# DESIGN OF ODIA HANDWRITTEN ALPHABET RECOGNITION SYSTEM USING CONVOLUTIONAL NEURAL NETWORK

#### **A project submitted in partial fulfilment for the requirement of the degree of Master in Computer Application**

##### **Submitted by:**

**NAME:** Disha Ghosh

(**Registration No.:** 2306151031)

**Under the Supervision of**

**Dr. Manas Ranjan Senapati**

Associate Professor

Department of Computer Science and Engineering,

Veer Surendra Sai University of Technology, Burla



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,**

**VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY,**

**BURLA, SAMBALPUR – 768018**

**[2024-2025]**

****

**VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY**

**BURLA, ODISHA**

**Declaration**

I, Disha Ghosh, hereby declare that the project report titled Design of Odia Handwritten Alphabet Recognition System using Convolutional Neural Networkis an original work completed solely by me and represents my own ideas. I affirm that where the ideas or words of others have been included, I have accurately cited and referenced these sources in accordance with academic standards. I have adhered to all principles of academic integrity and have not misrepresented, fabricated, or falsified any data, facts, or sources in this submission. I understand that any breach of the above principles may result in disciplinary action by the university, and potentially legal repercussions should any source be improperly cited or permissions overlooked.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[Disha Ghosh]

****

**Certificate of Examination**

This is to certify that the project report entitled **“Design of** **Odia Handwritten Alphabet Recognition System using Convolutional Neural Network"** submitted by **Disha Ghosh** (**2306151031**) of Master of Computer Application (MCA), Veer Surendra Sai University of Technology, Odisha has been examined by us. We are satisfied with the quality and correctness of the work.

HOP, PG & Ph.D. HOD,

Department of CSE Department of CSE

VSSUT, Burla VSSUT, Burla

****

**Certificate of Approval**

This is to certify that the Project Report entitled **"Design of Odia Handwritten Alphabet Recognition System using Convolutional Neural Network"** is being submitted by **Disha Ghosh** (**Regd. No:** **2306151031**) of the Department of Computer Science and Engineering, Veer Surendra Sai University of Technology, Burla, in partial fulfilment of the requirement for the degree of Master of Computer Application (MCA) during the academic year 2024-2025. It is an original work carried out by her and has been examined by us.

**External Examiner Supervisor**

# Acknowledgement

I would like to thank my supervisor Dr. Manas Ranjan Senapati for invaluable guidance, constant assistance, and unwavering support throughout the course of this project. Her constructive suggestions and insights have been instrumental in the completion and enhancement of this project.

I would also like to extend my thanks to the authors and researchers whose work served as a valuable resource, guiding me through the technical aspects and helping me refine the content. Additionally, I am grateful to the staff members of the Department of Computer Science and Engineering, who provided assistance, both directly and indirectly, in various stages of this work.

Finally, I am thankful to my friends for their moral and material support throughout this project, which motivated me to complete and present it successfully.

Date: Full Signature of the candidate

Place: VSSUT, Burla

# Abstract

Odia is one of the famous languages spoken in India. It is the official language of Odisha and its speaker make 3.10% of total population of India. Therefore, a system needs to be developed for the recognition of Odia characters that would help many illiterate and physically handicapped people. Each character in Odia comprises of more than one connected symbols.

Character recognition is one of the difficult problems of Machine learning. The recognition rate for the offline recognition system is highly affected by various writing style, size and shape. In this paper work, we have developed a two phase system for the recognition of Odia alphabets. In the first phase, backend for the mentioned system is developed. This includes the development of CNN model for the classification of Odia alphabets in python environment and training the model for deployment. In the second phase, we have developed a front end for taking user input. This includes deployment of trained model and designing a website. The front end is built using streamlit. Our model exploits the inherit characteristics of Odia character images. We have also developed a VSSUT Odia handwritten character dataset which comprises of 3576 images belonging to 37 classes. The propose convolution neural network has given us an accuracy of 79.44%.

**Keywords:**

* Streamlit
* Convolutional Neural Networks (CNN)
* Deep Learning
* TensorFlow / Keras

List of Figures

|  |  |  |
| --- | --- | --- |
| **Figure/Table No.** | **Figure/Table Caption** | **Page No.** |
| Fig 3.1 | Input (5 x 5) Data and (3 x 3) Kernel | 5 |
| Fig 3.2 | Conversion of Input x filter into Feature Map | 6 |
| Fig 3.3 | Stacking of Feature Maps | 6 |
| Fig 3.4 | Stride 1 with Padding and the Feature Map | 7 |
| Fig 3.5 | Feature Map with a 2 x 2 Max pool | 9 |
| Fig 3.6 | Pooling Operation on Feature Map | 9 |
| Fig 3.7 | Fully connected Layer | 10 |
| Fig 4.1 | Odia Vowels and Consonants | 13 |
| Fig 6.1 | Architecture of the Model | 20 |
| Fig 6.2 | Homepage of the Application | 22 |
| Fig 6.3 | Result page of the Application | 23 |
| Fig 6.4 | User Interface code | 23 |
| Fig 7.1 | Plot showing loss vs no. of epochs | 25 |
| Fig 7.2 | Plot showing accuracy vs no. of epochs | 26 |
| Fig 7.3 | **Model Performance Metrics** | 29 |
|  |  |  |
|  |  |  |
| **List of Tables** |  |  |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **Table No.** | **Table Caption** |  |  |  |  | **Page No.** | | Table 4.1 | Train- Test Split of dataset | 22 |  |  |  | 1515 | | Table 5.1 | Comparison of various Classifiers with their Accuracy |  |  |  |  | 29 | | Table 5.2 | Comparison of various Classifiers with their Accuracy |  |  |  |  | 30 | | | **Page No**  14  30  31 |  |

Contents

Title Page i

Declaration ii

[Certificate of Examination iii](#_Toc152622876)

[Certificate of Approval iv](#_Toc152622877)

[Acknowledgements v](#_Toc152622877)

[Abstract vi](#_Toc152622877)

List of Figures ………………………………………………………………………………...…vii

List of Tables ……………………………………………………………………..………..…..viii

Contents……………………………………………………………………………………..........ix

[1. INTRODUCTION 1](#_Toc152622879)-2

[2. LITERATURE REVIEW 3-4](#_Toc152622880)

[3. Overview of CNN 5-](#_Toc152622883)12

[4. Data Description 13-14](#_Toc152622888)

5.Tools and Technology……………………………………………………………………………………………………….15-16

6.Purposed Methodology……………………………………………………………………………………………………17-23

6.1 Back End Phase……………………………………………………………………….17-21

6.2 Front End Phase……………………………………………………………………………………………………….22-23

[7. Result and Discussion 24-31](#_Toc152622889)

8.Future Scope……………………………………………………………………………………………………………………32-34

9.Conclusion…………………………………………………………………………………………………………………………….35

References 36-39

## Chapter 1

**Introduction**

Handwritten character recognition has gained interest of research in the field of pattern recognition. An immense effort has been spent on developing automatic document analysis and processing systems for the correct interpretation of characters as it has wide commercial applications. Character recognition of various scripts written in English, French, Dutch and many more languages have been carried out by different researchers. Character recognition process of handwritten characters is harder than the recognition of printed characters. In India, OCR is used for the character recognition and is the active area of research. It uses the conventional method of document image analysis where inputs are converted into digital text format. Odia is an official language of state Odisha. Odia is the sixth language of India to get classical language status. In this paper, we have attempted to produce a Machine Learning model that uses Convolutional Neural Networks for the classification of Odia characters. While developing the model we faced lot of difficulties due to the presence of similar orientation in shape and size. We also faced difficulties due to various writing skills of different users. Along with these difficulties there are many more which has made the evolution of an accurate handwritten character recognition system is still considered as on open research problem. The reason for using CNN is that in the field of Image processing, Neural Networks is one of the essential methods for classifying images. It solves many issues of handwritten character recognition to achieve a good recognition rate. A high recognition rate can be achieved by having a good feature extraction technique along with an eminent classifier. In recent times, CNN based models have got more attention because these models have achieved tremendous success in several fields like in image Net (Krizhev sky et al. 2012), Object Detection (He et al. 2015), Facial expression Recognition (Zhao et al. 2015), etc. Many researchers [1-3] from IIT Bhubaneswar, NIT Rourkela, Ravenshaw University, Utkal University and IIIT Bhubaneswar took interest and did pioneering work in the field of Odia Character and numerals recognition. They have used dataset from ISI, Kolkata and NIT, RKL. Sufian et al. [4] in the year 2020 produced a CNN based model named BDNet for the recognition of Bengal characters. They have also proposed a new dataset of 1000 images of Bengali handwritten numerals. These works show that automation of interpreting Odia scripts (Characters and Numerals) has enormous significance in social and commercial fields.

The rest of the paper is divided into six sections. Section 2 describes the previous works done on Odia handwritten alphabet recognition. The motivation for proposing a model and various developed techniques related to classification of Odia characters is discussed in this section. The concepts related to convolutional neural network and its internals are discussed in Section 3. The dataset developed for our proposed model is described the Section 4. Section 5 explains the process of developing the Odia handwritten alphabet recognition. This includes the development of dataset, classification model and user interface for taking input. The result obtained of classification and hyper parameter tuning along with recognition rate is reported in Section 6. Section 7 provides the ending of the report with a conclusion.

## Chapter 2

**Literature Review**

The recognition of Odia characters is a difficult task as it contains compound and similar characters. These compound characters are made up of amalgamation of basic Odia characters leading to confusion in recognition. The intrinsic cursive nature of Odia characters make them look alike which forms the major challenge for recognition. This necessitates the study of research contributions relating to Odia handwritten alphabet recognition.

