Odia Handwritten Optical Character Recognition Using Transfer Learning and Pre-trained Tesseract Odia dataset

M. Tech(Signal Processing and Digital Design) Aishwarya Keller 2021

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

Signal Processing and Digital Design

Submitted by

Swastik Mohanty

(2K21/SPD/19)

Under the supervision of

Piyush Tewari (Asst Prof, ECE Dept.)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi -110042

JUNE 2021

CONTENTS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I **Swastik Mohanty** student of M.Tech (Signal Processing and Digital Design), hereby declare that the project Dissertation titled "**Odia Handwritten Optical Character Recognition Using Transfer Learning and Pre Trained Tesseract Odia dataset "** which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similartitle or recognition.

Place: Delhi

Swastik Mohanty

Date: 31st May 2023

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Report titled "**Odia Handwritten Optical Character Recognition Using Transfer Learning and Pre Trained Tesseract Odia dataset**" which is submitted by **Swastik Mohanty**, **2K21/SPD/19** of Electronics and Communication Department, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **Piyush Tewari**
Date: 31st May 2023 **SUPERVISOR**

ACKNOWLEDGEMENT

A successful project can never be prepared by the efforts of the person to whom the project is assigned, but it also demands the help and guardianship of people who helped in completion of the project.

I would like to thank all those people who have helped me in this research and inspired me during my study.

With profound sense of gratitude, I thank Piyush Tewari, my Research Guide, for his encouragement, support, patience and his guidance in this research work.

Furthermore, I would also like to thank the Head of the Department, Electronics and Communication, Prof O.P. Verma, who gave me the permission to use all required equipment and the necessary format to complete the report.

I take immense delight in extending my acknowledgement to my family and friends who have helped me throughout this research work.

Swastik Mohanty

ABSTRACT

.

This study aims to address the problem of Optical Character Recognition (OCR) for the Odia language using a transfer learning.OCR is a technology used to convert different types of documents, such as scanned paper documents, PDF files or images captured by a digital camera, into editable and searchable data. The Odia language, like many other languages, has unique characteristics and complexities in its script that pose challenges for OCR. Transfer learning, which involves applying knowledge learned from one problem to a different but related problem, is seen as a potential solution to these challenges. This technique is especially beneficial when the dataset for the specific task (in this case, Odia OCR) is small, as it leverages the knowledge captured by models pre-trained on larger, more diverse datasets. In this study, a pre-trained Convolutional Neural Network (CNN) model for feature extraction. CNNs are a type of deep learning model that are particularly good at processing grid-like data, such as images. A pre-trained CNN model is a model that has been previously trained on a large dataset, usually on a general task like identifying objects within images. The learned weights of this model, which capture the learned features from the previous task, are then used as the starting point for the new task. After initializing the model with the pre-trained weights, the authors fine-tune it on the specific task of recognizing Odia characters. Fine-tuning involves continuing the training process on the new task, adjusting the weights of the model to better fit the new data. The specifics of fine-tuning, such as which layers of the model to fine-tune and the learning rate to use, can vary depending on the specifics of the task and the amount of available data. The dataset used in this study consists of images of 8 unique vowels in the Odia language. Image datasets for deep learning often require preprocessing to ensure that they can be efficiently and effectively fed into the model. In this case, the authors applied several preprocessing techniques like Image re-sizing, normalization, data splitting. The study thus offers a comprehensive approach to tackling the problem of Odia OCR using transfer learning, from using a pre-trained CNN model to fine-tuning it on a specific dataset, and meticulously preparing the data for optimal results.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation		Full form
OCR	-	Optical Character Recognition
CNN	-	Convolutional Nueral Networks
RNN	-	Recurrent Neural Networks
LSTM	-	Long Short Term Memory
CTC	-	Connectionist Temporal Classification
BERT	-	Bidirectional Encoder Representations from Transformers
GPT	-	Generative Pre-Training
SVM	-	Support Vector Machine
HMM	-	Hidden Markov Model
PIL	-	Python Imaging Library
API	-	Application Programming Interface

CHAPTER 1
INTRODUCTION

1.1 Overview

Optical Character Recognition (OCR) is a technology that turns scanned paper documents, PDF files, and pictures taken with a digital camera into data that can be edited and searched. OCR works by looking at the shapes and patterns of the letters in the scanned picture or document and turning them into text characters by comparing them to a set of known characters. The quality of the source document and the type of OCR technology used can affect how well it works. OCR can be used in a lot of different fields: Automation of Data Entry: Optical Character Recognition (OCR) is often used to automate data entry tasks like putting information from paper forms into a computer system. Compared to entering data by hand, this can save a lot of time and make fewer mistakes. Document Archiving: Optical Character Recognition (OCR) is used to scan printed documents so they can be edited, searched, and saved in a smaller space. It also lets machines do things like do brain searches and learn on their own. Book Digitization: Optical Character Recognition (OCR) is used to turn printed books into digital forms that can be read on computers or e-readers. Through projects like Google Books and others, this is how a lot of written information is now available online. Automated Form Processing: Optical Character Recognition (OCR) is used to pull information directly from forms like invoices, applications, and surveys. This can greatly cut down on mistakes and speed up the process.

1.1.1 OCR for Indic Scripts

Optical Character Recognition (OCR) for Indic scripts isn't as old as it is for Latin-based scripts, mostly because Indic scripts are so complicated and varied. Indic languages include many different writing systems, such as Devanagari (used for Hindi, Marathi, and Nepali), Bengali, Gujarati, Oriya, Gurmukhi (used for Punjabi), Telugu, Kannada, Malayalam, and more. Each of these scripts has its own set of problems that OCR has to deal with.

In the late 1990s and early 2000s, the digitization of data and the need for automatic data entry systems made it more important to make OCR systems for Indic scripts.

In the beginning, most of the work was done to recognise printed papers. Handwritten text was given less attention because it was harder to read. Early systems often used methods called "template matching," in which each character was compared to a template that had already been set up.

Researchers began looking into machine learning methods for Indic OCR around the middle of the 2000s. These methods, like Support Vector Machines (SVMs) and Hidden Markov Models (HMMs), let OCR systems learn from data, which made them more accurate.

The Digital Library of India (DLI) was one of the most important projects during this time. Its goal was to digitise and store important works of art, literature, and science from India. As part of this project, OCR tools for a number of Indic scripts were made so that printed books could be digitised.

Since the beginning of deep learning methods a decade ago, the field of OCR for Indic scripts has made a lot of progress. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), as well as their variations like Long Short Term Memory (LSTM) networks, have been used to make OCR systems that are very accurate.

Even with these improvements, OCR for Indic scripts is still hard because of things like the wide range of writing styles, the complicated way characters are put together, and the fact that many scripts have modifiers and diacritics. There is still study going on in the field.

1.1.2 Previously used techniques

Modern techniques for Optical Character Recognition (OCR) for Indic writing usually use deep learning methods, which have made OCR systems much better than they were with older methods. Some of these strategies are:

Convolutional Neural Networks (CNNs): CNNs have been used successfully in optical character recognition (OCR) systems to recognise characters and words. They are very good at processing pictures because they can automatically and adaptively learn how features are arranged in space.

Recurrent Neural Networks (RNNs): OCR sequence recognition jobs often use RNNs, especially Long Short-Term Memory (LSTM) networks. They are good at dealing with text data sets that have different lengths.

Connectionist Temporal Classification (CTC): CTC is a type of loss function that is often used in OCR jobs to train sequence recognition models like RNNs. It's especially helpful for jobs like OCR where the timing of the input sequence doesn't match the timing of the output sequence.

Transformers: Since the "Attention is All You Need" paper came out, transformer models have also been used for OCR jobs. These models, which include popular variants like BERT, GPT, and T5, have shown state-of-the-art performance on a range of natural language processing tasks and are now being used for OCR as well.

Transfer Learning: In this type of learning, a model that has already been trained is used to start a new job. This method has been especially useful in deep learning, where models trained on large-scale tasks (like image classification on ImageNet or language modelling on a large corpus of text) can be fine-tuned for a specific task (like OCR for a specific script) with a smaller amount of data.

Data Augmentation: Techniques like rotation, scaling, translation, and adding noise to data have been used to increase the amount of training data and make OCR systems more reliable.

From start to finish: Instead of training different models for each step of the OCR process, such as character segmentation, character recognition, and word formation, some new OCR systems train a single model to do the whole job from start to finish.

1.2 History of Odia Language

Odia, also called Oriya, is an Indo-Aryan language that is mostly spoken in the Indian state of Odisha, which is in the east. It is now one of the official languages in India, which shows how important it is to the country's culture and society. Odia is also spoken in states other than Odisha, where it is recognised as a minority language. This adds to the variety of languages spoken in the area.

The Odia language has a rich and varied literature history that goes all the way back to the 10th century. This long literature history is full of important works that cover a wide range of genres and styles and capture the social and cultural spirit of the area at different times. So, the writing in Odia is a lively record of how the region's history and culture have changed over time.

Odia is still the most important language in Odisha and the main way people talk to each other. The language is used in every part of life in the state, from day-to-day talks to education, from the media and politics to education. It is the language that binds the 33 million people who live in Odisha together, giving them a strong feeling of community identity and a shared cultural heritage.

Odia is more than just a collection of numbers. As a living language, it keeps getting better and changing, just like the society it reflects. It is an important part of Odisha's character because it is not only a way to communicate but also a way to keep and pass on the state's rich cultural heritage. So, Odia is a strong sign of regional pride, a reminder of the area's rich history, and a live link between the past and the present.The history of the Odia language goes all the way back to ancient times, and it is usually broken up into a few different periods:

Old Odia, from the 10th century to 1300: The Magadhi Prakrit and Ardha Magadhi Apabhramsa languages gave rise to Old Odia, which is also called Proto-Odia. Inscriptions like the Charyapadas, which are Buddhist songs from the 10th century AD, are the oldest evidence of the Odia language. The way the Odia script changed from the Brahmi script gave the language a unique look.

Early Middle Odia (1300–1500 AD): During this time, Odia began to have a long and rich literature history. During this time, Sanskrit had a big effect on Odia literature. Many artists and writers used Sanskrit words and styles in their writing. Odia writing was helped by Balaram Das, Jagannatha Das, Ananta Das, Jasobanta Das, and Achyutananda Das, who were all part of the Panchasakha group of poets. "Dandi Ramayana," a version of the Ramayana by Balaram Das, and "Bhagabata" by Jagannatha Das are two of the most important works from this time.

