School of Computer Science and Information Technology

Subject Name: Innovative Project

Subject Code: BTCS0509

SYNOPSIS REPORT

For the Academic Year 2021 - 2022

B.Tech

(Computer Science & Information Technology)

Vth Semester

Submitted To:
Mentor Name: Dr.Neha Gupta Mam
Designation: Associate Professor

Submitted By:
Name: Yash Gupta
Roll No: 2019BTCS088

CONTENTS

Chapter 1

INTRODUCTION

1.1 Abstract of the project

Hindi is the most widely spoken language in India, with more than **300 million speakers**. As there is no separation between the characters of texts written in Hindi as there is in English, the Optical Character Recognition (OCR) systems developed for the Hindi language carry a very poor recognition rate.
In this project we propose an OCR for printed Hindi text in **Devanagari script**, using **Convolutional Neural Network (ANN),** which improves its efficiency. One of the major reasons for the poor recognition rate is error in **character segmentation**.

1. The presence of touching characters in the scanned documents further complicates the segmentation process, creating a major problem when designing an effective character segmentation technique.

    - **Data Preprocessing**
    - **Character Segmentation**
    - **Feature Extraction**

Finally, **classification and recognition** are the major steps which are followed by a general OCR. The preprocessing tasks considered in the paper are conversion of gray scaled images to binary images, image rectification, and segmentation of the document´s textual contents into paragraphs, lines, words, and then at the level of basic symbols. The basic symbols, obtained as the fundamental unit from the segmentation process, are recognized by the neural classifier.
In this work, three feature extraction techniques-: histogram of projection based on mean distance, histogram of projection based on pixel value, and vertical zero crossing, have been used to improve the rate of recognition. These feature extraction techniques are powerful enough to extract features of even distorted characters/symbols. For development of the neural classifier, a back-propagation neural network with two hidden layers is used. The classifier is trained and tested for printed Hindi texts. A performance of approximately **95%** correct recognition rate is achieved.

1.2 Problem of Existing System

We took this problem statement from **manthan.nic.gov.in** under **Institution's Innovation Council** hackathon. We have to create an **OCR Engine** which has capabilities to perform OCR on **Hindi Handwriten Text** for **Bureau of Police Research & Development.**[Figure 1.1]

