COMPARATIVE ANALYSIS OF HANDWRITTEN CHARACTER RECOGNITION USING ML MODELS

LAU UNG HIEN

This project is submitted in partial fulfilment of

the requirements for the degree of Bachelor of Computer Science with Honours

(Computational Science)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE **COMPARATIVE ANALYSIS OF HANDWRITTEN CHARACTER RECOGNITION USING ML MODELS**

ACADEMIC SESSION: ____2024/2025____

LAU UNG HIEN

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

☐	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
☐	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
☑	UNRESTRICTED	

Validated by

Dr Lau Ung Hien
Senior Lecturer
Faculty of Computer Science and Information Technology
Universiti Malaysia Sarawak

(AUTHOR'S SIGNATURE) (SUPERVISOR'S SIGNATURE)

Permanent Address

6A, Jalan Kulas Lorong 3,
96000, Sibu, Sarawak

Date:____17/7/2025_____ Date: _17/7/2025_____

Note * Thesis refers to PhD, Master, and Bachelor Degree
** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

Declaration

I hereby declare that this report titled " Comparative Analysis of Handwritten Character Recognition Using ML Models" represents my own work, except where due acknowledgment is made. The content has not been submitted, either wholly or in part, for assessment in any other course or institution. All sources of information have been appropriately cited and referenced according to APA 7th guidelines.

Signature: 

Name: Lau Ung Hien

Student ID: 79822

Date: 23 Jun 2025

Acknowledgements

I would like to express my deepest gratitude to all those who contributed to the successful completion of this Final Year Project. This endeavour would not have been possible without their unwavering support, guidance, and encouragement. First and foremost, my sincere thanks go to my project supervisor, Dr. Ling Yeong Tyng for her invaluable expertise, patient guidance, and constructive feedback throughout every phase of this project. Their insightful suggestions and willingness to dedicate time and resources were instrumental in shaping this work. I am also grateful to the course lecturer of Artificial Intelligence, Dr Sarah Flora Anak Samson Juan who taught me the fundamentals of machine learning so that I could better apply this knowledge in model training. Special thanks go to friend, Mr Chan Choi Liang for their camaraderie, advice and motivation during challenging times. Our collaborative spirit made this journey enjoyable and rewarding. Last but not least, I would like to thank my beloved family for their encouragement and endless support, be it financial or emotional for me to complete this project.

Abstract

Handwritten Character Recognition (HCR) remains a pivotal research domain within Machine Learning (ML), intersecting computer vision, pattern recognition and human-computer interaction. Its applications spread as critical areas such as digitizing historical manuscripts, automated postal sorting, bank check processing, and assistive technologies. While numerous studies have explored HCR using diverse ML methodologies deep learning architectures (e.g., CNNs, RNNs, Transformers), and hybrid approaches. Each model presents distinct strengths and limitations in terms of accuracy, computational efficiency, robustness to variability (script, style, noise), and scalability. This study conducts a comprehensive comparative analysis of prominent ML-based HCR models, employing rigorous benchmarking across standardized datasets and diverse scripts. Performance is evaluated quantitatively (accuracy, precision, recall and F1-score) and qualitatively (error pattern analysis). As the result, the Character Error Rate of each model is as follow CNN(49.76%), CNN-LSTM (43.10%) and TrOCR (36.24%)

Table of Contents

List of Figures

List of Equation

List of Table

CHAPTER 1
INTRODUCTION

1.0 Overview

The chapter covers the outline the main content of Handwritten Character Recognition using ML models. It highlights the project background, problem statement, objective, Methodology, scope, significant of project, project schedule and expected outcome. This chapter presents the direction of this project to the reader.

1.1 Background

Handwritten character recognition (HCR) is a crucial area for study in Machine Learning (ML), Computer vision and pattern (Vinjit, Bhojak, Kumar, & Chalak, 2020). Its application is variously used in automated data processing such as document digitization, postal sorting, and online form processing. The main usage of HCR is to change human-written text into machine-readable characters, which can significantly reduce the workload of manual data entry and improve accessibility for digital records. In document digitization, HCR also help in transforming physical records into searchable digital files. For instance, it enables automated reading of addresses and zip codes in package sorting, and further streamlining delivery services (AlKendi W. , Gechter, Heyberger, & Guyeux, 2024).

