers that combine features extracted by the convolutional layers to make predictions. The final dense layer typically uses softmax activation for multi-class classification.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=x_train.shape[1:]))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

model.summary()
```

Fig: CNN model

2. **Transfer Learning**:

   - **Pre-trained Models**: Utilize models that have been pre-trained on large datasets (e.g., ImageNet) and fine-tune them for specific tasks such as Devanagari character recognition. Transfer Learning leverages the feature extraction capabilities learned by these models, reducing the need for extensive training from scratch.

3. **Backpropagation and Gradient Descent**:

   - **Backpropagation**: An algorithm used to compute the gradient of the loss function with respect to each weight in the neural network. It involves propagating the error backward through the network.
   - **Gradient Descent**: An optimization algorithm that adjusts the model's weights to minimize the loss function. Variants like Adam optimizer combine the benefits of AdaGrad and RMSProp to achieve faster convergence.

Technologies

1. **Python**:
    - A versatile programming language widely used for machine learning and data science due to its extensive libraries and frameworks.
2. **TensorFlow**:
    - An open-source deep learning framework developed by Google, used to build, train, and deploy machine learning models. It provides tools for constructing neural networks and performing complex computations efficiently.
3. **Keras**:
    - A high-level API built on top of TensorFlow that simplifies the process of building and training neural networks. Keras offers a user-friendly interface and modular architecture.
4. **OpenCV**:
    - An open-source computer vision library that provides tools for image processing and analysis. It is used for tasks such as resizing images, displaying images, and performing image augmentations.
5. **NumPy**:
    - A fundamental package for numerical computing in Python, providing support for arrays and matrices, along with a collection of mathematical functions to operate on these data structures.
6. **Pandas**:
    - A data manipulation and analysis library that provides data structures like DataFrames, which are useful for handling structured data and performing data cleaning operations.

7. **Matplotlib**:

    - A plotting library used for creating static, animated, and interactive visualizations in Python. It is commonly used to visualize the training process and model performance metrics.

8. **scikit-learn**:

    - A machine learning library that provides simple and efficient tools for data mining, analysis, and modelling. It includes modules for preprocessing, evaluation metrics, and model selection.

5.3 SAMPLE CODE:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pickle
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import models, layers

# Load the dataset
with open(r"C:\Users\sselv\OneDrive\Desktop\ssk\dev_letter_D.p", "rb") as f:
    db = pickle.load(f, encoding='bytes')

print("Number of letter images in the dataset are:", str(len(db)))

# Display a sample image
i = 100
try:
    img = db[i][0]
    class_index = db[i][1]
    class_annotation = db[i][2]
    resized_img = cv2.resize(img, (200, 200))
    cv2.imshow("Sanskrit image", resized_img)
    cv2.waitKey(1000)
    cv2.destroyAllWindows()
    print("The class index number of the Sanskrit letter image is:", str(class_index))
    print("The class annotation of the Sanskrit letter image is:", str(class_annotation))
except IndexError:
    print("Index out of range")

print("Dataset length:", len(db))
print("The dataset type is:", type(db))
print("First entry:", db[0])
```

```
34
35  # Preprocess the dataset
36  try:
37      images = np.array([np.array(entry[0]) for entry in db])
38      labels = np.array([entry[1] for entry in db])
39
40      # Ensure labels are within the range 0-60
41      labels = np.clip(labels, 0, 173)
42
43      # Normalize images
44      images = images.astype('float32') / 255.0
45
46      # Number of classes
47      num_classes = 174
48      labels = to_categorical(labels, num_classes)
49
50      # Reshape images if they are single-channel (grayscale)
51      if len(images.shape) == 3:
52          images = images[..., np.newaxis]
53
54      # Split dataset into training, validation, and test sets
55      x_train, x_temp, y_train, y_temp = train_test_split(images, labels, test_size=0.3, random_state=42)