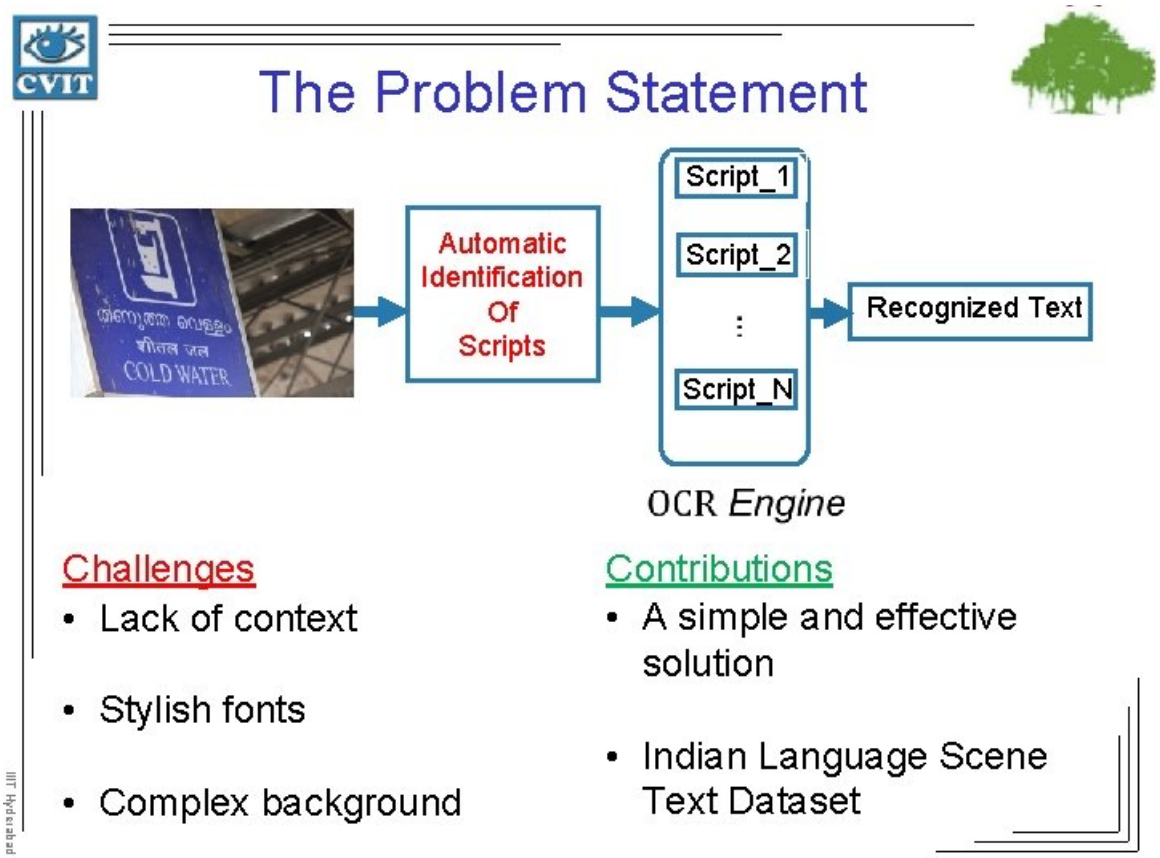

Figure 1.1: Problem Statement

1.3 Solution of the Problem

1. We took **'Devanagari Hindi'** dataset from **Kaggle** as well as real world Dataset from student Hindi Notebook.

2. As this is a **Classification** Problem, having **Multi-Class Logistic Regression.**

3. So we implemeted **Deep Learning** with the help of **CNN Algorithm.**

4. We used Multiple Python Libraries in our project:-

   - **Python v3.9**
   - **Anaconda Package Manager v2021.11**
   - **Jupyter Notebook v6.4.7**
   - **OpenCV Library v4.5.5**
   - **Keras Library v2.7.0**
   - **Pandas Library v1.4.0**
   - **Matplotlib Library v1.3.1**

Chapter 2

THE PROJECT

2.1 Project Definition

2.1.1 What is OCR?

Optical Character Recognition (OCR) is a process of converting printed or handwritten scanned documents into ASCII characters that a computer can recognize. In other words, automatic text recognition using OCR is the process of converting an image of textual docu- ments into its digital textual equivalent.
The advantage is that the textual material can be edited, which otherwise is not possible in scanned documents in which these are image files.
The document image itself can be either machine-printed or handwritten, or a combination of the two. Computer systems equipped with such an OCR system improve the speed of input operation, decrease some possible human errors and enable compact storage, fast retrieval and other file manipulations. The range of applications includes postal code recognition, automatic data entry into a large administrative system, banking, automatic cartography and, when interfaced with a voice synthesizer, reading devices for the visually handicapped.

2.1.2 Overview of OCR

An OCR system would usually perform the following tasks:-

1. Functions of an OCR Engine [Figure 2.1]
    - **Text Digitization**
    - **Gray Tone to Two Tone Conversion**
    - **Noise clearing**
    - **Text Block Identification**
    - **Skew Correction**
    - **Line and Word Detection**
    - **Character Segmentation**