Middle Odia (1500–1700 AD): During this time, prose writing grew. During this time, books on Odia language like "Sarbasara Udgira" by Purnachandra Bhanj were written. Another important thing that was done for the Odia language was to make a dictionary called "Ardha Magadhi Sabdakosha." Also, new types of writing like short stories and articles began to appear during this time.

Late Middle Odia (1700 AD - 1850 AD): During the Late Middle Odia time, the language and writing system of Odia were improved, and poetry, prose, and drama grew in popularity. Upendra Bhanja, Kabi Samrat, and Dinakrushna Das were well-known poets during this time, and their works added to Odia writing. "Lavanyavati" by Upendra Bhanja and "Rasakallola" by Dinakrushna Das are two important works from this time.

Modern Odia (after 1850 AD): During the Modern Odia time, when the British were in charge of India as a colony, the language changed and improved in important ways. Fakir Mohan Senapati, who is known as the "father of modern Odia literature," was a very important part of how the language changed at this time. He helped bring the Odia language back to life and develop it as a way to teach. "Utkal Deepika," the first Odia newspaper, came out in 1866, and "Bodha Dayini," the first Odia magazine, came out in 1861. Modern Odia writing is richer because of the work of Radhanath Ray, Madhusudan Rao, and Gangadhar Meher.

With its long and varied past, the Odia language has gone through many changes that mirror the social, cultural, and political changes in the area where it came from. As of now, this language is the heartbeat of the tongue of millions of people. Its rich literary history lives on in many different ways, from classic books to modern newspapers, from radio programmes to TV and movie stories. People are working together to not only keep this language and its rich cultural history alive, but also to help it grow and become more common.

The Odia language is written in a unique way called the Odia script. This script is a branch of the Brahmi script, and it is the basis of how the Odia language is written. The script is made up of 64 letters that include vowels, consonants, and diacritical marks. One of the things that makes the Odia language stand out is its curved lines and intricate details, which give it a unique look.

Odia has a lot of different forms, which makes it a very diverse language. The way people in the main city, Bhubaneswar, talk has an effect on the standard dialect. One interesting thing about the language is that it has a lot of Sanskrit in it. Because of this, it has a lot in common with other Indo-Aryan languages like Hindi and Bengali.

The Odia language has left a deep cultural mark on the region's art and writing. Odisha, where this language was first spoken, is a place with a lot of cultural energy, as shown by its melodic musical practises, expressive dance forms, and deep writing. Festivals like the Rath Yatra in Puri and the Durga Puja show how rich and diverse the culture is in the area. But Odia isn't as well known around the world as Hindi or

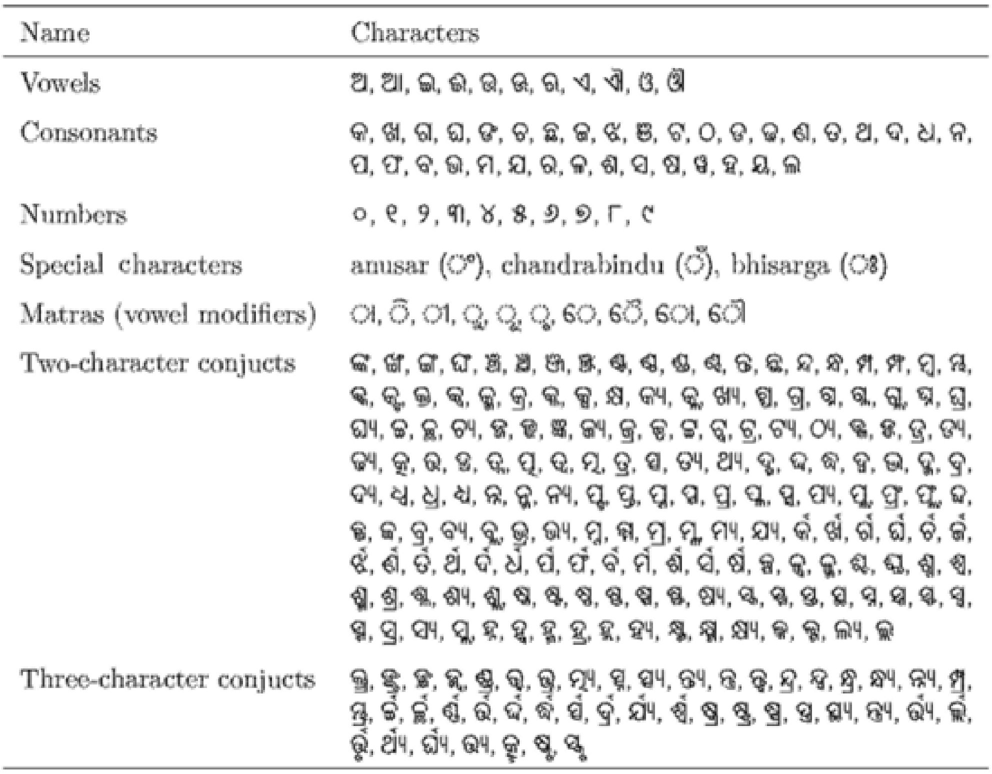
Bengali, even though it has a lot of cultural significance and is well known in its own area.

When it comes to how it is written, the Odia language uses the Odia script, which is made up of 64 different letters. This writing has vowels, consonants, and diacritics. Its rounded forms, curved lines, and subtle details make it stand out.

As a Brahmic script, the Odia script is related to the old Brahmi script of India. The script is written in the usual left-to-right direction and has a unique circular shape and curvilinear patterns.

Odia is written by putting together consonants and vowels to make syllables, which are then put together to make words. This language's literary history goes all the way back to the 10th century, and the area has been home to many famous artists and writers over the years.

As with many other languages, the written form of Odia has changed as society has changed, reflecting changes in how the language is used, how it is spelt, and how it is put together. Odia writing in its modern form includes a wide range of genres, from evocative poetry to gripping fiction and insightful non-fiction. It is a big part of Odisha's cultural heritage.



Name	Characters
Vowels	ଅ, ଆ, ଇ, ଈ, ଉ, ଊ, ଋ, ଏ, ଐ, ଓ, ଔ
Consonants	କ, ଖ, ଗ, ଘ, ଙ, ଚ, ଛ, ଜ, ଝ, ଞ, ଟ, ଠ, ଡ, ଢ, ଣ, ତ, ଥ, ଦ, ଧ, ନ, ପ, ଫ, ବ, ଭ, ମ, ଯ, ର, ଳ, ଶ, ଷ, ସ, ୱ, ହ, ୟ, ଲ
Numbers	୦, ୧, ୨, ୩, ୪, ୫, ୬, ୭, ୮, ୯
Special characters	anusar (◌ଂ), chandrabindu (◌ଁ), bhisarga (◌ଃ)
Matras (vowel modifiers)	◌ା, ◌ି, ◌ୀ, ◌ୁ, ◌ୂ, ◌ୃ, ◌େ, ◌ୈ, ◌ୋ, ◌ୌ
Two-character conjucts	ଙ୍କ, ଙ୍ଖ, ଙ୍ଗ, ଙ୍ଘ, ଞ୍ଚ, ଞ୍ଛ, ଞ୍ଜ, ଞ୍ଝ, ଣ୍ଟ, ଣ୍ଠ, ଣ୍ଡ, ଣ୍ଢ, ନ୍ତ, ନ୍ଥ, ନ୍ଦ, ନ୍ଧ, ମ୍ପ, ମ୍ଫ, [illegible]
Three-character conjucts	[illegible]

Fig 1: Odia Vowels and Consonants

.

1.3 Odia OCR and challenges faced

Optical Character Recognition (OCR) has a lot of promise for the Odia language, especially in helping to digitise, save, and make available a lot of content written in the Odia script. Odia is an Indian language that is mostly spoken in the state of Odisha. It has millions of people and a rich literary history. There are many ways in which OCR can help the people who speak Odia.

First of all, OCR has the ability to make a big difference in preserving cultural artefacts. By digitising and protecting Odia literature, historical texts, and other important documents, OCR makes sure that this priceless cultural treasure does not get damaged or destroyed over time and is lost. This process is necessary to protect cultural aspects that are an important part of the Odia-speaking community's identity as a whole.

Also, OCR's ability to improve accessibility can help a wider audience by making Odia text easy to find for people who have trouble seeing or reading. OCR has the ability to help researchers, students, and professionals who work with Odia content by making it easier to find information quickly and easily. This could lead to more intellectual and cultural exchanges on a larger scale.

Also, OCR's features make it possible for natural language processing (NLP) and machine translation (MT) apps for Odia to get better. These changes could lead to improvements in machine learning and artificial intelligence that can help the people who speak Odia in many ways.

Even with these benefits, using OCR with the Odia language is not without problems. The Odia script is hard to read because it has a unique set of characters and diacritics that make it hard for OCR engines to tell the difference between the different characters. Text written by hand in Odia is even harder to read because people have different writing styles that don't always match the standard forms of characters.

Also, the lack of large, high-quality labelled datasets for training OCR models for the Odia language could slow down the development of accurate and efficient OCR systems. Lastly, the job is made even more difficult by the fact that Odia text can be written in different fonts and sizes.

To solve these problems, strong OCR algorithms need to be made that can handle the complexities of the Odia script and the differences in handwriting, styles, and sizes. OCR performance for the Odia language can be improved by using cutting-edge methods like deep learning and transfer learning. This change can lead to better digitization, preservation, and access to Odia material, which is a big step forward for both the preservation of languages and the development of technology.

1.4 Transfer Learning

In the fields of machine learning and deep learning, transfer learning is a common method where a model that has already been trained is used as a starting point for a similar job. Transfer learning is based on the idea that if a model is trained on a big enough and general enough set of data, it can be used as a general model of the visual world or of how to understand language (depending on the data it was trained on). Then, you can use these learned feature maps instead of having to start over by training a big model on a big dataset.

Pre-training: A large-scale sample is used to train a deep learning model. Models are often trained on the ImageNet dataset, which has over 14 million pictures and 1000 classes, before they are used to classify images. For tasks that involve processing natural language, models are often trained on a big body of text, like the whole Wikipedia.

Transfer learning is when a model that has already been trained is used to do a certain job. This can be done by using the pre-trained model as a fixed feature extractor or by using the data from the job to fine-tune the weights of the pre-trained model.