Besides that, various studies have been conducted on handwritten character recognition (HCR) using different machine learning models. However, a comprehensive comparative analysis is required to evaluate these models systematically. Such an analysis will help identify the most effective techniques for HCR, particularly in terms of accuracy, efficiency, and applicability to diverse datasets and languages

1.2 Problems Statement:

The main problem that faced by Handwritten character recognition (HCR) is lacking a comparative analysis on their relative performance in handling style variability. Studies often focus on individual models like support vector machines (SVMs), Recurrent Neural Network (RNN) and Convolutional Neural Network, but few compare their strengths and weaknesses across various handwriting styles and noise conditions. (AlKendi W. , Gechter, Heyberger, &

Guyeux, 2024)Without such comparative insights, it is difficult to determine which models are best suited for HCR tasks and under what conditions they excel or falter. This gap in the literature indicates the need for a comprehensive analysis that evaluates the performance of multiple ML model s in HCR to identify optimal approaches for handling style variability and improving recognition accura**cy**

1.3 Objectives:

This study aims to rigorously assess the efficacy and robustness of prominent machine learning architectures specifically Convolutional Neural Networks (CNN), Long Short-Term Memory networks (LSTM), and Transformer-based models in handwritten character recognition (HCR). The overarching goal is to establish comparative benchmarks and characterize performance dependencies under diverse operational conditions through systematic evaluation.

The first primary objective is to establish baseline accuracy for each selected machine learning model. This will be accomplished by assessing the performance of models such as CNN, LSTM and Transformer model on a standardized handwritten character recognition (HCR) dataset. The evaluation aims to determine fundamental accuracy metrics, thereby providing a reliable benchmark against which further performance analyses can be compared.

The second primary objective is to examine the performance of these models under varying data conditions. This investigation will focus on how differences in handwriting style and varying levels of image noise affect each model's accuracy and robustness. By analyzing performance under these diverse conditions, the study seeks to identify the unique strengths and limitations of each machine learning model, ultimately contributing to a deeper understanding of their practical applicability in real-world HCR tasks.

1.4 Methodology:

This methodology of the project is done by four stage which are data collection, model selection, data pre-processing, and metrics evaluation for handwritten character recognition (HCR). (Monica & Shital P., 2015)

The first stage is identifying and selecting a suitable dataset of handwritten characters which collected large collections of handwritten samples with variety of handwritten style. A diverse dataset is essential, as it allows the selected ML models to be trained on a wide range of handwriting styles, which enhances the models' ability to generalize and accurately recognize characters despite variability and noise.

After selecting the dataset, suitable Machine learning models are analysed and chosen based on their strengths. For instance, Convolutional Neural Networks (CNNs), are known for their strength in complex pattern recognition, making them highly suitable for HCR tasks despite their higher computational demands. A selection of models is then chosen based on a balance between computational efficiency and anticipated accuracy for HCR.

Following model selection, data pre-processing is performed to enhance image quality and improve model accuracy. The pre-processing begins with binarization, where grayscale images are converted to binary (black and white) images, simplifying the task of distinguishing foreground text from background. Next, noise removal is applied to eliminate unwanted pixels that might interfere with character recognition. Techniques like Non-local Means, Anisotropic Diffusion, and image filters help smooth the image while preserving critical details. Next, normalization is used to standardize the shape and size of each character image, scaling pixel values to a fixed range, such as 0 to 1, which improves model consistency during training. Lastly, segmentation is done to separate each character individual for train. After pre-processing, the data is divided into training and validation/test partitions, where a larger portion is reserved for training, and a smaller partition is used to assess model performance.