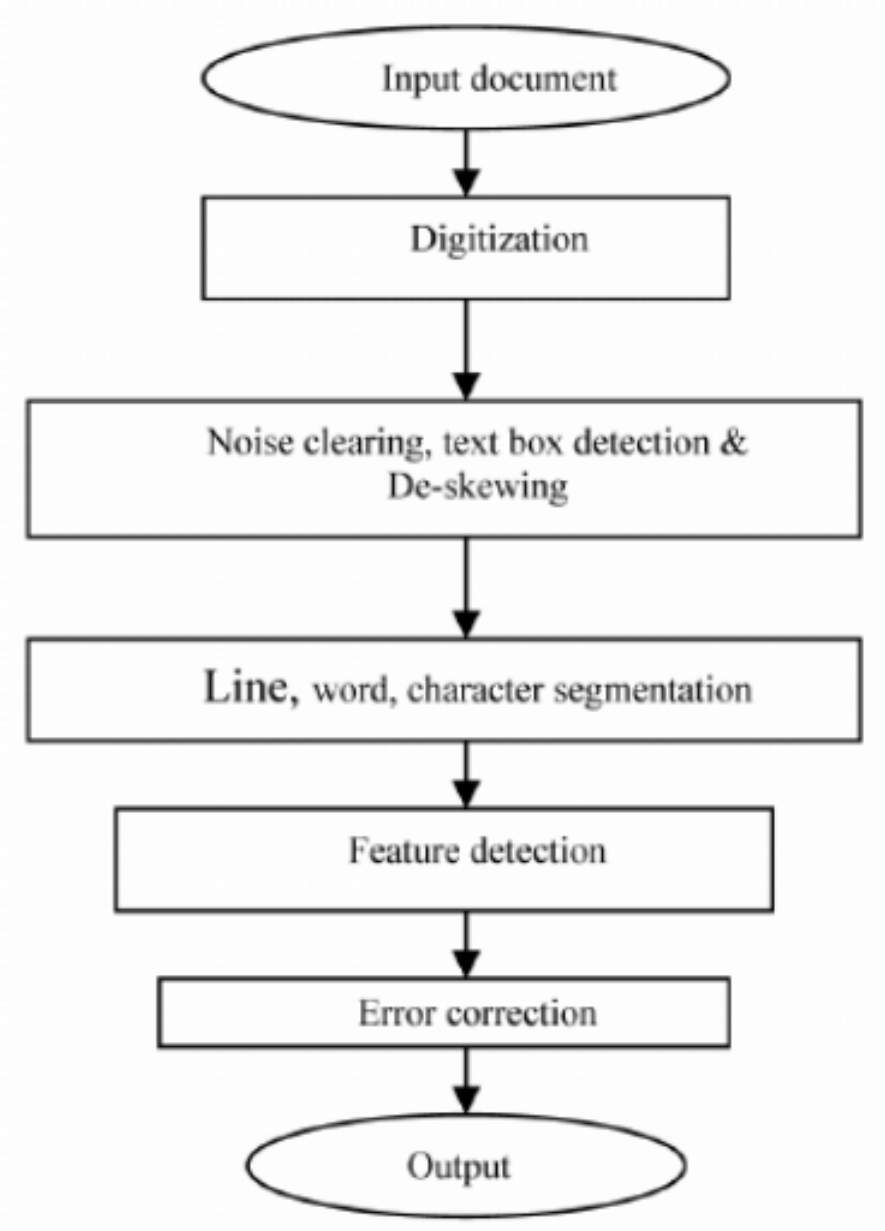

Figure 2.1: Flow control for Hindi OCR System

Chapter 3

SOFTWARE & HARDWARE REQUIRE.

3.1 Functional Requirements

These are described as a specification of behavior between inputs and outputs in our software lifecycle.

- **Project Scaffolding:** CookieCutter, Kedro
- **Documentation:** Sphinx, MikteX, MD5
- **CI and Deployment:** Jenkins, Docker, Gitlab
- **Data Modelling:** DBT
- **Data Exploration and Preparation:** Pandas (Pyspark if large)
- **Testing:** Pytest
- **Feature Store:** DVC, Feast
- **Workflow engine or orchestrator:** Luigi, Prefect, Airflow
- **Model Registry:** MLFlow (using Kedro-MLFlow or PipelineX)
- **Model serving:** FastAPI, BentoML, Cortex
- **Model monitoring:** Jenkins Pipelines, Prometheus, Graphana

Chapter 4

SYSTEM ANALYSIS

4.1 Architecture Design

The Architecture Design is very crucial for any software because the design principles, architectural decisions, and their outcome, i.e., software architecture together enable a software system to deliver its' business, operational, and technical objectives. Hence, we first consider following indications for our software architecture:

- The software is easy to maintain;
- Business stakeholders can understand it easily;
- Good software architectures are usable over the long-term;
- Such architecture patterns are flexible, adaptable, and extensible;
- It should facilitate scalability;
- The team can easily add features, moreover, the system performance doesn't diminish due to this;
- There is no repetition of the code;
- The system can be refactored easily.