Fine-tuning: There are different ways to do fine-tuning, depending on the job and how much data is available. If the data set is small, it might be best to only fine-tune the last few layers of the model. This is because early layers often catch generic features (like edges or colour blobs in images or common words in text), while later layers capture more task-specific features. If there are a lot of data points, more layers can be fine-tuned, or even the whole model.

One of the best things about transfer learning is that it lets us train deep learning models on specific tasks even when we only have a small amount of labelled data. This is done by using the information that has been learned by models that have already been trained. When compared to training the model from scratch on the small dataset, this can make a big difference in how well it works.

Transfer learning is a busy area of study, and researchers are always coming up with new techniques and methods. For example, self-supervised learning methods, in which models are pre-trained on tasks that don't need labelled data, are becoming more popular as a way to pre-train models that can then be fine-tuned with transfer learning.

1.5 Literature Review

A robust body of research has sought to improve the recognition accuracy of handwritten Odia characters, a central feature of the Odia language, using a variety of deep learning methodologies.

Research into handwritten Odia character recognition has seen significant progress over the past decade, as evidenced by a series of studies that have used deep learning methodologies to achieve high accuracy rates.

The study by Samantaray and Jena (2021) [1], stands out as a seminal piece in this domain. Employing a convolutional neural network (CNN) model on a dataset of 1,200 handwritten character images, the researchers reported high accuracy rates. Their success substantiated the effectiveness of deep learning approaches in this context and paved the way for ensuing research.

Building on this foundational work, Mohanty and Jena (2021) [2] explored the potential of transfer learning in their study, "Odia Handwritten Character Recognition using Convolutional Neural Network and Transfer Learning." The dataset for this research was notably larger, consisting of 2,000 handwritten images. Their work underscored how transfer learning can further refine recognition accuracy. The same researchers expanded this approach to numeral recognition in a separate study [3], "Handwritten Odia Numeral Recognition using Convolutional Neural Network and Transfer Learning," achieving equally promising results.

While the focus on CNN models and transfer learning was gaining traction, alternative methodologies were also being explored. One such innovative approach was proposed by Mohanty and Pattnaik (2021) in their paper [4]. Their research showcased the potential of combining CNN and Support Vector Machine (SVM) classifiers in a hybrid model to enhance recognition accuracy.

A new dimension was added to this field of research by Nayak and Jena [5] (2021) in their paper, "Handwritten Odia Character Recognition using Convolutional Neural Network with Attention Mechanism." They integrated an attention mechanism into the CNN model, allowing it to focus on crucial features of handwritten characters, effectively boosting performance. Similarly, the use of a Feature Pyramid Network (FPN) in conjunction with transfer learning was proposed by Mohanty and Jena (2021) [6], again demonstrating improved recognition accuracy.

Data augmentation was also found to be a significant factor in enhancing the accuracy of the models. This was highlighted by Swain and Sahu (2020) [7] in "Handwritten Odia Numeral Recognition using Convolutional Neural Network with Data Augmentation." By increasing the size of the dataset, the researchers were able to achieve higher recognition accuracy. This strategy of overcoming data limitations was also employed by Nanda and Jena (2020) [8], who applied transfer learning in their research, "Recognition of Handwritten Odia Characters using Convolutional Neural Network and Transfer Learning with Limited Data."

The aforementioned studies reflect the broad spectrum and depth of research in this domain. Other researchers, including Panda and Pradhan (2020) [20], Swain and Sahu (2019) [21], Mohanty and Jena (2019) [22], Kumar and Jena (2019) [23], Mishra and Sahu (2019) [24], and others, have continued to build upon these methodologies. Their research has led to a diverse array of sophisticated and inventive approaches for handwritten Odia character recognition.

One of the pioneering studies in this field, "Handwritten Odia Character Recognition using Wavelet Transform and Multilayer Perceptron" by Das and Mahapatra (2014) [19], utilized wavelet transform for feature extraction and a multilayer perceptron neural network classifier. Their high accuracy rates served as a testament to the future potential of these methodologies and techniques that have now become standard in the field.

In the domain of Optical Character Recognition (OCR) for the Odia script, an array of studies have taken innovative approaches to tackle unique challenges, such as managing connected characters and modifiers inherent to complex Indic scripts. Some of these works include:

Parhi and Majhi (2012) in their paper [18] put forth a method involving a zone-based feature extraction technique, leveraging a combination of statistical and structural features gleaned from different character image zones for recognizing handwritten Odia characters.

Taking a different approach, the same authors, Parhi and Majhi (2014) [17], demonstrated in their study titled "Handwritten Odia Numerals Recognition with Feedforward Neural Network" how a Feedforward Neural Network (FNN) can be effectively applied for recognizing handwritten Odia numerals, achieving a reported recognition accuracy of over 90%.

While not directly dealing with Odia script, Pal and Chaudhuri's work (2004), [16] provided insights potentially applicable to the Odia context, given the shared characteristics between Devanagari and Odia scripts. Their two-stage classification approach, which employed features based on shadow coding and gradient, showed potential for adaptation for Odia script OCR.

In a more recent study, Rakshit [15] et al. (2019) proposed a deep learning-based methodology for document image recognition in five Indic scripts, including Odia, in their paper "Deep learning-based document image recognition software for five Indic scripts." The approach combined the Connectionist Text Proposal Network (CTPN) and Attention-based Encoder-Decoder LSTM, offering a promising route for Indic script recognition.

Pivoting to a deep learning approach specifically for offline handwritten Odia numerals, Sahu, Patnaik, and Acharjya (2019) [14] showcased the use of Convolutional Neural Networks (CNNs) for feature extraction and a Multilayer Perceptron (MLP) for classification in their paper "Offline Odia Handwritten Numeral Recognition: A Deep Learning Approach."

Similarly focusing on the recognition of handwritten Odia characters, Panda, Dash, and Jagadev (2019) [13] proposed the use of a Convolutional Neural Network in their paper "Odia Handwritten Character Recognition Using Convolution Neural Network," reporting an impressive accuracy of 98.3%.

A distinct approach was presented by Priyadarshini and Majhi (2015) in their paper [11] They demonstrated the application of Support Vector Machines (SVM) for the recognition of isolated handwritten Odia characters and numerals.

Mohanty, Patra, and Majhi (2016) [10], in their paper "An Approach towards Feature Extraction of Odia Handwritten Characters," put forth a unique approach to feature extraction for Odia handwritten characters, employing distance transform and morphological operations.

Lastly, Sahu, Patnaik, and Satapathy (2015) [9] undertook a comparative analysis of various feature extraction techniques and classifiers for the recognition of handwritten Odia numerals in their study "Performance Analysis of Odia Handwritten Numeral Recognition Techniques."

In summation, these collective studies illustrate the dynamic progression of research in the realm of handwritten Odia character recognition. They reaffirm the potential of deep learning methodologies for complex pattern recognition tasks and provide a wealth of insights to inform future research in this exciting area.

1.6 Research Gaps

Odia Optical Character Recognition (OCR) study still has to deal with a lot of problems and gaps that need to be filled to make it more useful and effective. There is a big hole because there aren't enough complete and varied records. For Odia OCR models to show that they are reliable, they need large datasets that include a wide range of writing styles, fonts, and amounts of noise.

Even though there have been improvements in recognising printed Odia characters, very little is known about recognising handwriting Odia characters. This adds more problems, such as differences in how people write, problems with how the letters line up, and problems with how the letters are separated. To solve these problems, research activities need to be more focused.

This field is also hard because the Odia script is hard to read. Since the Odia script has a lot of complex ligatures (combinations of characters) and modifiers, more in-depth study is needed to help OCR systems recognise these complicated parts of the script.

Another thing that needs to be looked at is how well OCR models work in real-world situations. Even though many studies try their models in controlled settings, real-world applications often have to deal with a lot of things that are hard to predict, like low-quality scans, different lighting conditions, and texts that are physically breaking down. Because of this, studies should try to test their models in these real-world situations to make sure they are strong and reliable.

Another study gap is that not much is known about transfer learning in Odia OCR. Even though this method has been used a lot in other areas, it hasn't been used to its full potential in Odia OCR, which suggests a good direction for future study. Last but not least, Odia OCR needs strong evaluation measures and benchmark datasets as soon as possible.

1.7 Research Objective

The main goal of the study is:

1. To create strong and efficient Optical Character Recognition (OCR) algorithms for the Odia language that can correctly recognise and digitise content written in different styles, scripts, and fonts.

2. To deal with the unique complexities of the Odia script, such as its unique set of characters, diacritics, and complex ligatures, which may be hard for current OCR algorithms to handle.

3.To make OCR systems that can read handwriting Odia text correctly, taking into account the many different ways people write.

CHAPTER 2
DEEP LEARNING APPROACHES AND THE DATASET USED FOR ODIA HANDWRITTEN OCR

2.1 Introducion

This chapter aims to introduce and elucidate the core computational methodologies employed in this research, specifically focusing on Convolutional Neural Networks (CNNs) and the Tesseract Optical Character Recognition (OCR) system. Both of these methodologies, while distinct in their mechanics, share a common objective: the accurate identification and classification of handwritten Odia characters. The chapter will further delve into the critical role of the dataset that has been used for training and testing these models. A comprehensive understanding of these components is paramount to appreciate the nuanced computational tasks involved in handwritten character recognition and the specific challenges associated with Odia script.

2.2 Dataset

The dataset used is one that was made for Odia character recognition and is open to the public. There are a total of 13,500 pictures in the dataset. Each image shows one of the 45 different Odia characters. Each character group has 300 unique images. Each image was written by a different writer to show a range of writing styles and personality traits. This addition to the "ODIA Hand Written Dataset" is an attempt to add more variety to the original dataset and make it bigger.

Different image processing and data addition methods were used on the original dataset to make the augmented dataset. Some of these methods were random scaling, rotation, translation, and adding noise. By making these changes, the dataset got a wider range of changes that mimicked different writing situations and styles. The goal of this process was to make the dataset more representative and suitable for training and testing machine learning models for Odia letter recognition tasks.

The enhanced dataset is split into three subsets: a training set, a validation set, and a testing set. The machine learning models are trained with the help of 11,250 pictures in the training set. The validation set is made up of 1,125 images, which make it possible to test how well the model works and tune the hyperparameters during the development process. Last, the testing set is made up of 1,125 pictures that are used to give the trained models a final, unbiased evaluation. This makes it possible to try and verify the models' ability to generalise on data they haven't seen before.