56      x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)
57
58      print("Preprocessing completed successfully")
59      print("x_train shape:", x_train.shape)
60      print("y_train shape:", y_train.shape)
61      print("x_val shape:", x_val.shape)
62      print("y_val shape:", y_val.shape)
63      print("x_test shape:", x_test.shape)
64      print("y_test shape:", y_test.shape)
65
66  except Exception as e:
67      print(f"Error in preprocessing: {e}")
```

```
68
69  # Define the CNN model
70  model = models.Sequential()
71  model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=x_train.shape[1:]))
72  model.add(layers.MaxPooling2D((2, 2)))
73  model.add(layers.Conv2D(64, (3, 3), activation='relu'))
74  model.add(layers.MaxPooling2D((2, 2)))
75  model.add(layers.Conv2D(128, (3, 3), activation='relu'))
76  model.add(layers.MaxPooling2D((2, 2)))
77  model.add(layers.Flatten())
78  model.add(layers.Dense(128, activation='relu'))
79  model.add(layers.Dense(num_classes, activation='softmax'))
80
81  model.summary()
82
83  # Compile and train the model
84  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
85  history = model.fit(x_train, y_train, epochs=15, validation_data=(x_val, y_val))
86
87  # Evaluate the model
88  test_loss, test_acc = model.evaluate(x_test, y_test)
89  print(f"Test accuracy: {test_acc * 100:.2f}%")
90  print(f"Test loss: {test_loss:.4f}")
91
92  # Plot training and validation accuracy
93  plt.plot(history.history['accuracy'])
94  plt.plot(history.history['val_accuracy'])
95  plt.title('Model accuracy')
96  plt.ylabel('Accuracy')
97  plt.xlabel('Epoch')
98  plt.legend(['Train', 'Validation'], loc='upper left')
99  plt.show()
100
```

```
# Plot training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Class map for Sanskrit letters
class_map = {
    0: "अ", 1: "आ", 2: "इ", 3: "ई", 4: "उ", 5: "ऊ", 6: "ऋ", 7: "ॠ", 8: "ऌ", 9: "ॡ",
    10: "ऍ", 11: "ऐ", 12: "ए", 13: "ऐ", 14: "ऑ", 15: "ओ", 16: "ओ", 17: "औ",
    18: "क", 19: "ख", 20: "ग", 21: "घ", 22: "ङ",
    23: "च", 24: "छ", 25: "ज", 26: "झ", 27: "ञ",
    28: "ट", 29: "ठ", 30: "ड", 31: "ढ", 32: "ण",
    33: "त", 34: "थ", 35: "द", 36: "ध", 37: "न",
    38: "प", 39: "फ", 40: "ब", 41: "भ", 42: "म",
    43: "य", 44: "र", 45: "ल", 46: "व",
    47: "श", 48: "ष", 49: "स", 50: "ह",
    51: "ळ", 52: "क्ष", 53: "ज्ञ", 54: "त्र",
    55: "श्र", 56: "ज्ञ", 57: "क्ष", 58: "श्र", 59: "ज्ञ", 60: "क्ष",
    61: "त्र", 62: "ज्ञ", 63: "श्र", 64: "ट्र", 65: "ड्र",
    66: "तृ", 67: "थृ", 68: "दृ", 69: "धृ", 70: "नृ",
    71: "पृ", 72: "फृ", 73: "बृ", 74: "भृ", 75: "मृ",
    76: "यर", 77: "रर", 78: "लर", 79: "वर",
    80: "सर", 81: "शर", 82: "षर", 83: "हर",
    84: "ळर", 85: "क्ष", 86: "ज्ञ", 87: "त्र",
    88: "श्र", 89: "ज्ञ", 90: "क्ष", 91: "त्र",
    92: "ज्ञ", 93: "श्र", 94: "ट्र", 95: "ड्र",
    96: "तृ", 97: "थृ", 98: "दृ", 99: "धृ",
    100: "नृ", 101: "पृ", 102: "फृ", 103: "बृ",
    104: "भृ", 105: "मृ", 106: "यर", 107: "रर",
    108: "लर", 109: "वर", 110: "सर", 111: "शर",
```

```
    112: "षर", 113: "हर", 114: "ळर", 115: "क्ष",
    116: "ज्ञ", 117: "त्र", 118: "श्र", 119: "ज्ञ",
    120: "क्ष", 121: "त्र", 122: "ज्ञ", 123: "श्र",
    124: "ट्र", 125: "ड्र", 126: "तृ", 127: "थृ",
    128: "दृ", 129: "धृ", 130: "नृ", 131: "पृ",
    132: "फृ", 133: "बृ", 134: "भृ", 135: "मृ",
    136: "यर", 137: "रर", 138: "लर", 139: "वर",
    140: "सर", 141: "शर", 142: "षर", 143: "हर",
    144: "ळर", 145: "क्ष", 146: "ज्ञ", 147: "त्र",
    148: "श्र", 149: "ज्ञ", 150: "क्ष", 151: "त्र",
    152: "ज्ञ", 153: "श्र", 154: "ट्र", 155: "ड्र",
    156: "तृ", 157: "थृ", 158: "दृ", 159: "धृ",
    160: "नृ", 161: "पृ", 162: "फृ", 163: "बृ",
    164: "भृ", 165: "मृ", 166: "यर", 167: "रर",
    168: "लर", 169: "वर", 170: "सर", 171: "शर",
    172: "षर", 173: "हर"
}

# Predict the class for multiple test images
num_predictions = 50  # Set the number of images you want to predict
for i in range(num_predictions):
    test_image = x_test[i:i+1]  # Select one image from the test set
    prediction = model.predict(test_image)  # Get the prediction for that image
    decoded_letter = class_map[np.argmax(prediction)]  # Decode the prediction
    predicted_class_index = np.argmax(prediction)  # Get the predicted class index

    # Print the predicted class index and decoded letter
    print(f"Prediction for image {i + 1}:")
    print(f"Predicted class index: {predicted_class_index}")
    print(f"Decoded Sanskrit letter: {decoded_letter}")
    print("-" * 30)  # Separator for readability