We have used the best design patterns according to our need i.e.

1. Client-Server Pattern

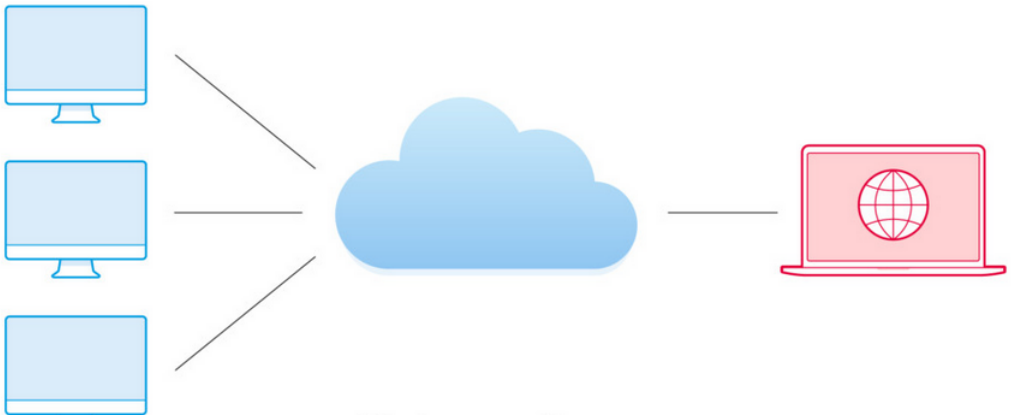

Figure 4.1: Client-server pattern

2. Master-Slave Pattern

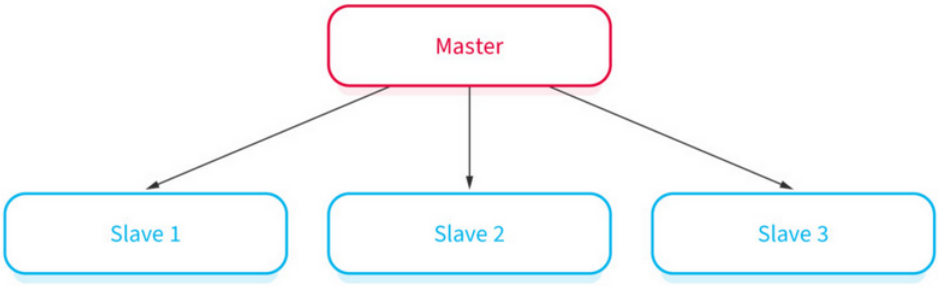

Figure 4.2: Master-slave pattern

3. Pipeline-Filter Pattern

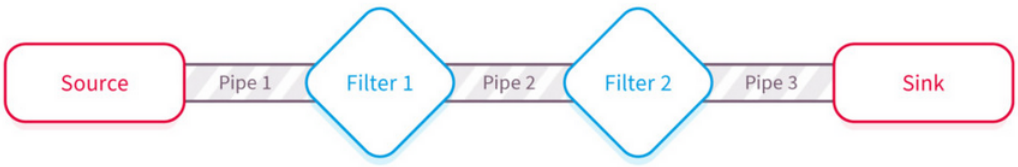

Figure 4.3: Pipe-Filter pattern

4.1.1 Level 0: Automation

The Machine Learning model design which was traditionally used can be referred from [Figure 4.4]

- **Level 0** looks very much like a portfolio or Kaggle project.
- There is little automation and limited options for data storage.
- If we wanted to retrain out model, we'd have to repeat all these steps again. This makes dealing with model drift difficult.
- If we wanted to repeat this with a different data or notebook, we'd have to copy code or adapt a notebook.
- We need additional tools to track our experiments when creating models.