Sethy and Patra [5] developed offline Odia handwritten numeral recognizer using neural networks in the year 2016. There work was based on Binarization and Discrete Cosine Transform (DCT) scheme. There extensive simulation results in recognition rate of 80.2% for Binarization and 90% for DCT. In the same year, Rushiraj et al. [6] presented character recognition of handwritten Odia scripts. They used Euclidean Distance method that used 48 geometric features to classify 36 Odia characters.  The overall recognition rate is 87.6%. Sethy et al. [7] proposed automation of handwritten character recognition using a set of symmetry axes in the year 2017. They used standard database of NIT RKL Odia handwritten character and ISI Kolkata Odia numeral database. They used J48 classifier that gave 96.2% accuracy upon Odia numeral database and 95.6% accuracy upon Odia character database.

Pal et al. [8] used curvature feature of Odia characters to develop a classifier in the year 2007. This classifier recognized the Odia characters at a rate of 94.6%. Bhowmik et al. [9] proposed a novel hidden Markov model (HMM) for classification of handwritten Odia numerals in the year 2006. They have constructed one HMM for each numeral for training the dataset. There proposed model figured class restrictive likelihood for each HMM to recognize an unknown numeral image. They tested there classification scheme on an enormous transcribed Odia numeral database which came up with a recognition rate of 95.89% and 90.50% for training and test sets respectively. Mishra et al. [10] proposed two recognizers for Odia handwriting in the year 2013. There recognizer was based on Discrete Cosine Transformation (DCT) and Discrete Wavelet Transformation (DWT). There proposed paper also compared pros and cons of the two transformation schemes.  The two recognizers exploit the inherent characteristics of the Odia numeral images to produce a vector that is given input to a Back Propagation Neural Network (BPNN). The general precision rate got in there experimentation on manually written numerals utilizing DCT and DWT features are seen as 92% and 87.50% respectively. Pujari and Majhi [11] proposed an ensemble model for the recognition of Odia handwritten character in the year 2015. This ensemble model was built using four base classifier: Artificial Neural Network (ANN), Support Vector Machine (SVM), C5.0 Decision Tree and Discriminant Analysis (DA). They performed simulation using a combination of gradient and curvature based features extracted from the numeral dataset. They also performed feature reduction using Principal Component Analysis before simulation. The classification results obtained shows that the ensemble of SVM + DA and SVM + C5.0 + DA performed best with 97.5% accuracy.  Same year Ray et al. [12] proposed a Deep Bidirectional Long Short Term Memory (LSTM) based Recurrent Neural Network architecture for text recognition. The results of Deep Neural Networks for isolated numeral recognition and improved speech recognition using Deep BLSTM based approaches motivated them to develop this architecture which uses Connectionist Temporal Classification for training to recognize the labels of united words of printed Odia text.

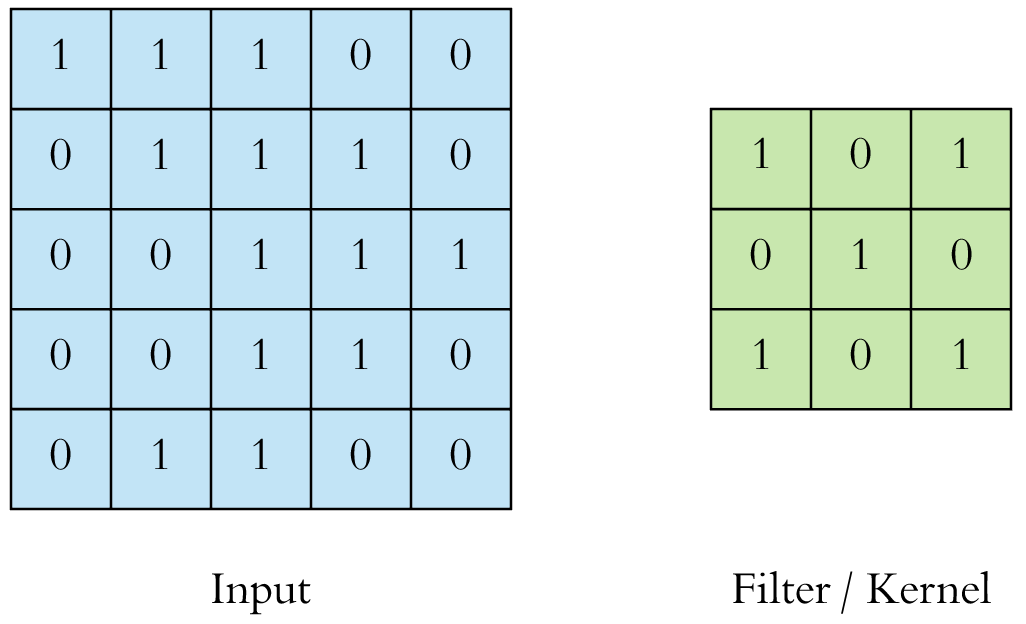
## Chapter 3

**Overview of CNN**

CNN is a specialized kind of neural network for processing data that has a known, grid-like topology, such as time- series (1D grid), image data (2D grid), etc. It is a supervised deep learning algorithm; it is used in various fields like speech recognition, image retrieval and face recognition.

**1. Convolution Layer**

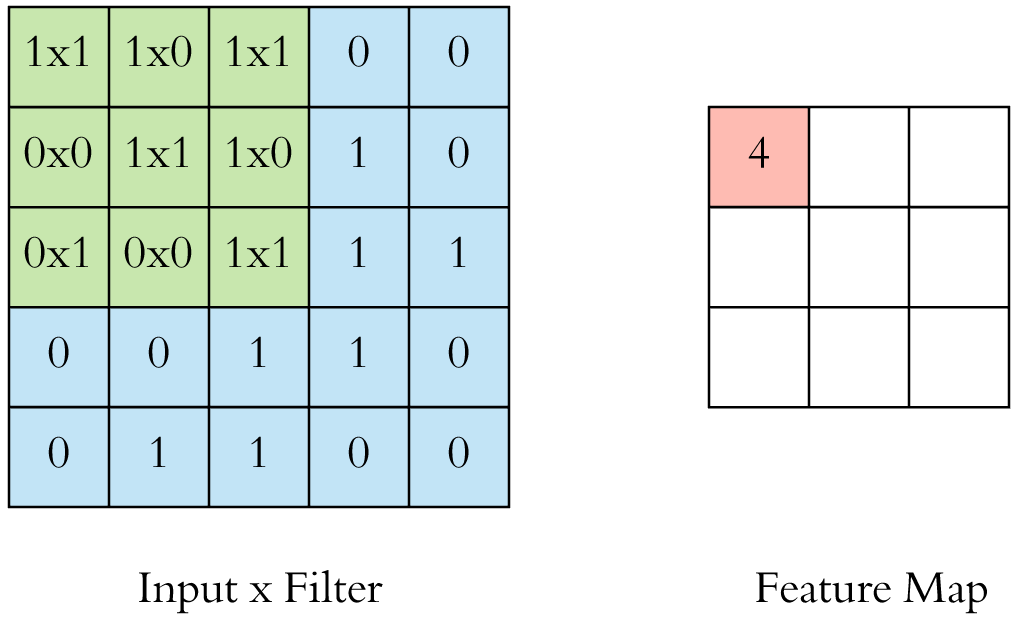
The main building block of CNN is the Convolutional layer. The primary focus of this layer is to extract high level spatial features. The convolution is applied on the input data using a convolutionfilter or kernel to produce a featuremap.



Input (5 x 5) Data and (3 x 3) Kernel

Fig 3.1

The convolution operation is performed by sliding the filter over the input image. At every cell of the matrix, matrix multiplication is performed and the result goes into the feature map and this is continued to aggregate the convolution results in the feature map.

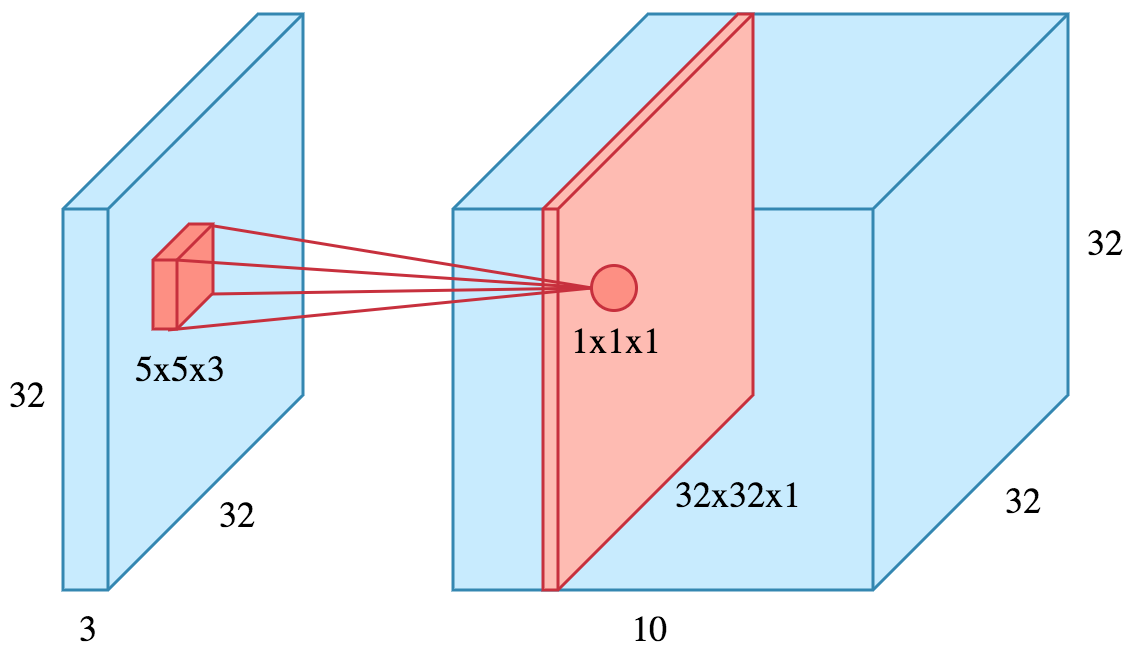


Conversion of Input x filter into Feature Map

Fig 3.2

An image is represented by a 3D matrix with height, width and depth, where depth corresponds to color channels (RGB).