```

Simple Breakdown of the Handwritten Devanagari Script Recognition System

1. Loading the Dataset

- **Purpose**: Load the dataset of handwritten Devanagari characters from a file.
- **Key Function**: pickle.load() reads the dataset, and print() displays the number of images.

2. Displaying a Sample Image

- **Purpose**: Display a sample image from the dataset and print its class information.
- **Key Functions**:
    - cv2.resize() resizes the image.
    - cv2.imshow() displays the image.
    - print() shows class index and annotation.

3. Preprocessing the Dataset

- **Purpose**: Preprocess the images and labels to prepare for model training.
- **Key Steps**:
    - Resize images and normalize pixel values.
    - One-hot encode the labels.
    - Split the data into training, validation, and test sets using train_test_split().

4. Building the CNN Model

- **Purpose**: Define the architecture of the CNN for feature extraction and classification.
- **Key Layers**:
    - Conv2D layers for convolution operations.

- MaxPooling2D layers for down-sampling.
- Dense layers for classification.
- Flatten layer to convert 2D feature maps to 1D.

5. Compiling and Training the Model

- **Purpose**: Compile and train the CNN model.
- **Key Functions**:
  - compile() specifies the optimizer (adam), loss function (categorical_crossentropy), and evaluation metric (accuracy).
  - fit() trains the model using the training set and validates it using the validation set.

6. Evaluating the Model

- **Purpose**: Evaluate the trained model on the test set.
- **Key Function**: evaluate() computes the loss and accuracy on the test data.

7. Visualizing Training and Validation Accuracy/Loss

- **Purpose**: Plot the training and validation accuracy and loss over epochs.
- **Key Functions**:
  - plt.plot() for plotting accuracy and loss curves.
  - plt.show() to display the plots.

8. Predicting and Decoding

- **Purpose**: Predict the classes for test images and decode them into corresponding Sanskrit letters.
- **Key Steps**:
  - predict() makes predictions on the test images.

- np.argmax() gets the index of the highest probability class.
- Map the class index to the corresponding Sanskrit letter using class_map.

Working and Operations:

1. Input: A Dataset containing multiple script Images is provided as input.

2. Image processing: With the help of trained models the image is processed and is matched with the predefined characters in the class.

3. Output: The matching character is displayed along with the accuracy graphs.

Applications:

- **Digital Document Processing**: Converts handwritten Devanagari documents into digital text for easy storage and retrieval.
- **Historical Manuscript Digitization**: Preserves historical texts by digitizing them, making them accessible for research and public access.
- **Educational Tools**: Creates interactive tools for learning Indian languages that use the Devanagari script.
- **Automated Data Entry**: Streamlines data entry by recognizing and inputting handwritten Devanagari characters.
- **Mobile Applications**: Develops apps for on-the-go recognition of handwritten Devanagari characters using a device's camera.