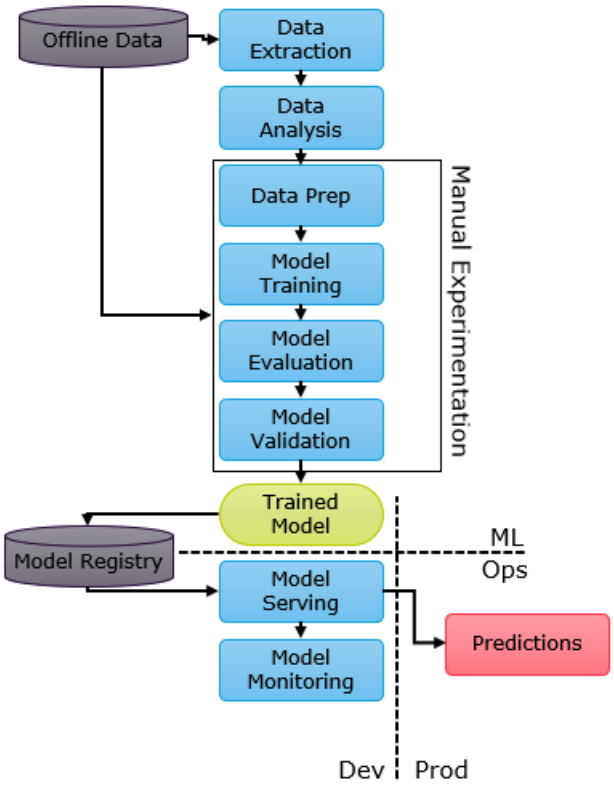

Figure 4.4: Level 0 MLOps

4.1.2 Level 1: High Automation

The Machine Learning model design we are using can be referred as **Level 1: Automation** [Figure 4.5]