More often, multiple convolutions are applied on an input, each having different filter and which results a distinct feature map. Finally, all these feature maps are stacked together which becomes the final output of the convolution layer.



Stacking of Feature Maps

Fig 3.3

**2. Non- Linearity or Activation Layer**

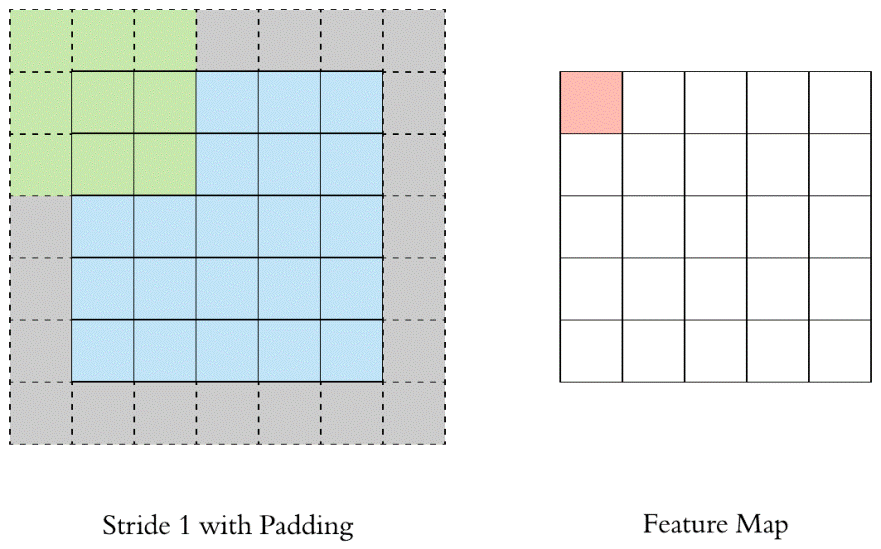
The result of the convolution operation is passed through an activation function. The Activation functions are used so that the values in the final feature maps are not actually the sums, but some non-linearity is function applied to them. One such activation functions are Relu (Rectified Linear Unit). It introduces non-linearity to our ConvNet. Other non- linear functions such as tanh or sigmoid can also be used instead of ReLU, but ReLU has been found to perform better in most situations

**3. Stride and Padding**

Stride specifies how much we move the convolution filter at each step and by default the value is 1.The stride can be manipulated as per the need; bigger strides will result less overlap between the receptive fields. This also makes the resulting feature map smaller since some of the potential locations are skipped.

In order to maintain same dimensionality, padding is used which surrounds the input with zeroes.

Padding is commonly used in CNN to preserve the size of the feature maps; otherwise they would shrink at each layer, which is not desirable.



Stride 1 with Padding and the Feature Map

Fig 3.4

**4. Pooling**

After a convolution operation, the pooling layer is applied so that there is reduction in dimensionality. Itreduces the number of parameters, which both shortens the training time and combats overfitting. Pooling layers down sample each feature map independently, reducing the height and width, keeping the depth intact.In all cases, pooling helps to make the representation become approximately invariant to small translations of the input.The local translation can be a very useful property in order to find some specific features.

There are basically three types of Pooling:-

• Max Pooling (works better)

• Average Pooling

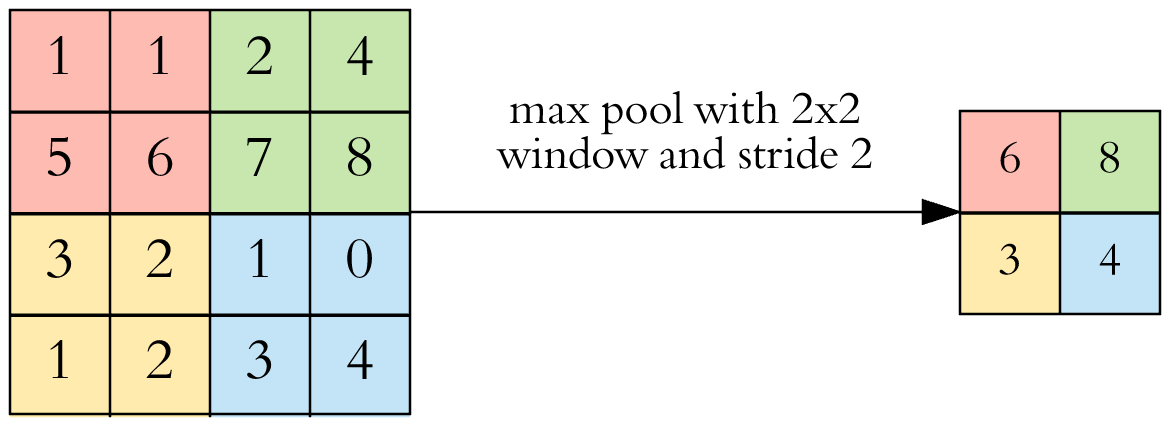
• Sum Pooling

The most common type of pooling is maxpooling which just takes the max value in the pooling window. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the max value in the window. Similar to a convolution, we specify the window size and stride.

In case of Average Pooling, the average value for each patch on the feature map is calculated. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch. In case of Sum Pooling, the sum for each patch on the feature map is calculated.

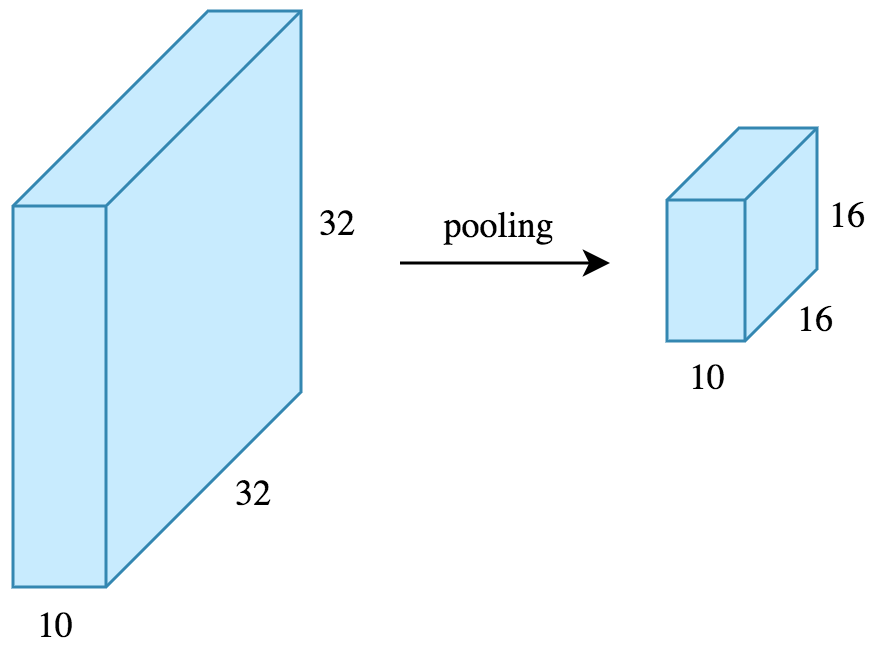
**Additionally**, pooling offers several important advantages:

* It introduces spatial hierarchy, meaning the network first focuses on local details (small features) and progressively learns broader, more abstract patterns.
* It helps reduce sensitivity to noise or small distortions in the input, making the system more robust, especially important in handwritten alphabet recognition where characters vary between writers.
* By reducing the spatial resolution, it lowers the memory and computational cost of the model, allowing deeper networks to be built without exploding resource demands.
* Finally, pooling can act as a natural bottleneck layer, encouraging the network to retain only the most meaningful activations and discard redundant or irrelevant information.