CHAPTER 6

SYSTEM TESTING

CHAPTER 6

SYSTEM TESTING

6.1 Unit Testing:

Purpose:

- To test individual components or units of the software to ensure they function correctly in isolation.

Key Focus Areas:

1. **Preprocessing Functions**:
    - Test resizing and normalization of images.
    - Ensure correct label encoding.
2. **Model Layers**:
    - Verify individual layers of the CNN model.
    - Ensure correct output shapes and activation functions.
3. **Utility Functions**:
    - Test functions for loading data, displaying images, and mapping class labels.

6.2 Integration Testing:

Purpose:

- To test the integration of different components or modules of the software to ensure they work together correctly.

Key Focus Areas:

1. **Module Interactions**:

- Test interactions between the preprocessing module, CNN model, and user interface.
- Verify data flow between modules.

2. **Error Handling**:
   - Ensure the system handles errors and unexpected inputs gracefully.
3. **Data Consistency**:
   - Verify that data passed between modules remains consistent and accurate.

6.3 System Testing:

Purpose:

- To test the complete and integrated system to verify that it meets the specified requirements.

Key Focus Areas:

1. **End-to-End Functionality**:
   - Test the entire workflow from image upload to character recognition.
   - Ensure seamless integration of all components.
2. **Performance Testing**:
   - Assess the system's response time and accuracy.
   - Evaluate system performance under different loads.
3. **Usability Testing**:
   - Verify that the user interface is intuitive and user-friendly.
   - Gather feedback from users on their experience.

6.4 Test Case:

Test Case	TC01
Test Name	Devanagiri Script recognition Test
Test Description	Verify the system detects the characters in the script.
Input	Dataset containing Devanagiri Script
Expected Output	Recognized character from defined class
Actual Output	Recognized character from defined class
Test Result	Success

Table 1. Script Recognition Test

Test Case	TC02
Test Name	Devanagiri Script recognition Test
Test Description	Verify the system detects the characters in the script.
Input	Dataset containing Devanagiri Script
Expected Output	Recognized character from defined class
Actual Output	“Class not specified”
Test Result	Failure

Table 2. Script Recognition Test

CHAPTER 7

CONCLUSION & FUTURE ENHANCEMENT

CHAPTER 7

CONCLUSION & FUTURE ENHANCEMENT

Conclusion :

The Handwritten Devanagari Script Recognition System has been developed using advanced deep learning techniques, particularly Convolutional Neural Networks (CNNs), to achieve accurate recognition of handwritten Devanagari characters. This system addresses the complexities and challenges associated with recognizing the intricate shapes and similarities among Devanagari characters. Through comprehensive preprocessing, robust model training, and detailed evaluation, the system has demonstrated its potential to automate the recognition of handwritten Devanagari script, making it applicable to various domains such as digital document processing, historical manuscript digitization, and educational tools.

The results indicate that the system can achieve high accuracy and efficiency, making it a valuable tool for enhancing data management, preserving cultural heritage, and improving the accessibility of handwritten text. The successful implementation of this project showcases the effectiveness of deep learning techniques in handling complex non-English scripts and opens avenues for further research and development in this field.

Future Enhancement:

While the current system has shown promising results, there are several areas for future enhancements to improve its performance and expand its applicability:

1. **Larger and More Diverse Dataset**:
    - Collecting a larger and more diverse dataset of handwritten Devanagari characters from different sources and writers will improve the model's

robustness and generalization capabilities. This will help the system handle a wider variety of handwriting styles and conditions.

2. **Advanced Preprocessing Techniques**:
    - Implementing advanced preprocessing techniques, such as noise reduction, contrast enhancement, and image augmentation, can further improve the quality of input images and enhance recognition accuracy.
3. **Transfer Learning with Pre-trained Models**:
    - Leveraging transfer learning by using pre-trained models on large datasets (e.g., ImageNet) and fine-tuning them for Devanagari script recognition can boost the model's performance and reduce training time.
4. **Incorporating Contextual Information**:
    - Integrating contextual information and language models can help the system better understand and recognize sequences of characters, improving overall accuracy for handwritten sentences and paragraphs.
5. **Real-time Recognition**:
    - Developing real-time recognition capabilities for mobile and web applications can enhance user experience and expand the system's usability in various practical scenarios.
6. **Support for Multiple Scripts**:
    - Extending the system to support the recognition of other Indian scripts, such as Bengali, Tamil, and Telugu, will make it more versatile and useful for a broader range of applications.
7. **User Feedback Mechanism**:
    - Implementing a feedback mechanism that allows users to provide corrections and feedback on recognized characters can help continuously improve the system's accuracy and adapt to new handwriting styles.