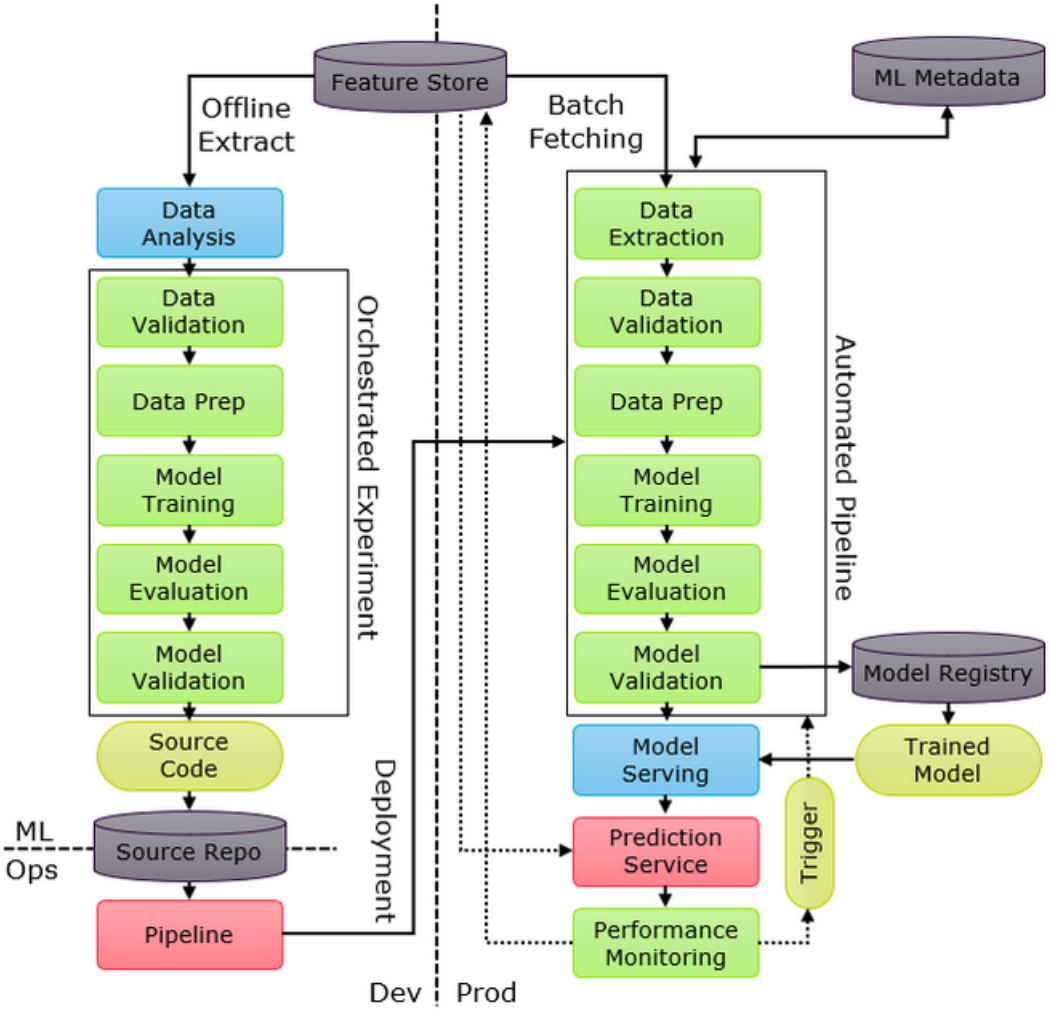

Figure 4.5: High Automation MLops

- The first new item here is a **feature store**. This is a database specifically for ML features. We do a variety of operations when training models such as scaling, feature engineering, encoding and mathematical transformations. Many of these features are not useful for typical analysis so a feature store allows us to store this and access it more easily.
- On the left side, we have **orchestrated** experiments. This is the concept of automation many operations in the Data Science work flow.
- The red square in the bottom left is the creation of a **pipeline**. This is another automation. When we deploy as pipeline, we are deploying the model and the transformations required to generate predictions from the model.
- Many validation steps can also be **automated**.
- We also allow **continuous monitoring and retraining** by collecting new data and storing it in the feature store.

4.2 Sequence Diagrams

Below figure [Figure 4.6] represents the workflow of how software works.

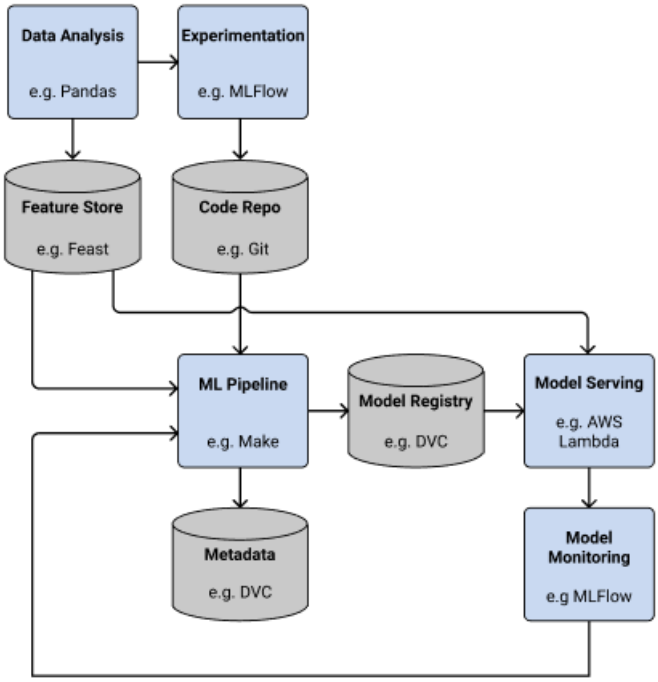

Figure 4.6: Sequence Diagram

Chapter 5

CONCLUSION

(Anticipatory approach)

In this project, we looked at developing an MLOps framework. The great thing about this is that we can plug tools into this framework as we find suitable tools. It also really helps with assessing new tools, as we can understand how they fit into our framework and any existing tooling.

Over the next few months, I plan to put this into practice. I'm going to try out new tools, link them together and try to get a genuine MLOps environment on my home workstation. If you have experience with any of the tools mentioned in this, leave a comment!

5.1 Problems and Issues in currents system

Problems

- Currently, no Major security tools are integrated with the DevOps Pipeline.
- Currently, Some processess are manual so we need to do automated those processes.

Issues

- Currently, the Infrastructure for production environment is not configured using via (IAC) approach which results in **More Time Consumption**
- Currently, No seperate **Testing environment** is created.

5.2 Future extension

Future extension will contain following things at the core for an efficient & highly secured software architecture i.e.

- Easy **Intersectionality** with least coupling.
- Easy **Multifunctionality**
- Best **Cost Optimization** of different cloud providers.
- Implementing different strategies for **High Scalability & High Availability**.
- Develop mobile application for controlling infrastructure.