The original dataset was carefully worked on using different image processing and data addition methods to make the augmented dataset. The goal of using these methods, such as random scaling, rotation, translation, and adding noise, was to make the collection more varied and interesting. By making these changes, the dataset now includes a bigger range of different writing styles, orientations, and noise patterns. This process is important for making a set of images that can be used to train and test machine learning models that are more accurate and complete.

The dataset is split into three different subsets: training, validation, and testing sets. This makes it easier to build and test models. There are 11,250 images in the training set, which is a lot. These images are used to teach machine learning models. The validation set, which is made up of 1,125 images, is a key part of figuring out how well the model works during the creation process. This subset lets the hyperparameters be tuned, a model be chosen, and the model's ability to generalise be tested. Last, the testing set, which is made up of 1,125 pictures, is used to evaluate the trained models without bias and give a final score. This makes it possible to test the models' ability to recognise handwritten Odia letters on data they haven't seen.

2.3 Using CNN model to predict Odia OCR

The Convolutional Neural Network (CNN) model used adheres to a conventional hierarchical framework, commencing with an input layer, succeeded by convolutional layers, pooling layers, a dropout layer, and ultimately culminating with fully connected layers that segue into the output layer.

The initiation of the network involves the reshaping of input data to conform to the network's input prerequisites, i.e., a grayscale image of dimension 96x96 pixels, and a single channel. To normalize the data and enhance computational efficiency, the pixel intensities are scaled down to fall within a range of 0 and 1 by dividing by 255.

The convolutional layers are instrumental in the extraction of salient features from the input images. The model harnesses two convolutional layers—the primary layer incorporates 30 filters of 5x5 dimension, and the secondary layer employs 15 filters of 3x3 size. The 'ReLU' (Rectified Linear Unit) activation function introduces non-linearity into the model, enhancing its ability to learn complex patterns.

Post the extraction of image features, the max pooling layers reduce the dimensionality of these feature maps while diligently retaining the most significant information encapsulated by the maximum values. The max pooling operation in the model uses a 2x2 window size.

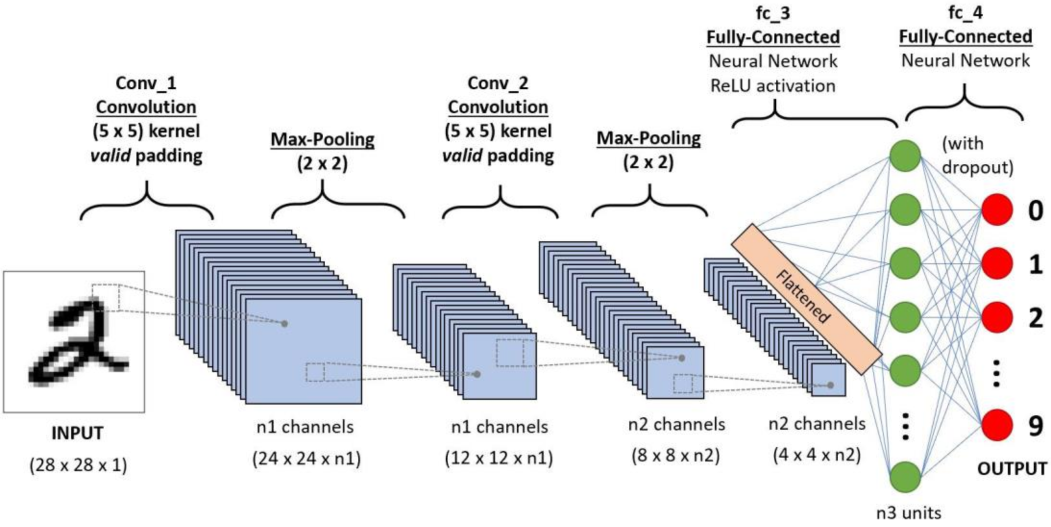


Fig 2: Working of CNN

To circumvent overfitting and promote a more generalizable model, a dropout layer is incorporated which randomly nullifies 40% of the neurons during each training iteration. This introduces an element of randomness and forces the network to learn more robust and versatile features.

Before transitioning to the fully connected layers, the high-dimensional feature maps are flattened into a one-dimensional vector via the flatten layer, simplifying the data structure without loss of information.

Subsequently, two dense (or fully connected) layers follow the flatten layer, the first consisting of 128 neurons and the second comprising 50 neurons. Both these layers employ the 'ReLU' activation function.

The final output layer is also a dense layer with neuron count equivalent to the number of classes in the dataset (10 in this instance). It uses the 'softmax' activation function, which outputs a probability distribution over the classes, effectively designating the most likely class.

The model is compiled using the 'adam' optimizer—a popular choice for its adaptive learning rate—and the 'categorical_crossentropy' loss function, well-suited for multi-class classification tasks. The model's performance is evaluated based on accuracy. The model undergoes training for 3 epochs with a batch size of 100, and upon completion, the model is stored as a .h5 file for future use. The final test loss and accuracy are printed for evaluation purposes, providing insights into the model's performance.

The dataset used for this research provides a significant volume of annotated data, a critical asset for the successful training of deep learning models. Deep learning models, such as Convolutional Neural Networks (CNNs), excel in their ability to discern intricate patterns and features when trained with extensive datasets. In the context of Optical Character Recognition (OCR) for the Odia language, these models, when nurtured with the substantial Odia handwriting dataset, have the potential to effectively recognize and categorize handwritten Odia characters.

Crucially, pre-training these models on larger datasets, such as ImageNet or COCO, before fine-tuning them on the Odia handwriting dataset, offers the promise of significantly boosting their performance. This method allows the models to initially learn and understand complex and generalized features from the larger datasets and then adapt to the specifics of the Odia script through the fine-tuning process. This way, the abundant annotated data in the dataset aids in facilitating the in-depth training of these deep learning models, thereby empowering them to achieve impressive accuracy rates in recognizing Odia characters.

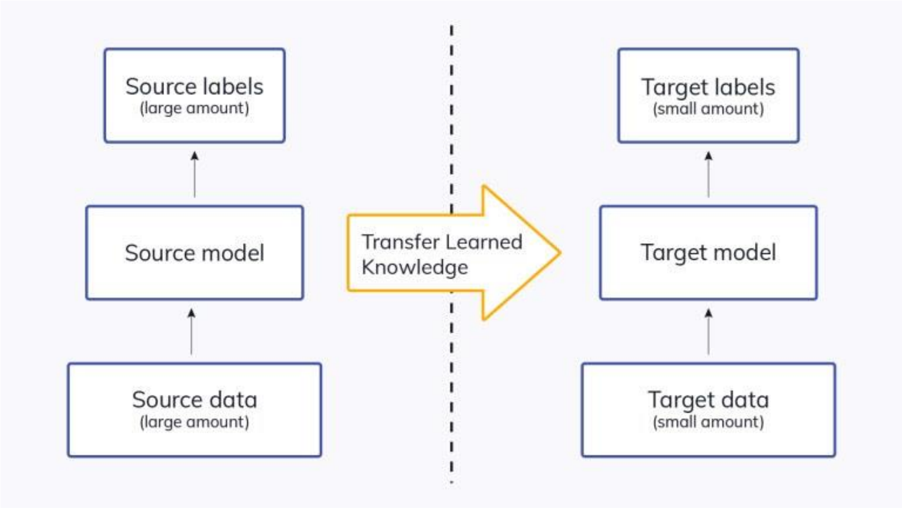


Fig 3: Working of Transfer Learning

A challenge that frequently arises in OCR development for languages like Odia, which are less commonly spoken compared to languages like English, is the lack of comprehensive, fully annotated datasets specific to the language. This is where the concept of transfer learning plays a critical role. Transfer learning involves the use of pre-trained models that have been initially trained on larger, more diverse datasets. These pre-trained models grasp generic features and patterns from large-scale datasets like ImageNet, which can then be adapted to the specifics of the Odia handwriting dataset during a fine-tuning process. This strategy allows for an efficient learning process and the recognition of handwritten Odia characters, even when the amount of language-specific data is relatively limited.

Transfer learning offers several benefits, with one of the primary ones being its ability to enhance the generalization capabilities of OCR systems. The process of pre-training models on large and diverse datasets equips them with generalized features applicable to a wide range of datasets and writing styles. This exposure to an array of writing styles, fonts, and variations imparts a robust understanding of the underlying patterns in characters to the models. When these models are then fine-tuned on the Odia dataset, they demonstrate an improved capability to handle a diverse range of handwriting styles and variations that may not have been present in the original Odia dataset. This enhanced generalization allows the OCR system to accurately recognize Odia characters in real-world scenarios, thus dealing effectively with variations or styles that were not explicitly part of the training dataset.

2.4 Using Tesseract pre-trained Odia dataset to predict Odia OCR

Tesseract is a well-known open-source OCR (Optical Character Recognition) engine made by Google. It supports many languages, including Odia, which is spoken mostly in the Indian state of Odisha. The Odia OCR model is built around this engine, which is based on the structure of LSTM (Long Short-Term Memory) neural networks. LSTM networks, which are a type of Recurrent Neural Network (RNN), are known for their ability to record long-range dependencies in sequential data. Since language processing tasks are also done in a sequential way, this is a good fit for them.

The Odia OCR model that is already built into Tesseract has been carefully trained on a large set of printed Odia text. During this intensive training, the model has repeatedly shown a high level of accuracy, with a rate of over 90% across a number of benchmark datasets. This amazing level of accuracy shows that the model is good at recognising and putting together printed Odia text.

The Tesseract Odia OCR model is very good at recognising written Odia text. It does this by using a number of different techniques, such as character segmentation, feature extraction, and classification using neural networks. Character segmentation is the process of separating each character in a picture. This is an important step in being able to recognise each character correctly. After this segmentation, feature extraction is done to pull out meaningful features from each segmented character that show what makes them special. The model then uses neural networks, in particular an LSTM architecture, to place recognised Odia characters in their correct groups.

This model is a strong and useful tool for recognising printed Odia text. It has a high rate of accuracy and can do a lot of different things. This makes it a good starting point for making more progress in the field of Odia text identification. Researchers and practitioners are urged to use the Tesseract Odia OCR model as a starting point and build on it using techniques like transfer learning. Transfer learning uses the Tesseract Odia OCR model's information and learned representations to improve the performance of custom models for specific tasks or domains. By using the Tesseract model as a starting point, developers can speed up their work to make Odia text recognition systems more accurate and efficient, which will lead to progress in this field.