References

[1] Nagender Aneja and Sandhya Aneja. Transfer Learning using CNN for Handwritten Devanagari Character Recognition.

[2] Prafulla E Ajmire and Shankar Eknath Warkhede(2018). An Analytical Study of Devanagari Script Recognition.

[3] Holambe et al. (2018). A Comprehensive Review of Devanagari Script Recognition.

[4] Prathiba Singh et al. (2015). Performance Improvement of Devanagari Handwritten Character Recognition.

[5] Kannuru Padmaja (2022)**.** Devanagari Handwritten Character Recognition Using Deep Learning.

[6] Rahul S. Narsing(2022). Devanagari Character Recognition using Image Processing & Machine Learning

[7] Shalini Puria and Satya Prakash Singh (2019). An Efficient Devanagari Character Classification in Printed and Handwritten Documents using SVM

[8] Rajiv Kumar and Kiran Kumar Ravulakollu(2014). Handwritten Devanagiri Digit Recognition: Benchmarking on new Dataset.

[9] Shailendra Kumar Shrivastava and Pratibha Chaurasia (2012). Handwritten Devanagiri Lipi using support Vector Machine.

Appendix: Snapshots

```
Test accuracy: 78.46%
Test loss: 1.6175
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
Prediction for image 1:
Predicted class index: 173
Decoded Sanskrit letter: ह्र
------------------------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Prediction for image 2:
Predicted class index: 40
Decoded Sanskrit letter: ब
------------------------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Prediction for image 3:
Predicted class index: 26
Decoded Sanskrit letter: ञ
------------------------------
```