Feature Map with a 2 x 2 Max pool

Fig 3.5

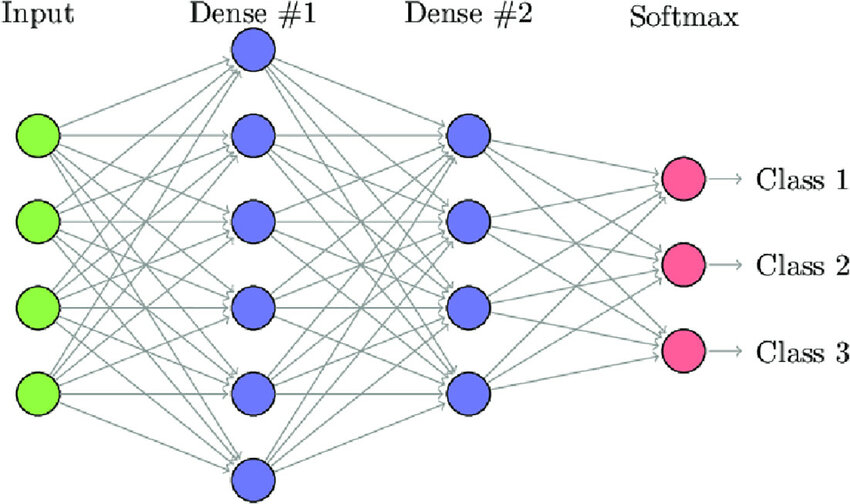


Pooling Operation on Feature Map

Fig 3.6

**5. Fully Connected Layer**

After the convolution + pooling layers, couple of fully connected layers to wrap up the CNN architecture is applied. In Fully Connected (FC) layer, every neuron of previous layer is connected to every neuron of the next layer in the model like normal MLP. This layer normally uses Softmax activation function in the output layer. The output of both convolution and pooling layers are 3D volumes, but a fully connected layer expects a 1D vector of numbers. So, the output of the final pooling layer is flattened to a vector and that becomes the input to the fully connected layer. Flattening is simply arranging the 3D volume of numbers into a 1D vector, Each value of the output vector represents the class probability.



Fully connected Layer

Fig 3.7

**6. Loss Function**

Loss function is an important statistical parameter to measure the performance of the model. Loss is defined as the difference between predicted values and expected values. It can be used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation.

Several loss functions are used to measure the performance of the network:

* Regression Loss Functions
  + Mean Squared Error Loss
  + Mean Squared Logarithmic Error Loss
  + Mean Absolute Error Loss
* Binary Classification Loss Functions
  + Binary Cross-Entropy
  + Hinge Loss
  + Squared Hinge Loss
* Multi-Class Classification Loss Functions
  + Multi-Class Cross-Entropy Loss
  + Sparse Multiclass Cross-Entropy Loss
  + Kullback Leibler Divergence Loss

**7. Regularization**

Weight regularization provides an approach to reduce the overfitting of a deep learning neural network model on the training data and improve the performance of the model on new data. There are multiple types of weight regularization, such as L1 and L2 vector norms, and each requires a hyper parameter that must be configured. Other types of regularization like Drop out and Data Augmentation can be used.

**8. Hyper Parameters**

* **Filter size**: Typically, 3x3 filters are used, but 5x5 or 7x7 are also used depending on the application. There are also 1x1 filters which we will explore in another article, at first sight it might look strange but they have interesting applications. Remember that these filters are 3D and have a depth dimension as well, but since the depth of a filter at a given layer is equal to the depth of its input, we omit that.
* **Filter count**: this is the most variable parameter; it’s a power of two anywhere between 32 and 1024. Using more filters results in a more powerful model, but we risk overfitting due to increased parameter count. Usually we start with a small number of filters at the initial layers, and progressively increase the count as we go deeper into the network.
* **Stride**: we keep it at the default value 1 so that maximum pixel is used for computation.
* **Padding**: It refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN.

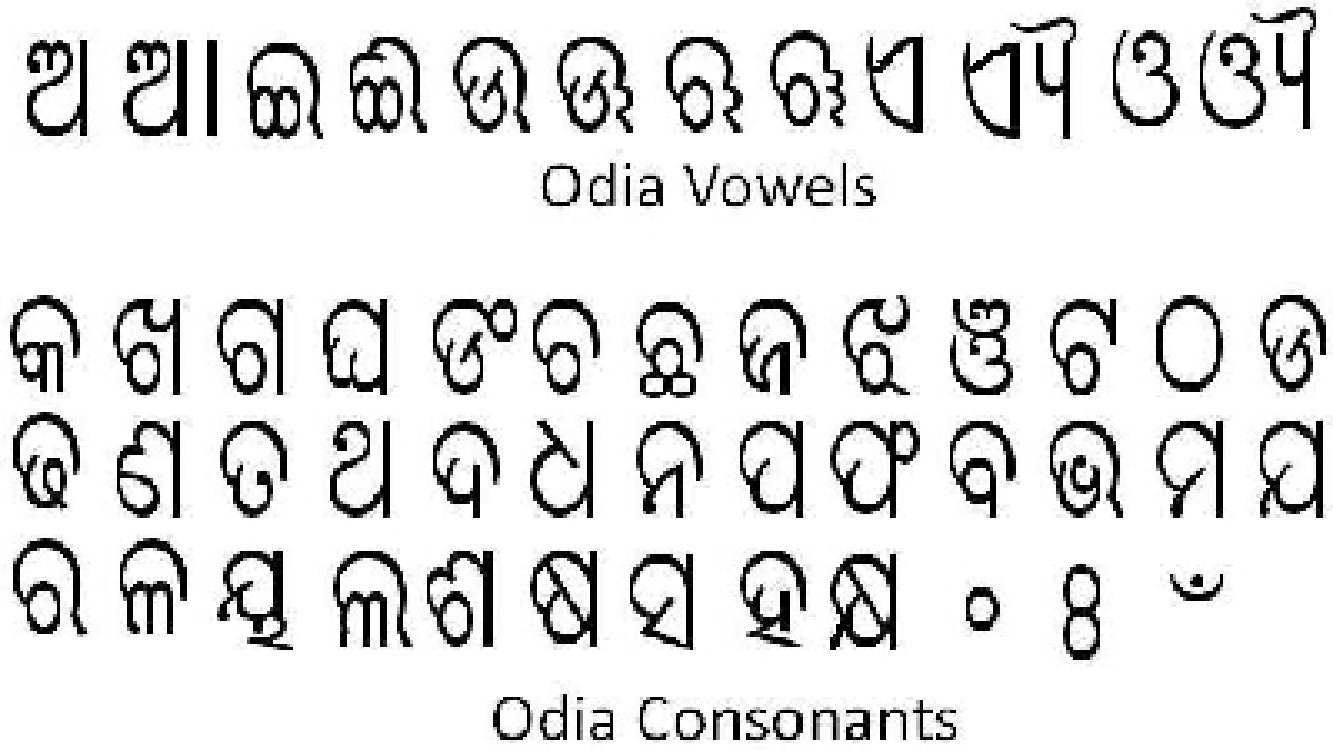
**9**. **Data Augmentation**

**Data augmentation** is a technique used to artificially expand the size and diversity of a training dataset by creating modified versions of existing images. This helps improve the model's ability to generalize to new, unseen data. In the project, data augmentation was applied to the original training images by introducing small random transformations such as rotation, shifting, shearing, and zooming. These transformations simulate real-world variations like slight tilts or changes in position, making the model more robust and reducing the risk of overfitting. Specifically, data augmentation was used to increase the number of training samples for each class to a target amount, ensuring a more balanced and comprehensive dataset for training the model effectively.

**Chapter 4**

**Data Description**

Odia is an Indo-Aryan language spoken by 33 million peoples around the globe. Odia language comprises of 12 vowels and 48 consonants. This language doesn't comprise the idea of upper and lower case letters.  Here we have developed our own dataset for processing our model. The dataset comprises samples of individual Odia characters written by different people, reflecting natural variations in handwriting styles. This helps to learn the different writing styles of Odia alphabets. These copies were scanned and converted into images. These scanned copies constituted many alphabets on the single page. Each of the cropped images was saved in the .jpg format.  Our dataset comprises of 45 classes. The table 4.1 describes the dataset distribution of each class.



Odia Vowels and Consonants

Fig 4.1

There are in total 6985 images out of which 5185 images belong to training class and 1800 images belong to test class.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CLASS** | **TRAIN** | **TEST** | **CLASS** | **TRAIN** | **TEST** |
| ଅ | 115 | 54 | ଏ | 225 | 60 |
| ଆ | 115 | 56 | ଐ | 115 | 40 |
| ଇ | 115 | 40 | ଓ | 115 | 40 |
| ଈ | 115 | 44 | ଔ | 115 | 40 |
| ଉ | 225 | 71 | କ | 115 | 40 |
| ଊ | 114 | 70 | ଖ | 115 | 40 |
| ଋ | 52 | 40 | ଗ | 115 | 40 |
| ୠ | 52 | 40 | ଘ | 115 | 40 |
| ଚ | 119 | 51 | ଙ | 115 | 40 |
| ଛ | 115 | 40 | ଥ | 115 | 40 |
| ଜ | 76 | 40 | ଦ | 115 | 40 |
| ଝ | 90 | 40 | ଧ | 115 | 40 |
| ଞ | 86 | 49 | ନ | 115 | 40 |
| ଟ | 67 | 40 | ପ | 115 | 40 |
| ଠ | 89 | 40 | ଫ | 115 | 40 |
| ଡ | 90 | 40 | ବ | 115 | 40 |
| ଢ | 115 | 40 | ଭ | 115 | 40 |
| ଣ | 115 | 40 | ମ | 115 | 40 |
| ତ | 115 | 40 | ୱ | 115 | 40 |
| ଯ | 115 | 40 | ଶ | 115 | 40 |
| ର | 115 | 40 | ଷ | 115 | 40 |
| ଳ | 115 | 40 | ସ | 115 | 40 |
| ହ | 115 | 40 |  |  |  |

Train- Test Split of dataset

Table 4.1

**Chapter 5**

**Tools and Technologies**

The following tools and technologies were used to implement the Odia hand written alphabet recognition system:

* **Programming Languages**:
  + **Python**: Python was chosen as the primary programming language for this project due to its simplicity, extensive documentation, and vast ecosystem of libraries geared toward machine learning, image processing, and data science. Python allows rapid prototyping and integration of various deep learning frameworks, making it ideal for both research and production stages.
* **Libraries and Frameworks**:
* **TensorFlow & Keras**: These deep learning frameworks were used to build and train the Convolutional Neural Network (CNN) model. Keras, with its user-friendly API, made it easier to design, compile, and train deep learning models.
* **NumPy**: Used extensively for numerical operations, array manipulation, and handling tensor transformations.
* **Pandas**: Although not a primary library for image processing, it was used for label handling and managing metadata when required.
* **Matplotlib & Seaborn**: Used for visualizing training metrics (accuracy and loss graphs), confusion matrices, and other evaluation plots.
* **Scikit-learn**: Provided tools for computing class weights (to handle imbalanced datasets), generating classification reports, confusion matrices, and splitting datasets.
* **PIL (Python Imaging Library)**: PIL was used for basic image loading, resizing, and format conversion before passing data to the model. It facilitated integration with the Keras data pipeline and augmentation techniques.
* **Image Data Generator (from Keras)**  
  To improve model generalization and deal with limited and imbalanced data, Keras’ Image Data Generator was utilized. It enabled real-time data augmentation techniques such as rotation, zoom, shift, and shear transformations, thereby increasing the effective size and diversity of the training dataset.
* **Streamlit**: For building an interactive user interface where users can upload handwritten Odia characters and view real-time predictions from the trained CNN model.
* **Development Environment**:

**Jupyter Notebook**: The entire development and experimentation process was conducted in Jupyter Notebook due to its interactive, code-and-output-in-one-cell approach. It facilitated:

* Rapid prototyping of model architectures.
* Real-time visualization of training metrics.
* Inline debugging and performance monitoring.
* Documentation of each development step through markdown annotations.

### ****Additional Utilities and Modules****

* **OS and shutil**  
  These standard Python libraries were used to programmatically manage the directory structure, copy and move image files, and automate the augmentation and dataset preparation pipelines.

## Chapter 6

## Proposed Methodology

The whole Recognition System has been divided into 2 phases:

1. The Back End Phase
2. The Front End Phase
3. **The Back End Phase**

The back end phase is the cornerstone of our Odia handwritten alphabet recognition system. It encompasses all the essential steps needed to prepare the raw data, design the deep learning architecture, and train the model for accurate predictions. Without a solid and well-planned back end phase, even the most sophisticated front-end applications will fail to deliver reliable results. This phase ensures that the system can handle variability, noise, and imperfections present in real-world handwritten samples. We divided the back end phase into three major components: preprocessing, data augmentation, and model development, each of which plays a critical role in ensuring the success of the final system.

### ****a. Preprocessing****

Preprocessing forms the initial stage of our pipeline, transforming raw image data into clean, structured, and standardized inputs ready for training a deep learning model. Handwritten data, especially in regional scripts like Odia, often comes with several challenges — varied handwriting styles, different scanning resolutions, paper marks, uneven lighting, background noise, and slight distortions. If left unaddressed, these inconsistencies can confuse the model and severely degrade its performance.

We began by organizing the raw dataset into clearly labeled folders, with each folder representing one of the 45 Odia alphabet classes. This structure allows us to apply consistent preprocessing techniques across all classes and helps downstream tools like Keras’s ImageDataGenerator to easily interpret the dataset.

Next, we applied **noise removal**. Noise in image data refers to any undesired or irrelevant visual information that can interfere with the recognition of the primary object — in our case, the Odia characters. We used basic denoising filters and contrast enhancement techniques to emphasize the foreground character and suppress unnecessary background textures. Removing this noise not only improves the clarity of the images but also helps the CNN model focus its learning on meaningful patterns rather than artifacts.

Another important step was **resizing.** We resized all images to a consistent **64×64 pixel resolution**. This decision balanced two important concerns: (1) maintaining sufficient detail to capture the unique stroke patterns of Odia characters, and (2) keeping computational costs reasonable so the model could process images efficiently during training and inference. A larger size might capture more detail but would increase the model size and training time, while a smaller size might risk losing important character features.

We also standardized the **DPI (dots per inch)** of all images to **300 DPI.** This step ensures consistency in the perceived sharpness and clarity of the images, especially when they originate from multiple scanning sources or devices. Higher DPI ensures that fine details are preserved, which is particularly important for distinguishing characters with similar shapes.

While many character recognition systems convert images to grayscale to reduce input dimensions, we deliberately kept the images in **RGB color format**. Our CNN architecture was designed to accept 3-channel inputs, and retaining the color channels gives us flexibility if color-based variations or artifacts carry useful information (for example, pen ink differences, paper color, or scanning marks).

Once preprocessing was complete, we split the dataset into three parts: **training set (80%),** **validation set (20%),** and a **separate test set** reserved for final evaluation. During splitting, we enabled the **shuffle** option to ensure that the data fed into each set represented a randomized mix of classes, preventing the model from learning spurious patterns caused by class ordering.

### ****b. Data Augmentation****

Despite preprocessing, one of the persistent challenges we faced was class imbalance and limited sample size. Some characters had far fewer samples than others, and training a deep neural network on such uneven data can result in biased models that perform well on dominant classes but poorly on underrepresented ones.

To overcome this, we employed a powerful strategy known as data augmentation. Rather than collecting additional handwritten samples, which is time-consuming and labor-intensive, we used augmentation techniques to synthetically expand our dataset. By applying small, controlled distortions and variations to existing images, we generated new, unique training samples that helped the model generalize better to unseen data. We implemented data augmentation using Keras’s ImageDataGenerator class, which performs real-time augmentation on image batches during training. This approach avoids the need to store thousands of extra images on disk, saving storage space and making the training process more memory-efficient.

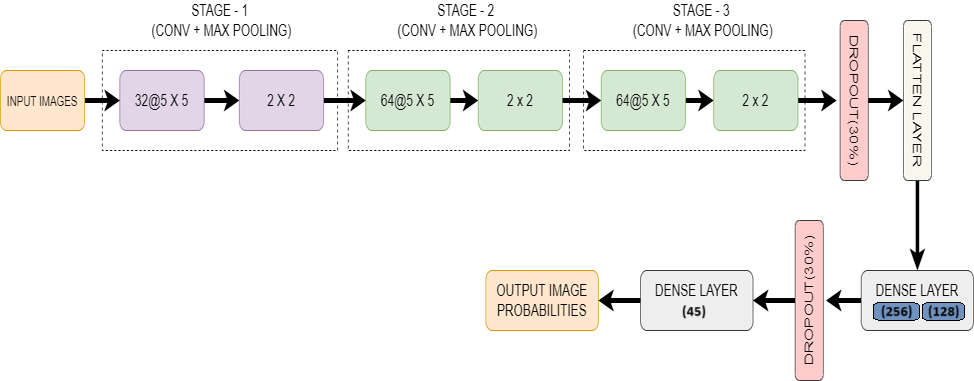
The augmentation transformations we applied included:

* **Rotation (up to 10°):** Mimics natural variations in handwriting where characters may be slightly slanted.
* **Width and Height Shifts (up to 10%):** Simulates the variability in character placement inside the image frame.
* **Shearing:** Introduces a diagonal slant to images, representing variations where strokes are distorted or stretched.
* **Zooming (up to 10%):** Helps the model learn characters that may appear slightly larger or smaller depending on the writer or scanning device.

Our augmentation pipeline aimed to increase each class’ssample size to at least 300 images. This balancing step ensured that the model received equal attention across all classes, reducing the chance of overfitting to dominant classes. Additionally, by presenting the model with diverse variations of each character, data augmentation significantly improved the generalization ability of the model, making it more capable of handling handwritten variations it had never seen before.

### ****c. Model Development****

With a clean, augmented dataset prepared, we moved on to developing the Convolutional Neural Network (CNN), the core machine learning model behind our recognition system. CNNs are particularly powerful for image recognition tasks because they can automatically learn hierarchical features — from simple edge detectors in early layers to complex shape detectors in deeper layers — without the need for manual feature engineering.



|  |  |
| --- | --- |
|  | Architecture of the Model  Fig 6.1 |

Our CNN architecture was carefully designed through experimentation to balance accuracy, efficiency, and robustness. Here’s a detailed breakdown of the architecture:

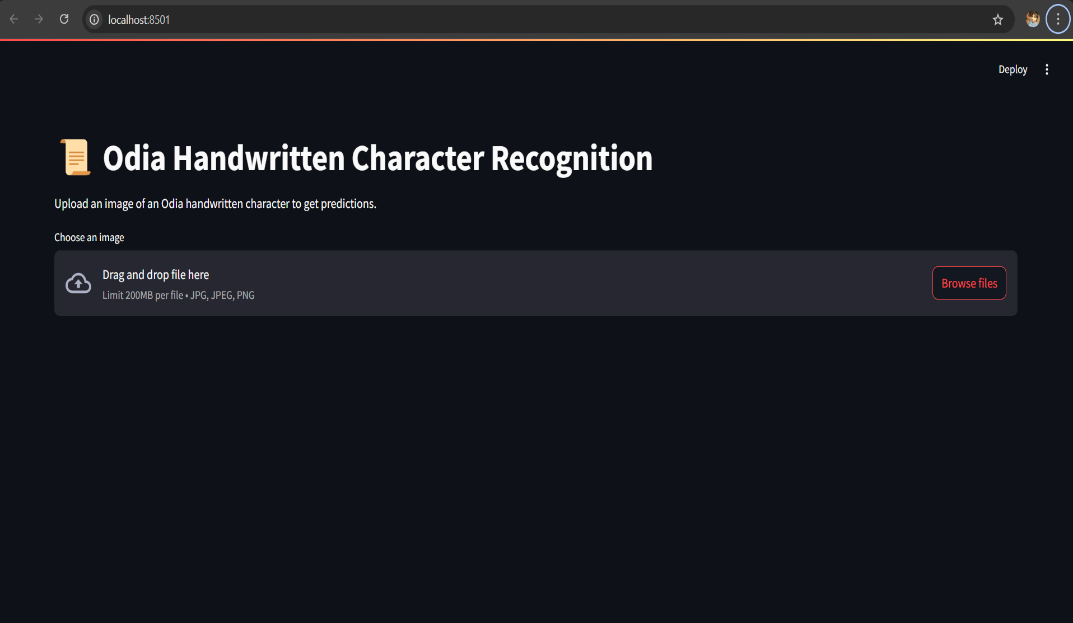
* **First Convolutional Block:**  
  We began with a convolutional layer containing 32 filters of size **5×5**, using a stride of 1×1 and same padding to preserve the spatial dimensions of the input. This layer learns low-level features like edges and lines. We applied ReLU activation to introduce non-linearity, which is essential for learning complex patterns, followed by a 2×2 maxpooling layer to downsample the feature maps and reduce computation.
* **Second Convolutional Block:**  
  The next block again uses 32 filters with the same kernel size, ReLU activation, and max pooling. This block builds on the features detected earlier to identify more complex patterns, such as combinations of edges or simple shapes.
* **Third Convolutional Block:**  
  We increased the filter count to 64 in this layer to capture even higher-level features, maintaining the 5×5 kernel size and 2×2 pooling. By stacking more filters, the model becomes capable of distinguishing subtle differences between similar-looking characters.
* **Flatten Layer:**  
  After the convolutional blocks, the output feature maps are flattened into a one-dimensional vector, preparing them for the fully connected (dense) layers.
* **Dense Layers:**  
  We added two dense layers: one with 128 neurons and another with 256 neurons, both using ReLU activation. These layers help the model learn complex combinations of the extracted features. To prevent overfitting, we applied a 30% dropout rate after the third pooling layer and again after the first dense layer. Dropout works by randomly disabling neurons during training, forcing the network to develop redundant representations and improving generalization.
* **Output Layer:**  
  The final dense layer contains 45 neurons, corresponding to the 45 Odia alphabet classes, with a softmax activation function. Softmax converts the output into a probability distribution, where the highest-probability class is selected as the model’s prediction.

For training, we used:

* **Adam optimizer** with a learning rate of **0.0003** for efficient gradient descent.
* **Categorical crossentropy** as the loss function, suitable for multi-class classification.
* **Batch size of 64** and **10 training epochs**.

1. **The Front End Phase**

In the second phase, we have developed a website for user input. Our application is developed entirely using **Streamlit**, a Python-based web application framework specifically designed for building data science and machine learning interfaces. Users can upload images of Odia handwritten characters directly through the UI. The uploaded image is preprocessed and passed to a pre-trained Convolutional Neural Network (CNN) model, which predicts the top three most probable Odia characters along with their respective probabilities. The results are displayed in a tabular format for easy interpretation. Basic styling is applied using **embedded CSS within Streamlit** to enhance the visual presentation. This approach allows for a **lightweight, interactive, and responsive** user experience without requiring separate backend technologies like Flask or traditional frontend tools like HTML, CSS, and Bootstrap. The user input is taken in form of image and the most three likely outcomes is displayed on the web page in tabular format. The Screenshots of the developed web application are depicted in the following Fig 6.2 & Fig 6.3.

****

Homepage of the Application

Fig 6.2

****

Fig 6.3

Result page of the Application

## 

Fig 6.4

User Interface code

## Chapter 7

**Results and Discussion**

The experiments are performed on a laptop powered by AMD Ryzen 5 5600H with Raedon as a graphics card for GPU support. As far as the software requirement is concerned, Python framework’ and ‘streamlit’ libraries are used. Program is written in Jupyter notebook environment.

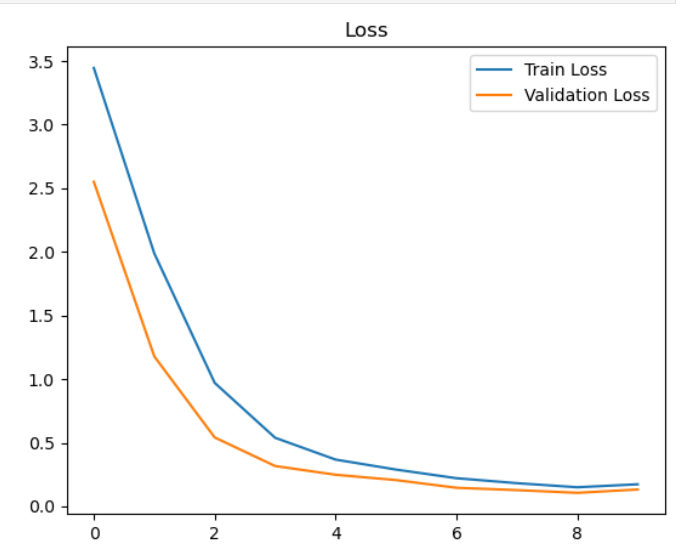
Our Proposed CNN Model has secured an accuracy of 79.44% over the whole dataset.

* When the validation loss starts to increase from the training loss, then the training should be stopped. When the gap between the losses starts widening, then it indicates overfitting in the model. This can be prevented by decreasing the network size or increasing the dropout.

* If your training/validation losses are about equal then the model is underfitting. It can be prevented by increasing the size of the model. It means increasing the number of neurons

in the layer.

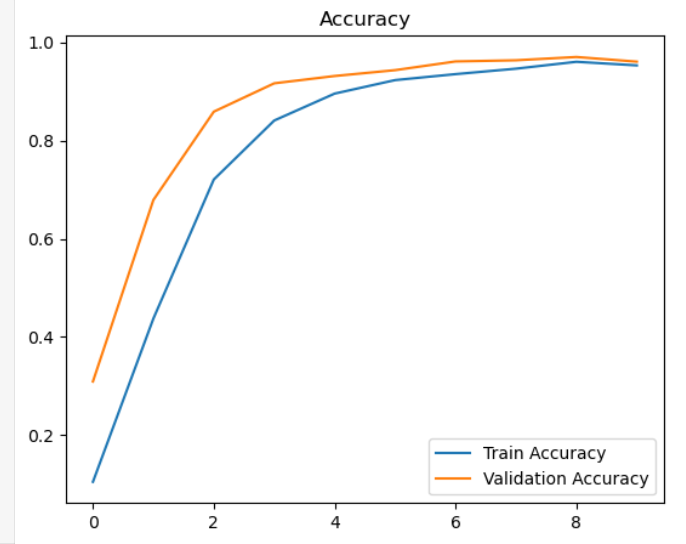
Hence, the plot shows that as the number of epochs increases the gap between training and test loss starts decreasing. Hence, it is fixed to 10. The losses for training and validation during the last epochs are 0.15 and 0.16 which is acceptable



Plot showing loss vs no. of epochs

Fig 7.1

The plot in Fig 7.1 denotes the relationship between number of epochs and the accuracy. As the number of epochs starts increasing the accuracy also increase. Training of the model should be stopped when the gap between train and test accuracy starts widening. Hence, the number of epochs is fixed to 10 as after these the lines starts diverging which can lead to overfitting in the model. The training and test accuracy during the last few epochs are around 95% and 92% respectively.



Plot showing accuracy vs no. of epochs

Fig 7.2

To achieve the best performance for the architecture, careful manual tuning of hyperparameters was carried out. While there are many hyperparameters that can be adjusted, we focused on tuning key ones such as the number of epochs, batch size, the number of neurons in the fully connected layers, the optimization algorithm, and dropout rates. The remaining parameters were selected based on commonly recommended values from literature and previous research, which are known to perform well in similar tasks. This tuning process also helped us monitor whether the model was overfitting or underfitting.

### ****Detailed Hyperparameter Tuning in the Project****

In building a high-performing Convolutional Neural Network (CNN) model for Odia handwritten character recognition, careful adjustment of hyperparameters was essential. Unlike model weights, hyperparameters are external settings that define how the learning process operates. Choosing the right values for these settings can greatly influence the model’s ability to learn meaningful patterns, avoid overfitting, and generalize effectively to new, unseen data. Below is an overview of the main hyperparameters we tuned in this project, why they matter, and how we selected their final values.

1. **Batch Size and No of epochs**

Batch size determines how many training samples are processed before the model updates its weights. Using the entire dataset at once can be computationally intensive, while updating after each sample can lead to unstable gradients. We experimented with batch sizes of 16, 32, and 64, ultimately choosing batch size 64 because it offered smooth, stable training while efficiently using available hardware resources.

Epochs refer to the number of times the model passes through the entire training dataset. We tested both short runs (5–10 epochs) and longer runs (20–30 epochs). However, we noticed that after 10 epochs, the model started showing signs of overfitting — training accuracy continued to rise, but validation accuracy stopped improving or slightly declined. Therefore, we finalized 10 epochs as the optimal setting, allowing the model to learn effectively without memorizing the training data.

1. **Optimization Algorithm**

The optimizer is responsible for updating the model’s weights in a way that reduces the loss function — essentially minimizing the gap between predictions and true labels. We compared several widely used optimizers, including **SGD (Stochastic Gradient Descent), RMSProp**, and **Adam**. Through manual testing, **Adam (Adaptive Moment Estimation)** consistently produced faster convergence and better final accuracy. Adam’s strength comes from combining adaptive learning rates (like RMSProp) and momentum (like SGD), making it particularly effective for deep architectures like CNNs. Using Adam gave us an approximate **6% improvement** in accuracy compared to the other tested optimizers.