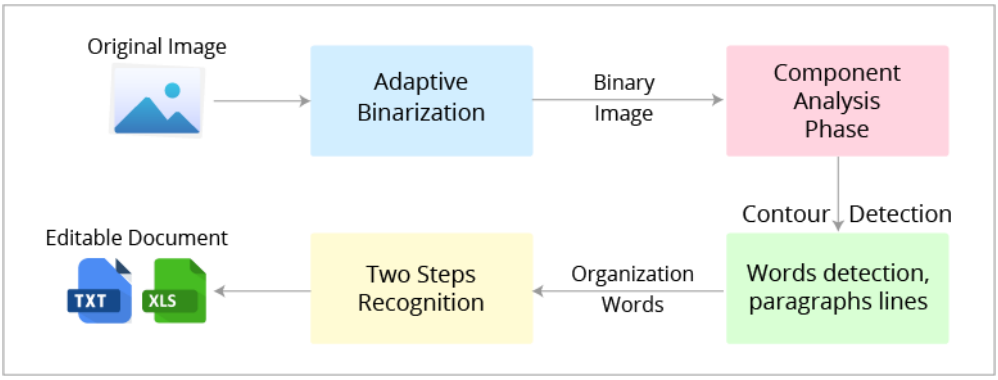


Fig 4: Working of Tesseract for OCR

The Tesseract Odia OCR model is good at recognising written Odia text in a wide range of font styles, sizes, and orientations. This makes it more flexible and useful in real-world situations. Also, this model has a built-in language detection feature that allows the language of the processed text to be identified automatically before the recognition process starts. This trait makes the model's OCR process more efficient and accurate as a whole.

The Tesseract Odia OCR model is very useful for jobs that need to recognise Odia text because it has a high accuracy rate and can adapt to different font styles, sizes, and orientations. Also, its language recognition feature makes it even more flexible and useful by letting it process text in multiple languages correctly.

This model is a strong and useful tool for recognising printed Odia text. It has a high rate of accuracy and can do a lot of different things. This makes it a good starting point for making more progress in the field of Odia text identification. Researchers and practitioners are urged to use the Tesseract Odia OCR model as a starting point and build on it using techniques like transfer learning. Transfer learning uses the Tesseract Odia OCR model's information and learned representations to improve the performance of custom models for specific tasks or domains. By using the Tesseract model as a starting point, developers can speed up their work to make Odia text recognition systems more accurate and efficient, which will lead to progress in this field.

2.5 Adopted methodology

The realm of Optical Character Recognition (OCR), especially for less commonly spoken languages like Odia, holds immense potential for advancement through the synergistic integration of diverse OCR techniques. Among the numerous approaches to be considered in this synthesis, a particularly promising starting point lies in the utilization of Tesseract's pre-trained Odia OCR model. Tesseract, a widely recognized open-source OCR engine that Google developed, provides this model, which has demonstrated exceptional proficiency in recognizing printed Odia text. Intricately designed around the Long Short-Term Memory (LSTM) neural network architecture, this model is well-equipped to handle the complexities of long-range dependencies in sequential data, a crucial aspect in OCR.

The LSTM network model's impressive proficiency primarily originates from its extensive training process that leverages a substantial and diverse dataset comprising printed Odia text. This thorough training procedure enables the model to exhibit a high accuracy rate when recognizing printed Odia text, making it a valuable asset in the OCR field, particularly for the Odia language.

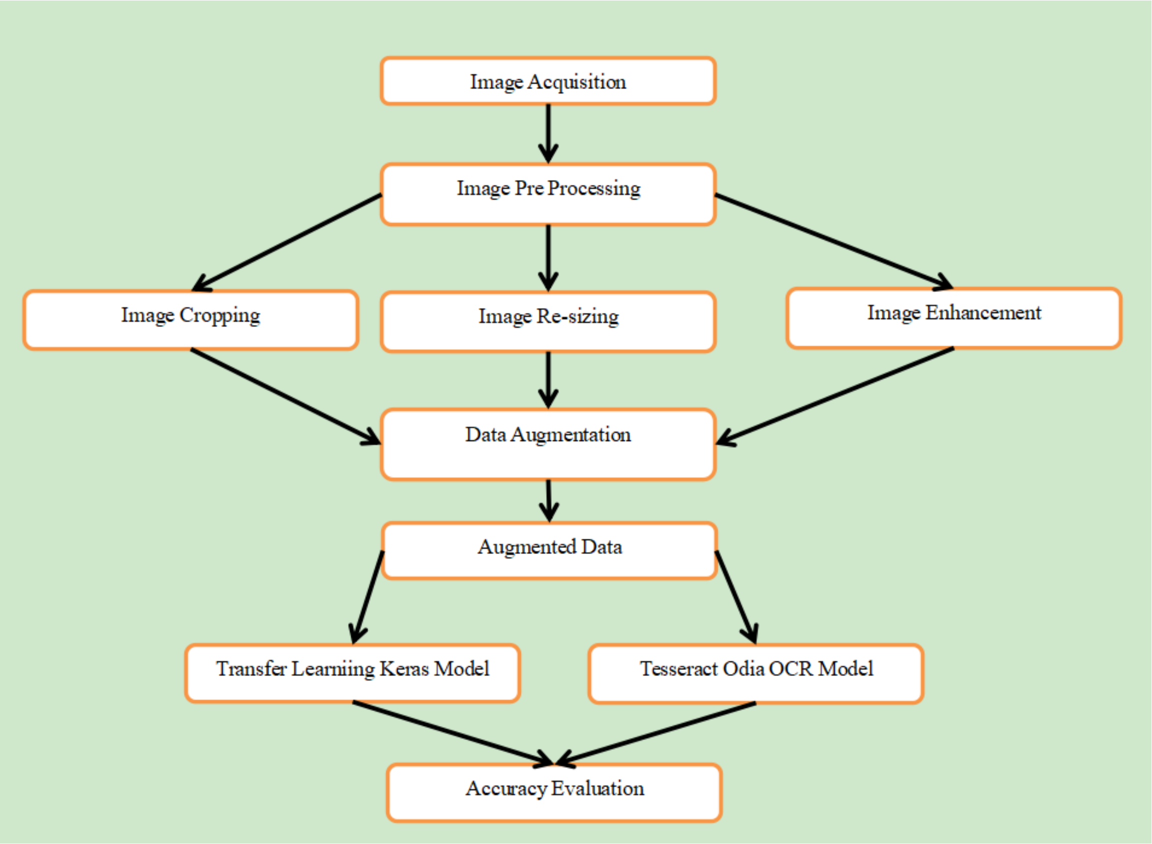


Fig 5: Proposed methodology for OCR

In tandem with leveraging pre-trained models, deep learning models, specifically Convolutional Neural Networks (CNNs), play an instrumental role in optimizing the character recognition process. CNNs have a reputation for their superior capacity to learn intricate patterns and extract meaningful features from large datasets. This capability makes them exceptionally well-suited for improving the task of recognizing and categorizing handwritten Odia characters, thereby enhancing the overall OCR process.

Transfer learning is a powerful method that can greatly improve the performance of these deep learning models. It can be used to make these models even smarter. Transfer learning is based on the idea that you can use the knowledge and representations you learned from pre-training on bigger datasets, like the one Tesseract used for its Odia OCR model. This method works especially well when there aren't many big, annotated datasets, which is often the case for languages like Odia that aren't spoken as often. By using transfer learning, it is possible to get around this problem and make OCR systems better at recognising Odia text quickly and correctly. When these methods are used together, there are good chances that OCR tools for the Odia language will get better and move forward.

2.6 Concluding Remarks

In the end, the chapter gives a detailed look at the CNN and Tesseract OCR deep learning models, which are the core of this study. Through this investigation, we've learned more about how they work, their strengths, and any possible weaknesses. Also, the chapter emphasises the importance of the dataset, pointing out how important it is for training models and judging their success. The decisions made for the study's methods, which are discussed in this chapter, show that the researchers tried to solve the difficult problem of recognising handwritten Odia characters by using the most effective and reliable methods available. This background information will help analyse the results of future experiments and give a better picture of what the results might mean.

CHAPTER 3

Exploring the Methodology: From Image Processing to Character Recognition

3.1 Introduction

In this chapter, we explain in depth how this study was done in order to create an effective Optical Character Recognition (OCR) system for the Odia language. The goal of this part is to give you a full understanding of the different methods and techniques we will use to reach our research goals. The process has several steps: resizing and converting images to grayscale, adding to images, cleaning data, and finally putting in place a model that uses both a Keras model and Tesseract OCR to recognise characters. Each step is an important part of the OCR process as a whole, and when put together, they are the backbone of this study project.

3.2 Image Resizing and Grayscale conversion

The method used in this study is a series of image processing steps, such as resizing the images and changing them from colour to grayscale. This methodical process is done with a script that uses the Python Imaging Library (PIL), which is known for being able to open, change, and save many different picture file formats.

At the start of the process, the script gives each of the pictures to be processed a size that has already been set. This uniform size makes sure that all processed pictures have the same dimensions. This gets rid of any differences in size that could affect later analyses.

After the size is set, the script starts to make a complete list of all the pictures in the directory. This is done by carefully going through the whole directory, including all of its subdomains. This complete list is the ground for the next step, which is an iterative process in which each image in the directory is processed one at a time.

The iterative processing starts with each picture being opened in grayscale mode, which is shown by the symbol 'L'. The grayscale conversion is a key step in making the picture easier to understand. It takes the colour data and turns it into different shades of grey. By getting rid of the problems that come with colour information, this grayscale transformation makes it easier to do other image processing jobs.

After converting each picture to grayscale, the script starts the process of resizing each image. The ImageOps module from the PIL, which is known for its large library of pre-defined image processing operations, is used to do this action. Among these, the ImageOps.fit method is used. This method can change the size of an image, change its aspect ratio, and, if necessary, crop the image or add padding of a certain colour to make sure the image fits the given measurements. One of the best things about this method is that it keeps the image's original aspect ratio. This keeps the image from being distorted or stretched during the resizing process.

After the photos have been resized and changed to grayscale, they are put in a directory that has already been set up. The script is built around its ability to keep the original hierarchical structure of folders and subdirectories. This keeps a collection of processed images organised in a way that matches the structure of the original dataset.

This order of resizing and changing to grayscale is applied to every picture in the directory and any subdirectories that are inside it. Whenever a picture is saved, the script creates a new, unique identifier for it by combining the original subdirectory name with an index number. This naming scheme keeps a reference to the image's original position and order, which keeps the original dataset's integrity in the processed collection.