The selected models are then trained on the training dataset and evaluated with the validation dataset. Several metrics are used to assess performance: accuracy measures the overall effectiveness of character recognition, while precision and recall provide insights into the model’s ability to avoid false positives and false negatives. F1-score combines precision and recall giving a balanced view, especially helpful with imbalanced datasets. A confusion matrix is generated to visualize classification performance for each character class, identifying any specific areas of misclassification. Additionally, processing speed and computational efficiency are considered, particularly for real-time applications, along with robustness testing where noise and style variability are added to the validation set to assess adaptability to real-world scenarios.

1.5 Scope

This project focuses on implementing and evaluating the performance of various machine learning models on handwritten character recognition. The dataset that will used are" Handwritten Recognition dataset from (Kaggle, n.d.) and IAM dataset from (Marti & H., 2002). Training model will be evaluated using various metrices such as accuracy, sensitivity, recall and F1-score. This project also need to develop a system for handwritten character recognition. Through develop a software , user can convert their handwritten document to digital document.

1.6 Significant of Project

One key significance is that the project bridges gaps in existing research by machine learning models in a systematic way, offering a comprehensive analysis of each model's strengths and weaknesses. This comparative evaluation provides a foundation for selecting the most suitable models for specific HCR applications and gives future researchers and developers a clearer understanding of trade-off between accuracy, computational efficiency, and adaptability among the models, such as strength and limitations of each different models.

Additionally, this project's findings have the potential to provide references to industries where handwritten text remains essential, such as healthcare, finance, and government services. Improved HCR systems can streamline data entry from handwritten forms, reduce error rates, and enhance the speed of service delivery, which is especially valuable in high-volume situations. By proposing model combinations and alternative algorithms to address challenges like skew, noise, and handwriting variation, the project contributes toward creating more robust HCR systems that are well-suited for real-world conditions.

1.7 Expected Outcome

Model Performance Analysis: A comprehensive report that analyses performance metrics of each machine learning model applied to the handwritten character recognition task in different data condition. This will help identify the most effective models under different conditions.

Exploring the application of HCR: A software with basic function will be developed. The trained model will be implemented into software, enabling automated and efficient recognition of handwritten text.

1.8 Thesis Outline

1.8.1 Introduction

This chapter provides a brief discussion on the Project background, problem statement, scope, and objectives of the entire project. It also includes a concise overview of the methodology and project schedule used to accomplish the project. Additionally, the significance and expected outcomes of the project are highlighted the advantages of undertaking this project.

1.8.2 Literature Review

Chapter 2 will cover the introduce on the architecture and algorithm of different model that commonly used by researchers to do Handwritten Character Recognition. The related work that done by other research also been investigated. As a result, gaining more knowledge and avoiding repeated mistakes can add greater value to this project.

1.8.3 Methodology

This chapter will focus on the methodology employed throughout the project. It will provide detailed descriptions of the procedures for handwritten character recognition via deep learning techniques. The wireframe of handwritten character recognition website is also included.

1.8.4 Implementation

This section mainly details the technical process of building the model, training, and testing the dataset.

1.8.5 Result and Discussion

This section details how the performance metrics will be computed using the results from the training and testing datasets. These metrics will serve as the basis for comparing and analyzing the results, enabling the identification of appropriate interpretations.

1.8.6 Conclusion and Future Works

Creating a comprehensive summary is essential for helping others understand the overall goals and significance of the project. This section also will outlining potential future work for the entire project.

1.9 Summary

For summary, Handwritten Character Recognition (HCR) is a vital area in Machine Learning (ML), Computer Vision, and pattern recognition. It focused on converting handwritten text into digital text. This project aims to address the lack of comparative analysis of various ML models' performance under different conditions. By testing and analysing various models, the project seeks to identify the most effective models for HCR tasks. The methodology includes dataset preparation, data pre-processing, model training and evaluation using metrics such as accuracy, precision, recall, and F1-score. The findings can benefit industries such as healthcare, finance, and government services by improving data entry efficiency and accuracy, and the project will culminate in the development of software for automated handwritten text recognition.

CHAPTER 2
Literature Review

2.0 Overview

This chapter presents an overview of recent advances in handwritten character recognition. This review focusing on research conducted between 2019 and 2024. This chapter analysed various ML model include Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Transformer-based models which is TrOCR, Xception, and Support Vector Machines (SVM). The algorithm of models also will be explain in this chapter.