1. **Number of Neurons**

The number of neurons in the fully connected (dense) layers plays a crucial role in the model’s ability to capture and combine complex patterns extracted by the earlier layers. We tuned the number of neurons in the second-to-last dense layer — the layer connecting the feature extractor to the final classifier — testing values like **128, 256**, and **1000** neurons. While increasing the number of neurons did slightly improve performance, we found diminishing returns beyond **256 neurons**, where further increases added computational cost without meaningful accuracy gains. Therefore, we selected **256 neurons** as the best balance

between complexity and performance.

1. **Dropout Regularization**

To prevent overfitting, we applied dropout, a regularization technique where a random fraction of neurons is turned off during each training pass, forcing the network to learn more robust, redundant features. We experimented with dropout rates ranging from 0.0 (no dropout) to 0.9 (heavy dropout). After testing, we found that a 30% dropout rate worked best, applied after:

* The third convolutional + max pooling layer
* The first dense layer

This dropout setting effectively reduced overfitting while maintaining the model’s ability to learn meaningful patterns.

### ****5.Model Performance Metrics****

After completing the training and tuning phases, we rigorously evaluated the CNN model on a separate test dataset, ensuring that no data leakage from training or validation sets occurred. To comprehensively assess model performance, we employed several widely accepted metrics:

* **Accuracy**: The proportion of correctly predicted samples across all classes. This metric gives an overall sense of how well the model performs but can sometimes mask issues like class imbalance.
* **Precision**: The fraction of true positive predictions among all positive predictions made by the model. High precision indicates that when the model predicts a class, it’s usually correct.
* **Recall (Sensitivity)**: The fraction of true positive predictions among all actual positive cases. High recall means the model successfully captures most of the true instances of a class.
* **F1-Score**: The harmonic mean of precision and recall, providing a balanced measure that accounts for both false positives and false negatives, especially useful when classes are imbalanced.

By analyzing these metrics, we ensured that our model was not just accurate in aggregate but also fair and effective across all 45 classes of Odia handwritten characters. Additionally, we plotted confusion matrices and generated classification reports to pinpoint which classes the model struggled with, helping guide future improvements.

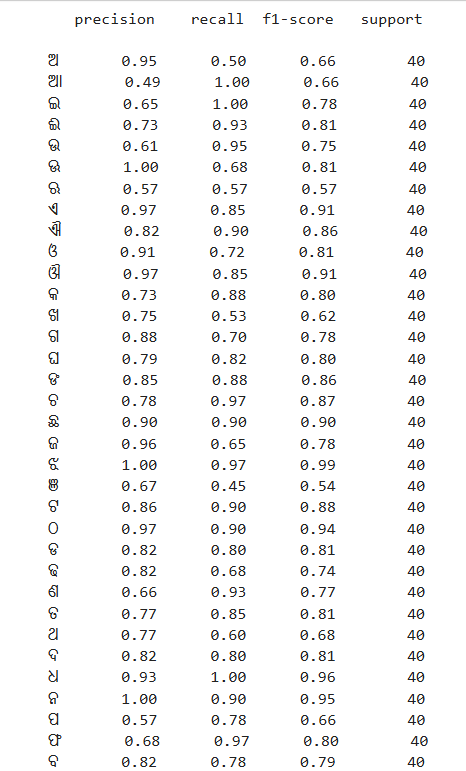


Fig 7.3

**Model Performance Metrics**

Accuracy Comparison In order to validate the proposed machine learning model, we have analyzed many research papers based on this topic. Many researchers have done lots of research on handwritten character recognition but very less research is done in the field of Odia character recognition.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Authors | Features | Classifier | Accuracy (%) | |
| ISI kolkata | IITBBS |
| Bhowmik et al.[25] | Scalar | HMM | 90.50 | 88.74 |
| Dash et al. [26] | Gestalt | Complexity metric | 92.82 | 91.40 |
| Roy et al. [27] | Directional | QDF | 94.81 | 94.12 |
| Pal et al. [28] | Directional | MQC | 98.40 | 96.30 |
| Dash et al. [29] | Hybrid | DLQDF | 98.50 | 98.28 |
| Dash et al. [30] | AZNRST | K-NN | 99.10 | 98.60 |
| Dash et al. [31] | BESAC | Random forest | 98.44 | 97.30 |
|  |  | SVM | 99.02 | 98.56 |
|  |  | Nearest neighbor | 99.35 | 98.90 |

## Comparison of various Classifiers with their Accuracy

## Table 7.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Authors | Techniques for Feature Extraction | Classifier used | Recognition Rate (%) | |
| NIT, RKL(Odia Character) | ISI, Kolkata(Odia numeral) |
| Pal et al. in [5] | Curvature | Quadratic | 94.6 |  |
| Sethy et al. in [32] | DWT | BPNN | 94.8 |  |
| Mishar et al. in [10] | DCT+DWT | SVM |  | 92 & 87.5 |
| Sethy et al. in [33] | DCT+Binarization | ANN |  | 90 & 80.2 |
| Sethy et al. [7] | Symmetry axes | J48(Decision Tree) | 95.6 | 96.2 |

## Comparison of various Classifiers with their Accuracy

## Table 7.2

## Table 7.2 and Table 7.3 compare the accuracy of different research works done on Odia handwritten character recognition. The datasets used in these researches are ISI, Kolkata (Odia numerals and character), IIT, BBSR (Odia numerals and character) and NIT, RKL (Odia character). Our machine learning model produces accuracy of 79.44% over our dataset. The study of these research paper shows that recognition rate is highly dataset dependent. Each dataset has its own features that need to be extracted using various techniques for training the model. The researchers have proposed various feature extraction techniques that have been applied to the dataset and trained the model.

## 

### Chapter 8

### Future Work

While the current Odia handwritten alphabet recognition system demonstrates promising accuracy and performance, there are numerous opportunities for further improvement, enhancement, and expansion. Below are several key areas identified for future work:

**1.Expansion of the Dataset**  
One of the most crucial steps to improve the model’s robustness is to increase the size and diversity of the dataset. Currently, the dataset may contain limited handwriting samples, which can restrict the model’s ability to generalize across different writing styles. By collecting additional samples from a wider demographic—such as children, adults, elderly individuals, and people using various types of pens, pencils, or markers—the model can be trained on a more representative dataset. This would enable it to handle natural variations in handwriting, ultimately improving real-world recognition performance.

**2. Incorporation of More Advanced Neural Network Architectures**  
While the current system uses a CNN-based architecture, future work could explore more advanced deep learning models. Architectures such as ResNet (Residual Networks), DenseNet (Dense Convolutional Networks), or EfficientNet are known for their high accuracy in image classification tasks. These networks can learn deeper and more complex features, which could help distinguish subtle differences between visually similar Odia characters. Additionally, experimenting with hybrid models that combine convolutional layers with recurrent or attention-based mechanisms might further improve character recognition accuracy.

**3. Use of Transfer Learning**  
Transfer learning involves using pre-trained models that have been trained on large, generic datasets (such as ImageNet) and fine-tuning them on the target dataset. Applying transfer learning to Odia character recognition could boost performance significantly, especially when labeled data is limited. Since the early layers of such models capture generic image features (edges, shapes, textures), they can provide a solid foundation for training on more specific tasks like handwritten alphabet recognition.

**4. Model Optimization for Deployment**  
For practical deployment on mobile devices, embedded systems, or low-resource environments, optimizing the model’s size and computational efficiency is essential. Techniques such as model pruning (removing unnecessary weights), quantization (reducing precision), and knowledge distillation (training a smaller model to mimic a larger one) can help create lightweight models without sacrificing much accuracy. This would make the system faster and more energy-efficient, allowing for real-time character recognition even on resource-constrained devices.

**5. Real-Time Application Development**  
Currently, the system is tested using Flask-based and Streamlit-based web applications. Future work could focus on developing a fully functional mobile or desktop application that allows users to capture handwritten characters using a camera and receive instant predictions. Integrating the system with mobile app frameworks (such as Flutter or React Native) would make the technology more accessible and practical for everyday use, especially for educational or document digitization purposes.

**6. Extension to Multilingual Handwritten Recognition**  
While this project focuses on Odia characters, the techniques and methodologies can be adapted to recognize handwritten characters from other Indian languages, such as Bengali, Hindi, Telugu, or Kannada. Developing a multilingual recognition system would broaden the application scope and provide a valuable tool for preserving regional languages in digital form. It could also support cross-language educational tools or translation systems.

**7. Continuous Learning Through User Feedback**  
Implementing a feedback loop where the system collects corrections from users (for example, when a user flags a wrong prediction) could allow for continuous model improvement over time. This type of human-in-the-loop learning would make the system adaptive, allowing it to learn from its mistakes and gradually enhance its accuracy in real-world conditions.

**8. Advanced Error Analysis and Visualization**  
Conducting detailed error analysis using techniques such as saliency maps, Grad-CAM, or confusion matrix heatmaps could help researchers understand where the model is making mistakes. Insights from such analyses can guide future adjustments in the architecture, preprocessing, or training pipeline to specifically address the weak points in the current system.

**9. Integration with Optical Character Recognition (OCR) Systems**  
Another interesting direction is to integrate the handwritten recognition system with broader OCR systems that can handle both typed and handwritten texts. This could lead to the development of a comprehensive document digitization tool, capable of reading mixed-content documents, including forms, historical manuscripts, or handwritten notes.