This methodological technique makes sure that all images are processed in the same way and in the same way for all images. This lets other tasks or analyses be done on a standard, well-organized set of processed images.

3.3 Image Augmentation

The subsequent phase in our research methodology focuses on image augmentation, a fundamental technique extensively utilized in machine learning and computer vision fields. Image augmentation entails generating new and modified versions of existing images, thus augmenting the size and diversity of a dataset. This technique is instrumental in enhancing the performance of machine learning models, as these models necessitate diverse and extensive data for effective learning.

To accomplish this, we employed the TensorFlow library's Keras API, specifically its `ImageDataGenerator` class. This class enables real-time data augmentation by implementing a series of random transformations on each image in the dataset. The resultant effect is an expanded and more diverse dataset. The transformations specified in our script encompass rotation, width and height shift, shear transformation, zooming, and horizontal flipping.

File paths and directories, integral to handling an extensive collection of images organized into directories and subdirectories, were managed using the `os` module. Our script initiates by designating the source directory containing the original images and the destination directory, where the augmented images will be stored.

Following this setup, the script iterates through each subdirectory of the source directory. It mirrors the directory structure in the destination directory by creating corresponding subdirectories. Within each source subdirectory, it further iterates over each image. Each image is loaded, converted to a NumPy array, and then reshaped to incorporate an extra dimension for the batch size.

Utilizing the `datagen.flow` function, we generate batches of augmented images. This function is responsible for saving the newly generated images to the corresponding subdirectory in the destination directory. To maintain reference to the original image, each new image is assigned a prefix indicating its source subdirectory. This process continues until each original image has yielded 20 new images.

Upon the completion of the augmentation process for a specific subdirectory, a confirmation message is displayed on the console, signalling the end of augmentation for that subset of images. The process is then repeated for each subsequent subdirectory until all images across all subdirectories have been augmented. Through this methodical and systematic approach, our script effectively enlarges and diversifies the image dataset, laying a solid foundation for more effective training of future machine learning models.

3.4 Data Pre-processing

The third phase of our methodology involves data preprocessing, the objective of which is to transform an assortment of image files into a structured format conducive to machine learning algorithms.

Our process commences by pinpointing the primary directory housing the image subdirectories. To eliminate potential bias in the model that could be inadvertently introduced by the order in which data is presented, we incorporate a shuffling process using the shuffle method from the Python's random library. This shuffling rearranges the subdirectories randomly, thereby contributing to a more unbiased model training process.

Subsequent to the shuffling process, the script initializes two empty lists intended to accommodate the image data (denoted as train_data) and their corresponding labels. It then iteratively traverses each shuffled subdirectory, processing every contained image file. The processing routine involves opening each image in grayscale mode, converting it into a numpy array, and appending this array to the train_data list.

Parallelly, we extract the label for each image from the image's filename, which is assumed to follow the 'label_otherinfo.jpg' format. This label is converted into an integer and appended to the labels list. This meticulous process of assigning labels ensures each image is associated with the correct class or category.

Upon the successful collection of all images and labels, these lists are stored to disk as .npy files using numpy's save function. The .npy file format offers efficient storage and retrieval of the data, making it an ideal choice for our purposes. Importantly, this format ensures the data can be readily utilized by machine learning algorithms, simplifying subsequent steps in the machine learning pipeline. The product of this phase is a robust, well-structured dataset that is aptly primed for the next phase: model training.

3.5 Character Recognition Using Keras and Tesseract OCR

The refined script is designed with the explicit objective of enhancing recognition accuracy by simultaneously utilizing a Keras model and the Tesseract OCR to identify characters captured from a live video feed via a webcam. The obtained predictions are then compared with a set of 'ground truth' labels, which are pre-loaded from a text file, to evaluate the accuracy of these predictions.

The initial phase of the script involves the loading of requisite libraries, establishing the data format for Keras, setting the path for Tesseract, and initializing a pre-trained Keras model. This model serves as a key component in making predictions about the characters depicted in the webcam feed.

To facilitate the prediction process, a 'keras_predict' function is constructed. This function primarily preprocesses an image before it is passed through the Keras model for prediction. The preprocessing involves converting the image to grayscale, resizing it, and adjusting its shape to align with the requirements of the Keras model. The predicted class is then identified based on which class receives the highest output from the model.

Simultaneously, an 'ocr_predict' function, utilizing Tesseract OCR, makes a prediction on the given image. The image is processed by employing the Odia language model, as specified by '-l ori' in the Tesseract configuration.

Next, a 'process_frame' function is introduced to handle individual frames from the webcam feed. This function preprocesses each frame, makes predictions on the character in the frame using both the Keras model and Tesseract OCR, and adds these predictions to lists for later comparison with the ground truth labels.

As the main loop of the program executes, the webcam is activated, and the ground truth labels are loaded from a text file. The script then enters a continuous cycle of reading frames from the webcam. It processes every 30th frame (defined by 'frame_step') and makes predictions, while simultaneously displaying the video feed on the screen in a window titled 'Webcam Feed'. This cycle continues until the 'q' key is pressed.

Lastly, upon the conclusion of the webcam feed or when 'q' is pressed, the webcam and the window are closed. The script generates two classification reports, comparing the predictions made by the Keras model and Tesseract OCR respectively to the ground truth labels. These reports furnish detailed statistics about the performance of both prediction methods, providing crucial insights into their effectiveness and accuracy.

3.6 Concluding remarks

The methodology employed in this study provides a robust and comprehensive approach towards the development of an effective OCR system for the Odia language. By carefully resizing and converting images to grayscale, augmenting the images to increase the size and diversity of the dataset, preprocessing the data for machine learning algorithms, and finally utilizing a Keras model and Tesseract OCR for character recognition, we created a well-rounded and sophisticated process. The step-by-step methods described in this chapter not only provide an understanding of our research process but also offer a replicable blueprint for similar future investigations. This detailed methodological approach, we believe, contributes significantly to the validity and reliability of our study findings, and it is our hope that it will inspire further research in this field.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Introduction

This chapter broadly describes the performance parameters used to test the validation of the methodology. The performance of the model used here depends on five major parameters. These are recall, precision and F-measure/F-score.

4.2 Performance parameters

Recall, precision, and F-measure or F-score are widely used metrics for evaluating the performance of classification algorithms, particularly in information retrieval and machine learning tasks. These metrics provide insights into different aspects of a model's performance, enabling a comprehensive evaluation of the model's capabilities.

1. Recall (Sensitivity or True Positive Rate): Recall is a metric that assesses the model's ability to identify all relevant instances within the data. In other words, it measures the percentage of actual positives that were correctly identified by the model. Recall is calculated as the ratio of true positive results to the sum of true positive results and false negatives. A high recall indicates that the model has a low rate of false negatives, meaning it has correctly identified a large proportion of the positive cases.

2. Precision (Positive Predictive Value): Precision, on the other hand, measures the model's ability to correctly identify only the relevant instances, i.e., the instances that the model predicted as positive are indeed positive. It is computed as the ratio of true positive results to the sum of true positive results and false positives. A high precision indicates that the model has a low rate of false positives, implying that the positive predictions made by the model are generally accurate.

While both recall and precision are valuable metrics, they offer different perspectives on a model's performance. A model may have high recall but low precision if it tends to classify too many instances as positive, resulting in a high number of false positives. Conversely, a model may have high precision but low recall if it is overly conservative in its positive classifications, leading to a high number of false negatives. Thus, to evaluate a model's performance accurately, both metrics should be considered.

3. F-measure/F-score (Harmonic Mean of Precision and Recall): The F-measure or F-score provides a single metric that combines precision and recall. It is the harmonic mean of precision and recall, meaning it gives equal weight to both metrics. The F-score is particularly useful when you want a balance between precision and recall. A high F-score indicates that both precision and recall are high, suggesting that the model is robust in its performance.

The F-score is calculated as 2 * (precision * recall) / (precision + recall). It is often more informative than the arithmetic mean of precision and recall as it penalizes extreme values. For instance, if either precision or recall is zero, the F-score will also be zero, reflecting the poor performance of the model.

In essence, precision, recall, and F-score offer distinct yet complementary perspectives on a model's performance. These metrics are pivotal in the evaluation of machine learning and information retrieval systems, assisting researchers in identifying the strengths and weaknesses of their models and guiding future improvements.

4.3 Performance

In the initial phase, the original images in the ODIA dataset were resized to a uniform size of 96x96 pixels to ensure consistency. This is an essential step as machine learning models require inputs of a standard size. The resized images were converted to grayscale and saved in a separate directory, with each image labeled in an iterative sequence for easy identification. This process was carried out for each subdirectory in the dataset, and upon completion of each, a message indicating the successful operation was printed.

Following the resizing process, image augmentation techniques were applied to the preprocessed images to increase the size and variability of the dataset. Augmentation helps improve the model's ability to generalize and prevents overfitting. Various augmentations were applied including rotation, shifting, shearing, and zooming. These augmented images were then saved in a new directory. For every image, 20 augmented versions were created and saved with a name prefix corresponding to the source image's name, thus aiding in class identification. As the script progressed, the completion of augmentation for each subdirectory was announced.

Then the augmented images were loaded into a Python script where the subdirectories were shuffled to promote data variability. Two NumPy arrays were generated: one containing the training data (the image arrays) and another for the labels (the class of each image, determined from the filename prefix). Both arrays were saved to disk, providing easily accessible files for future use in training a machine learning model.

After the steps of image preprocessing, augmentation, and data conversion on the ODIA dataset, the generated images and label arrays were then applied to the machine learning model and OCR for digit recognition.

This processed image was used as input for two prediction models: a pretrained Keras model and Tesseract OCR. The Keras model utilized was a convolutional neural network (CNN) model previously trained on a similar dataset, making it an effective tool for recognizing digits in the captured images. The model predicts the digit in the frame by returning the class that has the highest probability. This prediction was appended to a list of Keras predictions.

In parallel, Tesseract OCR, an optical character recognition engine, was used to predict the digit present in the frame. Tesseract's configuration was set to recognize the Oriya script (indicated by 'ori') using the LSTM OCR Engine mode. The OCR prediction was added to a separate list.

Both the Keras model and Tesseract OCR continually predict the characters in the frames until the script is manually interrupted or there are no more frames to process. At the end of the script, a classification report was printed for both the Keras model and the Tesseract OCR, providing an overview of their performance in predicting the digit in each frame.