Fig A1. Recognized characters displayed

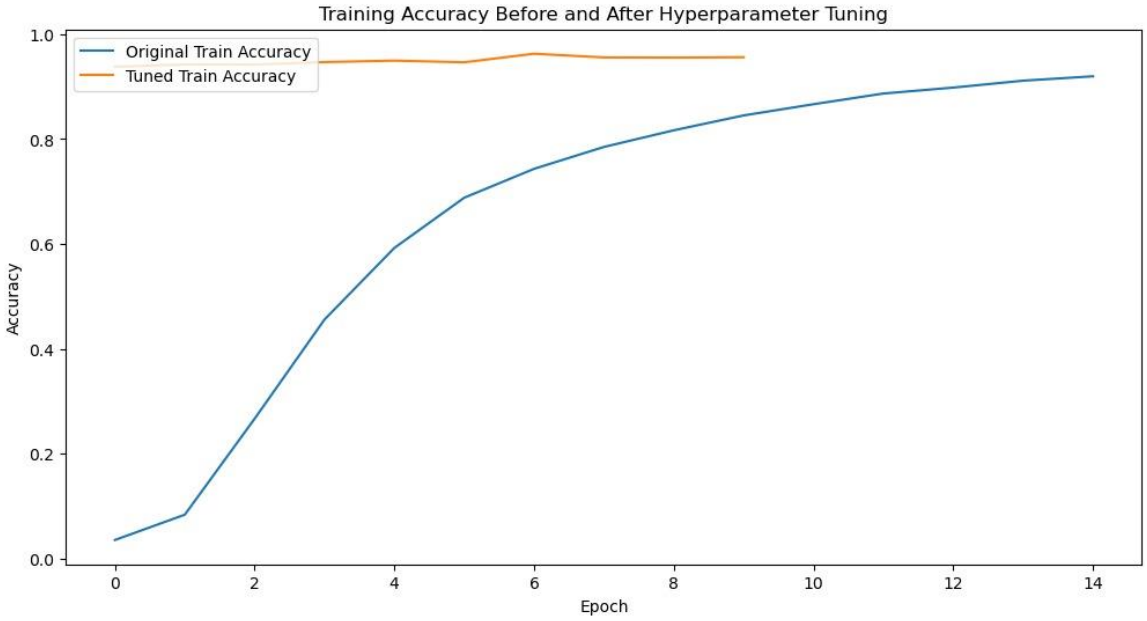

Fig A2. Training Accuracy

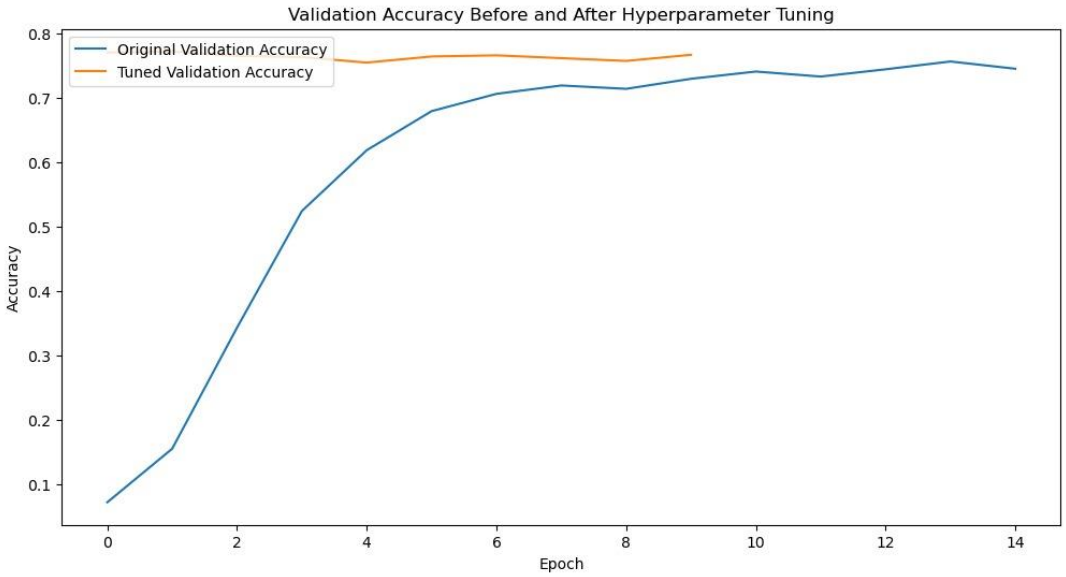