By pursuing these future directions, the Odia handwritten alphabet recognition system can evolve from a research prototype into a highly versatile and widely applicable technology that benefits educational institutions, government organizations, cultural preservation projects, and individual users alike.

## Chapter 8

**Conclusion**

Deep Convolutional Neural Networks (CNNs) have shown great success in solving computer vision tasks, and in this project, we implemented a CNN model specifically designed for Odia handwritten alphabet recognition. Our trained model achieves an accuracy of 79.44% on the test set, demonstrating strong learning ability and reliable generalization across varied handwritten samples.

The architecture consists of three convolutional-pooling stages followed by two dense layers, enabling effective feature extraction and classification. Compared to conventional Artificial Neural Networks (ANNs), this CNN approach offers superior performance in capturing spatial patterns and character-level details, making it better suited for handwritten recognition tasks. To reduce overfitting, dropout regularization was applied after the final convolutional block and dense layers, and extensive data augmentation — including rotation, zoom, shearing, and flipping — was used to enrich the training dataset and improve model robustness.

While the current dataset was carefully prepared and augmented, the system’s performance could be further enhanced by incorporating higher-resolution images and expanding the dataset size, which would allow for more fine-tuned training and improved detection accuracy.

Finally, we developed an interactive Streamlit web application that allows users to upload handwritten images and view real-time predictions. The app displays the most probable character match along with the top two alternative predictions and their probability scores, offering an engaging and user-friendly experience.

**References**

1. Jena, Om Prakash, et al. "Odia Characters and Numerals Recognition using Hopfield Neural Network based on Zoning Features." *International Journal of Recent Technology and Engineering* 8.2 (2019): 4928-4937.
2. Dash, Kalyan S., Niladri B. Puhan, and Ganapati Panda. "BESAC: Binary External Symmetry Axis Constellation for unconstrained handwritten character recognition." *Pattern Recognition Letters* 83 (2016): 413-422.
3. Dash, Kalyan S., Niladri B. Puhan, and Ganapati Panda. "Sparse Concept Coded Tetrolet Transform for Unconstrained Odia Character Recognition." *arXiv preprint arXiv:2004.01551* (2020).
4. Sufian, Abu, et al. "Bdnet: bengali handwritten numeral digit recognition based on densely connected convolutional neural networks." *Journal of King Saud University-Computer and Information Sciences* (2020).
5. U. Pal, T. Wakabayashi and F. Kimura, "A System for Off-Line Oriya Handwritten Character Recognition Using Curvature Feature," 10th International Conference on Information Technology (ICIT 2007), Orissa, 2007, pp. 227-229, doi: 10.1109/ICIT.2007.63.
6. I. Rushiraj, S. Kundu and B. Ray, "Handwritten character recognition of Odia script," 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), Paralakhemundi, 2016, pp. 764-767, doi: 10.1109/SCOPES.2016.7955542.
7. A. Sethy, P. K. Patra, S. Nayak and P. M. Jena, "Symmetric Axis Based Off-Line Odia Handwritten Character and Numeral Recognition," 2017 3rd International Conference on Computational Intelligence and Networks (CINE), Odisha, 2017, pp. 83-87, doi: 10.1109/CINE.2017.27.
8. U. Pal, N. Sharma, T. Wakabayashi and F. Kimura, "Off-Line Handwritten Character Recognition of Devnagari Script," Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), Parana, 2007, pp. 496-500, doi: 10.1109/ICDAR.2007.4378759.
9. T. K. Bhowmik, S. K. Parui, U. Bhattacharya and B. Shaw, "An HMM Based Recognition Scheme for Handwritten Oriya Numerals," 9th International Conference on Information Technology (ICIT'06), Bhubaneswar, 2006, pp. 105-110, doi: 10.1109/ICIT.2006.29.
10. T. K. Mishra, B. Majhi and S. Panda, "A comparative analysis of image transformations for handwritten Odia numeral recognition," 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Mysore, 2013, pp. 790-793, doi: 10.1109/ICACCI.2013.6637276.
11. P. Pujari and B. Majhi, "Performance comparison of ensemble models for recognition of offline handwritten Odia numerals," 2015 IEEE Power, Communication and Information Technology Conference (PCITC), Bhubaneswar, 2015, pp. 29-35, doi: 10.1109/PCITC.2015.7438187.
12. A. Ray, S. Rajeswar and S. Chaudhury, "Text recognition using deep BLSTM networks," 2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR), Kolkata, 2015, pp. 1-6, doi: 10.1109/ICAPR.2015.7050699.
13. T. Wakabayashi, U. Pal, F. Kimura and Y. Miyake, "F-ratio Based Weighted Feature Extraction for Similar Shape Character Recognition," 2009 10th International Conference on Document Analysis and Recognition, Barcelona, 2009, pp. 196-200, doi: 10.1109/ICDAR.2009.197.
14. S. Chanda, K. Franke and U. Pal, "Text Independent Writer Identification for Oriya Script," 2012 10th IAPR International Workshop on Document Analysis Systems, Gold Cost, QLD, 2012, pp. 369-373, doi: 10.1109/DAS.2012.86.
15. T. M. Bernard and A. Manzanera, "Improved low complexity fully parallel thinning algorithm," Proceedings 10th International Conference on Image Analysis and Processing, Venice, Italy, 1999, pp. 215-220, doi: 10.1109/ICIAP.1999.797597.
16. Mishra, T. K., Majhi, B., Sa, P. K., & Panda, S. (2014). Model based odia numeral recognition using fuzzy aggregated features. *Frontiers of Computer Science*, *8*(6), 916-922.
17. Xu, B., Ye, Y., & Nie, L. (2012, June). An improved random forest classifier for image classification. In *2012 IEEE International Conference on Information and Automation* (pp. 795-800). IEEE.
18. Swethalakshmi, H., Jayaraman, A., Chakravarthy, V. S., & Sekhar, C. C. (2006, October). Online handwritten character recognition of Devanagari and Telugu Characters using support vector machines.
19. Sarangi, Pradeepta K., P. Ahmed, and Kiran K. Ravulakollu. "Naïve bayes classifier with lu factorization for recognition of handwritten Odia numerals." *Indian Journal of Science and Technology* 7.1 (2014): 35-38.
20. Impedovo, Sebastiano, Francesco Maurizio Mangini, and Donato Barbuzzi. "A novel prototype generation technique for handwriting digit recognition." *Pattern Recognition* 47.3 (2014): 1002-1010.
21. Liu, Cheng-Lin, and Ching Y. Suen. "A new benchmark on the recognition of handwritten Bangla and Farsi numeral characters." *Pattern Recognition* 42.12 (2009): 3287-3295.
22. Pal, Umapada, Ramachandran Jayadevan, and Nabin Sharma. "Handwriting recognition in indian regional scripts: a survey of offline techniques." *ACM Transactions on Asian Language Information Processing (TALIP)* 11.1 (2012): 1-35.
23. Jindal, M. K., R. K. Sharma, and G. S. Lehal. "Segmentation of horizontally overlapping lines in printed Indian scripts." *International Journal of Computational Intelligence Research* 3.4 (2007): 277-286.
24. Hamamura, Tomoyuki, Takuma Akagi, and Bunpei Irie. "An analytic word recognition algorithm using a posteriori probability." *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. IEEE, 2007.
25. Mishra, T. K., Majhi, B., Sa, P. K., & Panda, S. (2014). Model based odia numeral recognition using fuzzy aggregated features. *Frontiers of Computer Science*, *8*(6), 916-922.
26. K.S. Dash , N.B. Puhan , G. Panda , Handwritten Odia numeral recognition: Com- bining gestalt configural superiority effect and inner distance based complexity metric, in: Proceedings of IEEE 14th International Symposium on Signal Pro- cessing and Information Technology, 2014, pp. 1–6 .
27. K. Roy , T. Pal , U. Pal , F. Kimura , Oriya handwritten numeral recognition system, in: Proceedings of IEEE 8th International Conference on Document Analysis and Recognition, 2005, pp. 770–774 .
28. U. Pal , N. Sharma , T. Wakabayashi , F. Kimura , Handwritten numeral recognition of six popular Indian scripts, in: Proceedings of IEEE 9th International Confer- ence on Document Analysis and Recognition, 2007, pp. 749–753 .
29. K.S. Dash , N.B. Puhan , G. Panda , A hybrid feature and discriminant classi- fier for high accuracy Odia handwritten numeral recognition, in: *Proceed- ings of IEEE Region 10 Technical Symposium* , *TENSYMP’14* , 2014, pp. 531–535 .
30. K.S. Dash , N.B. Puhan , G. Panda , Handwritten numeral recognition using non-redundant Stockwell transform and bio-inspired optimal zoning, IET Im- age Process. 9 (10) (2015) 874–882 .
31. Dash, Kalyan S., Niladri B. Puhan, and Ganapati Panda. "BESAC: Binary External Symmetry Axis Constellation for unconstrained handwritten character recognition." *Pattern Recognition Letters* 83 (2016): 413-422.
32. Sethy, Abhisek, Prashanta Kumar Patra, and Deepak Ranjan Nayak. "Off-line handwritten Odia character recognition using DWT and PCA." *Progress in Advanced Computing and Intelligent Engineering*. Springer, Singapore, 2018. 187-195.
33. Sethy, Abhisek, and Prashanta Kumar Patra. "Off-line Odia handwritten numeral recognition using neural network: a comparative analysis." *2016 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 2016.