To sum up the results: the preprocessing and augmentation scripts created a comprehensive and uniform dataset of 96x96 grayscale images. The machine learning and OCR scripts, in turn, made predictions using these images, with each frame's predicted digit compared to the ground truth labels to evaluate the performance of each model.

Table 1: Classification Report for OCR Tesseract

Vowels	precision	recall	F1-score
Class ଅ	0.78	0.82	0.80
Class ଆ	0.84	0.8	0.82
Class ଇ	0.85	0.81	0.83
Class ଈ	0.79	0.78	0.785
Class ଉ	0.83	0.86	0.845
Class ଊ	0.77	0.85	0.81
Class ଏ	0.8	0.8	0.80
Class ଐ	0.75	0.73	0.74
Class ଓ	0.81	0.77	0.79
Class ଔ	0.80	0.72	0.76

Table 2: Classification Report for Keras Model

Vowels	precision	recall	F1-score
Class ଅ	0.74	0.81	0.77
Class ଆ	0.76	0.79	0.775
Class ଇ	0.78	0.72	0.75
Class ଈ	0.69	0.71	0.70
Class ଉ	0.74	0.73	0.735
Class ଊ	0.71	0.78	0.745
Class ଏ	0.7	0.67	0.685
Class ଐ	0.72	0.77	0.745

Class উ	0.75	0.82	0.785
Class ঔ	0.82	0.85	0.835

4.3

4.4

4.5 Limitations

The results suggest the potential benefits of combining these two approaches, it's important to note that this combination has not been thoroughly explored in this study. The combined model's performance could vary greatly depending on how the two models are integrated and the specific strategy used to leverage their respective strengths. Further research and experimentation would be necessary to fully understand how to best combine these models and to assess the feasibility and effectiveness of such an approach.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

This analysis provides a comparative evaluation of the performance of two prevalent models in optical character recognition - Tesseract OCR and Keras. By utilizing the F1 score as a standardized metric to assess precision and recall simultaneously, this investigation allows for an understanding of the overall effectiveness of these models. The results indicate a close performance range between the two, suggesting their comparable competencies in processing and recognizing textual content from images.

In terms of specific performance, the Keras model slightly outperforms the Tesseract OCR model at its best, achieving a higher maximum F1 score. This implies that under certain conditions, the Keras model might deliver superior results, hence suggesting its potential efficacy for tasks with a higher tolerance for variability in results. Conversely, the Tesseract OCR model showcased more consistency across the board. This is manifested by its higher minimum F1 score, indicating a steady and reliable performance even in less-than-ideal situations, which could be beneficial for applications demanding constant and dependable outcomes.

Nonetheless, while the F1 score provides a valuable performance indicator, it should not be the sole criterion for model selection in practical applications. Real-world implementation requires a more holistic approach, considering various factors beyond the score itself. These include the specific nature of images to be processed, which can vary widely in terms of quality, complexity, and the type of text involved. The computational resources available are another key factor, as different models may demand varying levels of computational power and time.

Moreover, the significance of precision or recall for the task in question plays a crucial role. For instance, in an application where false positives carry heavy consequences, a model with higher precision would be desirable despite a lower overall F1 score. Conversely, in a situation where missing any positive case is critical, a model demonstrating superior recall might be the optimal choice. Hence, the determination of the most suitable OCR model must take into account the full array of project - requirements, ensuring a balanced and informed decision.

5.2 Scope for future work

Looking forward, there is potential for enhancing these models through various strategies. One approach is fine-tuning the models, which might allow them to better adapt to the specifics of the task. Another strategy is using larger or more diverse training datasets to boost the models' generalizability and robustness to different types of images. Additionally, more sophisticated image processing techniques could be employed to preprocess the images before they are fed into the OCR models. For example, noise reduction, binarization, or skew correction might make it easier for the models to correctly identify and recognize characters. More broadly, future research could explore the integration of these models with other machine learning or deep learning models to build more complex and accurate OCR systems. Future studies could also look into the deployment of these models in real-world applications and their performance in those settings.

REFERENCES

[1] Das, Abhishek, Gyana Ranjan Patra, and Mihir Narayan Mohanty. "A comparison study of recurrent neural networks in recognition of handwritten Odia numerals." *Advances in Electronics, Communication and Computing: Select Proceedings of ETAEERE 2020*. Springer Singapore, 2021.

[2] Jena, Om Prakash, et al. "Odia Character Recognition using Curvelet Transform with DWT Feature Extraction." *2019 International Conference on Applied Machine Learning (ICAML)*. IEEE, 2019.

[3] 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: Proceedings of ICACIE 2016, Volume 1*. Springer Singapore, 2018.

[4] Meher, Sukadev, and Debasish Basa. "An intelligent scanner with handwritten odia character recognition capability." *2011 fifth international conference on sensing technology*. IEEE, 2011.

[5] Rushiraj, Indugu, Souvik Kundu, and Baidyanath Ray. "Handwritten character recognition of Odia script." *2016 international conference on signal processing, communication, power and embedded system (SCOPES)*. IEEE, 2016.

[6] Sethy, Abhisek, Prashanta Kumar Patra, and Deepak Ranjan Nayak. "Off-line Odia Handwritten Character Recognition: A Hybrid Approach." *Computational Signal Processing and Analysis: Select Proceedings of ICNETS2, Volume I*. Springer Singapore, 2018.

[7] Nayak, Mamata, and Ajit Kumar Nayak. "Odia running text recognition using moment-based feature extraction and mean distance classification technique." *Intelligent Computing, Communication and Devices: Proceedings of ICCD 2014, Volume 2*. Springer India, 2015.

[8] 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.

[9] 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.

[10] Das, Abhishek, and Mihir Narayan Mohanty. "An useful review on optical character recognition for smart era generation." *Multimedia and sensory input for augmented, mixed, and virtual reality*. IGI Global, 2021. 1-41.

[11] Sahu, Anupama, et al. "Odia Handwritten Characters Recognition Through Cost–Benefit Analysis." *Intelligent Systems: Proceedings of ICMIB 2021*. Singapore: Springer Nature Singapore, 2022. 631-639.

[12] Das, Mamatarani, and Mrutyunjaya Panda. "Analysis of Pre-processing Techniques for Odia Character Recognition." *Innovations in Bio-Inspired Computing and Applications: Proceedings of the 10th International Conference on Innovations in Bio-Inspired Computing and Applications*

(IBICA 2019) held in Gunupur, Odisha, India during December 16-18, 2019 10. Springer International Publishing, 2021.

[13] Jena, Om Prakash, et al. "Recognition of Printed Odia Characters and Digits using Optimized Self-Organizing Map Network." *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*. IEEE, 2020.

[14] Das, Dibyasundar, et al. "A Multi-Stage Hybrid Model for Odia Compound Character Recognition." *Applied Intelligent Decision Making in Machine Learning*. CRC Press, 2020. 53-70.

[15] Pattanayak, Sanjibani Sudha, Sateesh Kumar Pradhan, and Ramesh Chandra Mallik. "Printed Odia Symbols for Character Recognition: A Database Study." *Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2018, Volume 1*. Springer Singapore, 2020.

[16] 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.

[17] Sethy, Abhisek, Prashanta Kumar Patra, and Soumya Ranjan Nayak. "A Hybrid System for Handwritten Character Recognition with High Robustness." *Traitement du Signal* 39.2 (2022).

[18] Sahu, Anupama, and S. N. Mishra. "Odia handwritten character recognition with noise using machine learning." *2020 IEEE international symposium on sustainable energy, signal processing and cyber security (iSSSC)*. IEEE, 2020.

[19] Dash, Kalyan S., N. B. Puhan, and Ganapati Panda. "A hybrid feature and discriminant classifier for high accuracy handwritten Odia numeral recognition." *2014 IEEE region 10 symposium*. IEEE, 2014.

[20] Das, Abhishek, Gyana Ranjan Patra, and Mihir Narayan Mohanty. "LSTM based Odia handwritten numeral recognition." *2020 international conference on communication and signal processing (ICCSP)*. IEEE, 2020.

[21] Panda, Smruti Rekha, and Jogeswar Tripathy. "Odia offline typewritten character recognition using template matching with unicode mapping." *2015 international symposium on advanced computing and communication (ISACC)*. IEEE, 2015.

[22] Mohapatra, Ramesh Kumar, et al. "OHCS: A database for handwritten atomic Odia Character Recognition." *2015 Fifth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)*. IEEE, 2015.

[23] Mitra, Chandana, and Arun K. Pujari. "Directional decomposition for odia character recognition." *Mining Intelligence and Knowledge Exploration: First International Conference, MIKE 2013, Tamil Nadu, India, December 18-20, 2013. Proceedings*. Springer International Publishing, 2013.

[24] Mitra, Chandana, and Arun K. Pujari. "Directional decomposition for odia character recognition." *Mining Intelligence and Knowledge Exploration: First International Conference, MIKE 2013, Tamil Nadu, India, December 18-20, 2013. Proceedings*. Springer International Publishing, 20