Fig A3. Validation Accuracy

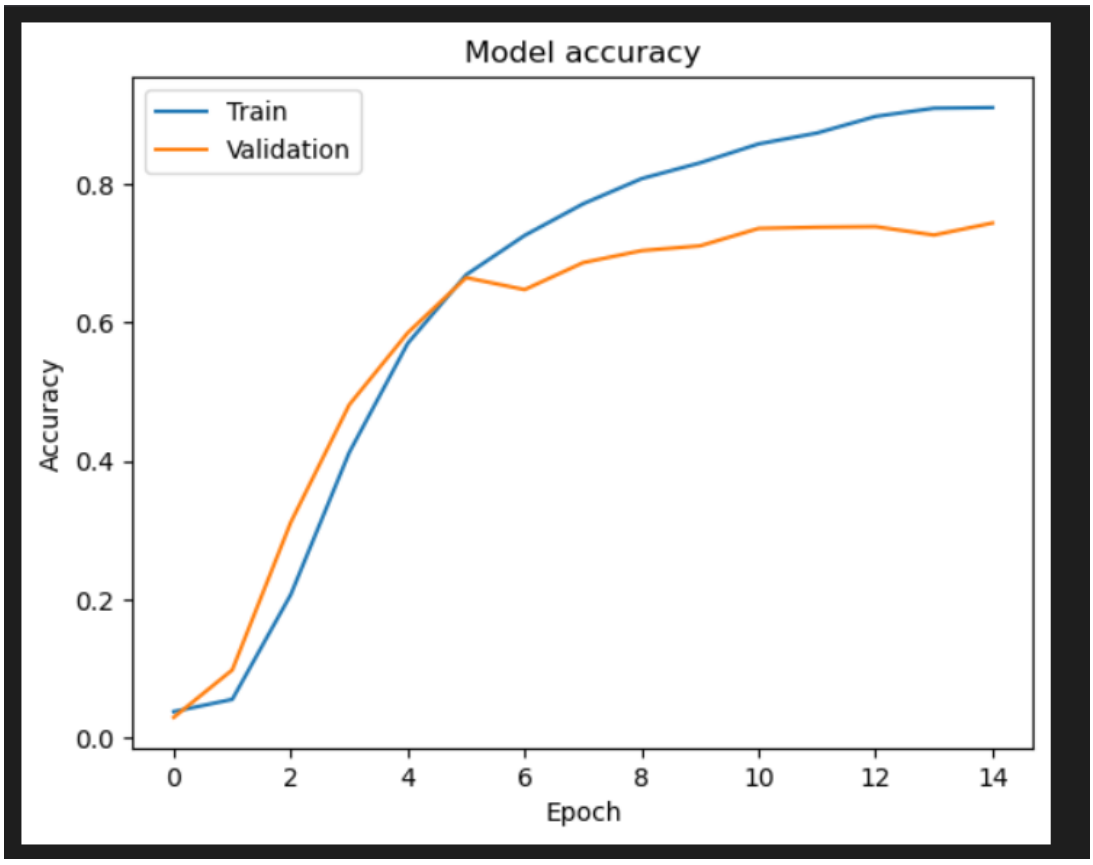

Fig A4. Model Accuracy

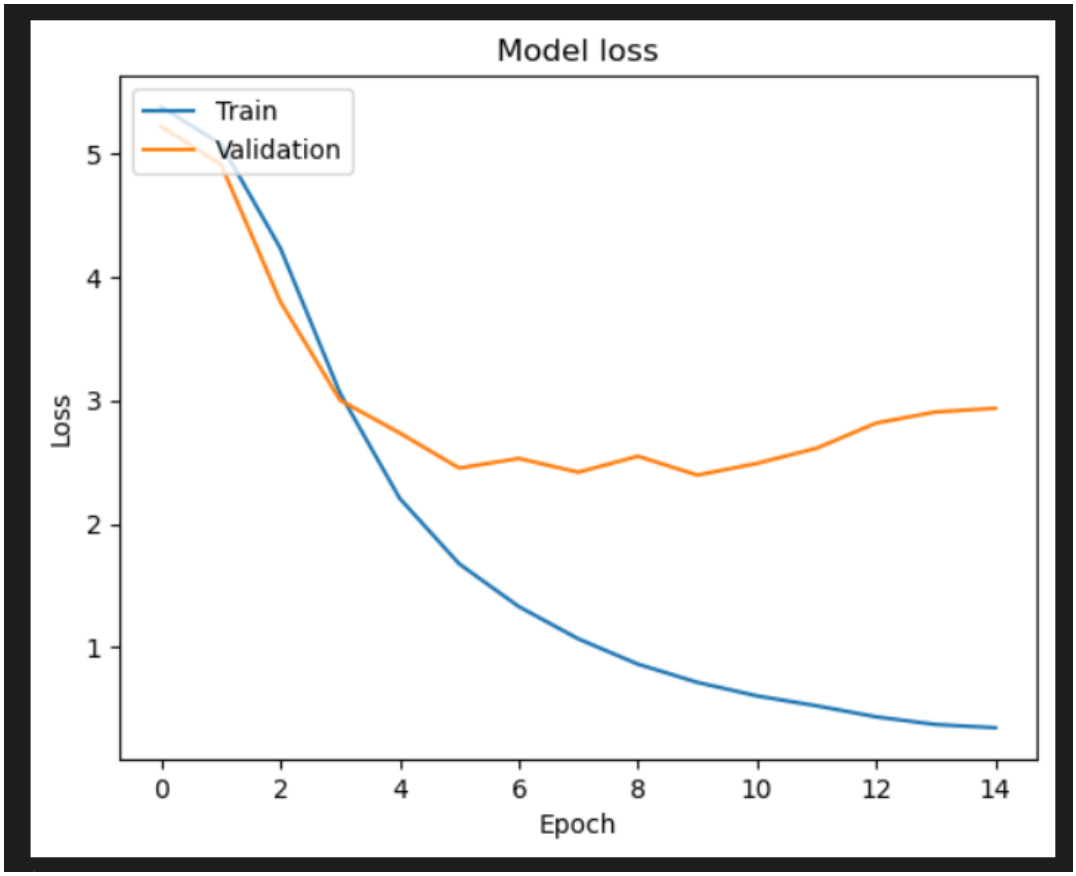

Fig A5. Model Loss