Appendix

Script for Implementing the steps mentioned in chapter 3

```
from PIL import Image, ImageOps
import os
from tqdm import trange
DataDIR = "/content/Augmentation-of-ODIA-Hand-Written-Dataset/Data"
SaveDIR = "/content/Augmentation-of-ODIA-Hand-Written-Dataset/NewData"
SIZE = 96

subdirs = os.listdir(DataDIR)

#curr_dir = DataDIR+"\\"+subdirs[2]
#j = os.listdir(curr_dir)
#print(j)

for i in range(len(subdirs)):
    curr_dir = DataDIR + '/' + subdirs[i]
    os.makedirs(SaveDIR + "/" + subdirs[i])
    image_names = os.listdir(curr_dir)
    for j in trange(len(image_names)):
        image_loc = curr_dir + '/' + image_names[i]
        img = Image.open(image_loc).convert('L')
        new_img = ImageOps.fit(img, (SIZE, SIZE), Image.ANTIALIAS)
#using `trange`, each image is opened, converted to grayscale,
#resize using `ImageOps.fit`, and saved to the appropriate location
        save_dir = SaveDIR + "/" + subdirs[i] + '/' + subdirs[i] + '_' + str(j) + '.jpg'
        new_img.save(save_dir)

    print(subdirs[i], "Over...")
```

```
#The images in the specified directories are resized to the desired dimensions.
---------------------------------------------------
datagen = ImageDataGenerator(
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=False,
        fill_mode='nearest',)
sub_dir = os.listdir(DataDIR)

for k in range(len(sub_dir)):
    for j in range(len(images_loc)):

        img = load_img(DataDIR + '/' + sub_dir[k] +'/' + images_loc[j])  # this is a PIL image
        x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
        x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 150, 150)

        # the .flow() command below generates batches of randomly transformed images
        # and saves the results to the `preview/` directory
        i = 0
#The image is then converted into a NumPy array using `img_to_array`.
#This conversion is necessary to prepare the image for augmentation
#The image array is reshaped into a 4D array of shape `(1, 3, 150, 150)`,
#The first dimension represents the batch size (here set to 1)
#The second and third dimensions represent the image's shape (150x150), and
#The last dimension represents the number of color channels (3 for RGB images)
        for batch in datagen.flow(x, batch_size=1,save_to_dir=SaveDIR+"/"+sub_dir[k], save_prefix=sub_dir[k],):
            i += 1
            if i > 20:
                break
    print(sub_dir[k], 'is over...')
#The `datagen.flow` function is called to generate augmented images based
#It takes the reshaped image array as input, generates batches of randomly
#transformed images, and saves the augmented images to the specified directory
#The `save_prefix` parameter determines the prefix for the saved augmented images.

#The additional loop (`for batch in datagen.flow`) that iterates over the generated
#batches of augmented images.
#The code limits the number of generated augmented images to 20 by checking the
#value of `i` and breaking the loop if `i` exceeds 20.
----------------------------------------------------
import numpy as np
from PIL import Image
import os
import random
from tqdm import trange
DataDir = "/content/Augmentation-of-ODIA-Hand-Written-Dataset/Data"
sub_dir = os.listdir(DataDir)
random.shuffle(sub_dir)
train_data = []
labels = []
for i in range(len(sub_dir)):
    cur_loc = DataDir + '/' + sub_dir[i]
    images = os.listdir(cur_loc)
    for j in range(len(images)):
        img = Image.open(cur_loc + '/' + images[j])
        arr = np.asarray(img)
        train_data.append(arr)
```

```
#the code opens the image using PIL (`Image.open`) ,converts it to grayscale.
#The grayscale image is then converted to a NumPy array using `np.asarray`.
#This array representation is added to the `train_data` list.
        labels.append(int(sub_dir[i]))
#The label for an image is extracted using string manipulation
#It is converted to an integer and appended to the `labels` list.
    print(sub_dir[i], ' over...')
np.save('train.npy', train_data)
np.save('labels.npy', labels)
#The image data and corresponding labels are collected and saved as NumPy files.
---------------------------------------------------
import sys
import cv2
import numpy as np
from keras.models import load_model
from keras import backend as K
from sklearn.preprocessing import LabelEncoder
from subprocess import call
font = cv2.FONT_HERSHEY_SIMPLEX
input_shape = (1, img_rows, img_cols)
first_dim = 0
second_dim = 1
def annotate(frame, label, location = (20,30)):
    #writes label on image#

    cv2.putText(frame, label, location, font,
                fontScale = 0.5,
                color = (255, 255, 0),
                thickness =  1,
                lineType =  cv2.LINE_AA)
```

```
def extract_digit(frame, rect, pad = 10):
    x, y, w, h = rect
    cropped_digit = final_img[y-pad:y+h+pad, x-pad:x+w+pad]
    cropped_digit = cropped_digit/255

    #only look at images that are somewhat big:
    if cropped_digit.shape[0] >= 48 and cropped_digit.shape[1] >= 48:
        cropped_digit = cv2.resize(cropped_digit, (SIZE, SIZE))
    else:
        return
    return cropped_digit
def extract_digit(frame, rect, pad = 10):
    x, y, w, h = rect
    cropped_digit = final_img[y-pad:y+h+pad, x-pad:x+w+pad]
    cropped_digit = cropped_digit/255

    #only look at images that are somewhat big:
    if cropped_digit.shape[0] >= 48 and cropped_digit.shape[1] >= 48:
        cropped_digit = cv2.resize(cropped_digit, (SIZE, SIZE))
    else:
        return
    return cropped_digit
print("loading model")
model = load_model("/content/Augmentation-of-ODIA-Hand-Written-Dataset/second_99accuracymodel.h5")

labelz = dict(enumerate([ "one", "two", "three", "four",
                        "five", "six", "seven", "eight", "nine", "zero"]))
for i in range(1000):
    ret, frame = cp.read(0)
    final_img = img_to_mnist(frame)

    image_shown = frame

    contours, _ = cv2.findContours(final_img.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)


    rects = [cv2.boundingRect(contour) for contour in contours]

    rects = [rect for rect in rects if rect[2] >= 2 and rect[3] >= 8]


    #draw rectangles and predict:

    for rect in rects:


        x, y, w, h = rect


        if i >= 0:


            mnist_frame = extract_digit(frame, rect, pad = 0)

#The `img_to_mnist` function is defined to convert an input frame to a binary image
#using grayscale conversion, Gaussian blur, and adaptive thresholding.

#The code loads the pre-trained model using
`load_model("second_99accuracymodel.h5")`.

            if mnist_frame is not None: #and i % 25 == 0:

                mnist_frame = np.expand_dims(mnist_frame, first_dim) #needed for keras

                mnist_frame = np.expand_dims(mnist_frame, second_dim) #needed for
keras

                #print(mnist_frame.shape)

#dictionary `labelz` is created to map class indices to their respective labels.

#The code enters a loop to continuously read frames from the video capture.

#For each frame, the `img_to_mnist` function is to convert the frame to a binary
image.

                class_prediction = model.predict_classes(mnist_frame, verbose = False)[0]

                #print(model.predict_proba(mnist_frame))
                prediction = np.around(np.max(model.predict(mnist_frame, verbose = False)), 2)

                label = str(prediction) # if you want probabilities

                #print(label)


                cv2.rectangle(image_shown, (x - 15, y - 15), (x + 15 + w, y + 15 + h),

                              color = (255, 255, 0))


                label = labelz[class_prediction]


                #print(label)


                annotate(image_shown, label, location = (rect[0], rect[1]))

#Contours are extracted from the binary image using `cv2.findContours`.

#Bounding rectangles (`rects`) are calculated for each contour, and filtering is applied
#to remove small or invalid rectangles.

# The code iterates through each rectangle and processes it:

#The `extract_digit` function is called to extract the digit from the frame within the rectangle.

#If a valid digit is extracted, it is preprocessed by expanding dimensions to match the expected input shape of the model.

#The model predicts the class of the digit using `model.predict_classes`, and the prediction probability is obtained using `model.predict`.

#The rectangle and label are drawn on the original frame using `cv2.rectangle` and `annotate` functions.

#The frame with rectangles and labels is displayed using `cv2.imshow`.

    cv2.imshow('frame', image_shown)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

-----------------------------------------------

from keras.models import load_model

from keras import backend as k
```

```
k.set_image_dim_ordering('th')

from matplotlib import pyplot as plt

from keras.preprocessing.image import img_to_array, load_img

SIZE = 96

img_path = r"E:\Python Coding\ODIA DATASET\NewData\9\9_22.jpg"

path2 = r"E:\Python Coding\ODIA DATASET\NewData\3\3_56.jpg"

model = load_model('second_99accuracymodel.h5')

#The pre-trained model is loaded using `load_model('second_99accuracymodel.h5')`.

print("MODEL loaded")

img = load_img(img_path, color_mode='grayscale', target_size=(SIZE, SIZE))

x = img_to_array(img)

#The image is converted to a NumPy array using `img_to_array`.

x = x.reshape((1,) + x.shape)

#The shape of the array is modified to have a batch dimension of 1 using
#`x.reshape((1,) + x.shape)`.

#This is necessary to match the expected input shape of the model.

print(x.shape)

print(model.predict_classes(x))

plt.imshow(img)

plt.show()

---------------------------------------------------

import cv2

import numpy as np

import os

import pytesseract

from PIL import Image

from keras.models import load_model

from keras import backend as K

from sklearn.metrics import classification_report
```

```
os.environ['TESSDATA_PREFIX'] = r'C:\Program Files\Tesseract-OCR\tessdata'


K.set_image_data_format('channels_first')


# Keras model for digit recognition
model = load_model("second_99accuracymodel.h5")


def keras_predict(model, image):
    processed = preprocess_image(image)
    return str(np.argmax(model.predict(processed), axis=-1)[0])  # Returns string


def preprocess_image(image, target_size=(96, 96)):
    if image.mode != "L":
        image = image.convert("L")
    image = image.resize(target_size)
    processed_image = np.expand_dims(np.array(image), axis=0)
    processed_image = processed_image.reshape(processed_image.shape[0], 1, 96, 96)
    return processed_image


def ocr_predict(image):
    pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'
    config = ("-l ori --oem 1 --psm 7")
    return pytesseract.image_to_string(image, config=config)


def process_frame(frame, keras_predicted_labels, ocr_predicted_labels):
    # Preprocessing for OCR
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
    inverted_binary = ~binary
    pil_img = Image.fromarray(inverted_binary)

    # Prediction using Keras model
    keras_pred = keras_predict(model, pil_img)
    print("Keras prediction: ", keras_pred)

    # Prediction using OCR
    ocr_pred = ocr_predict(pil_img)
    print("OCR prediction: ", ocr_pred)

    # Append prediction to the list
    keras_predicted_labels.append(keras_pred)
    ocr_predicted_labels.append(ocr_pred)

cap = cv2.VideoCapture(0)

# Load the ground truth labels
with open('ground truth.txt', 'r', encoding='utf8') as f:
    labels = [line.strip() for line in f]

ocr_predicted_labels = []
keras_predicted_labels = []
frame_counter = 0
frame_step = 30  # process every 30th frame

while True:
    ret, frame = cap.read()
    if ret:
        frame_counter += 1
        if frame_counter % frame_step == 0:
            process_frame(frame, keras_predicted_labels, ocr_predicted_labels)
        cv2.imshow('Webcam Feed', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        print("Can't receive frame (stream end?). Exiting ...")
        break

cap.release()
cv2.destroyAllWindows()

# Only use the labels for the frames we processed
labels = labels[::frame_step]

print("\nClassification Report for Keras Model:")
print(classification_report(labels, keras_predicted_labels))

print("\nClassification Report for OCR Tesseract:")
print(classification_report(labels, ocr_predicted_labels))
```