2.1 Related Machine Learning model

Machine Learning (ML) is a portion of artificial intelligence (AI) that concentrate on developing systems capable of learning from data and making decisions or predictions without being explicitly programmed for specific tasks. Instead of following predefined rules, ML models use statistical techniques to identify patterns, relationships and structures in data.

2.1.1 Convolutional Neural Networks (CNN)

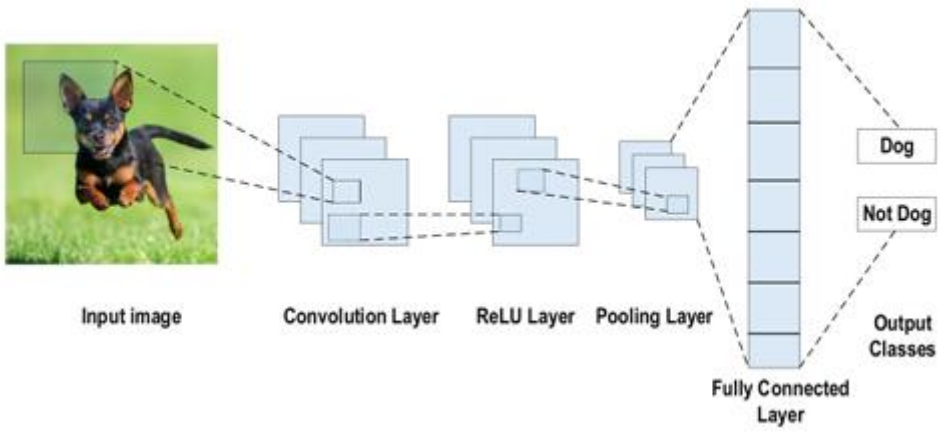


Figure 2.1: The architecture of CNN model for image classification

Convolutional Neural Networks (CNN) are one of the artificial neural network particularly effective for image processing tasks. Figure 2.1 illustrates the process of CNN-based image classification. In this model, it involves an input layer, multiple hidden layers, and an output layer. The hidden layer are convolutional layers, ReLU layers, Pooling layers and a fully

connected layer. Convolutional layers apply filters to the input image to create feature maps, pooling layers use to reduce the spatial size of these maps, and fully-connected layers perform classification.

In a CNN model, the input x of each layer is arranged in three dimensions which are depth, width, and height or m × m × r in mathematical equation, where the width (m) is equal to the height (m). Another name for the depth is the channel number. For instance, the channel for an RGB image is three. Each convolutional layer has several filters that are represented by symbol k and have three dimensions (n × n × q) like input image.

To satisfy the conditions, n must be smaller than m, while q should be either equal to or smaller than r. The filters form the basic of the local connections also share similar parameters (bias vector b^k and weight vector W^k) to produce k feature maps h^k, each with a size of (m – n – 1). These filter are convolved with the input. In the convolutional layer, the input and the weights undergo a dot product computation. The resulting output of the convolution layer is then processed by applying a nonlinearity or activation function.

Convolutional Neural Networks (CNN) also have several advantages over traditional neural networks, particularly in the field of computer vision including HCR. Firstly, CNN advance in employ weight sharing, which can significantly reduce the amount of trainable parameters and reduce the computation power need. This not only enhances the network's generalization capabilities but also helps prevent overfitting. Secondly, CNN can learn feature extraction and classification layers concurrently. This ensuring that the model's output is both well-organized and highly dependent on the extracted features. Lastly, CNNs facilitate easier implementation for large-scale networks compared to other ML models, making them a more practical choice for extensive and complex computer vision tasks.

2.1.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) is a type of ML model that designed for processing sequential data. (Mienye, Swart, & Obaido, 2024) Unlike traditional feedforward ML model, RNN have connections that form directed cycles. This connection can allow the hidden layer to maintain a form of memory about prior inputs in the sequence. This makes them particularly effective for tasks where the order of inputs matters, such as time series prediction, natural language processing, and speech recognition.

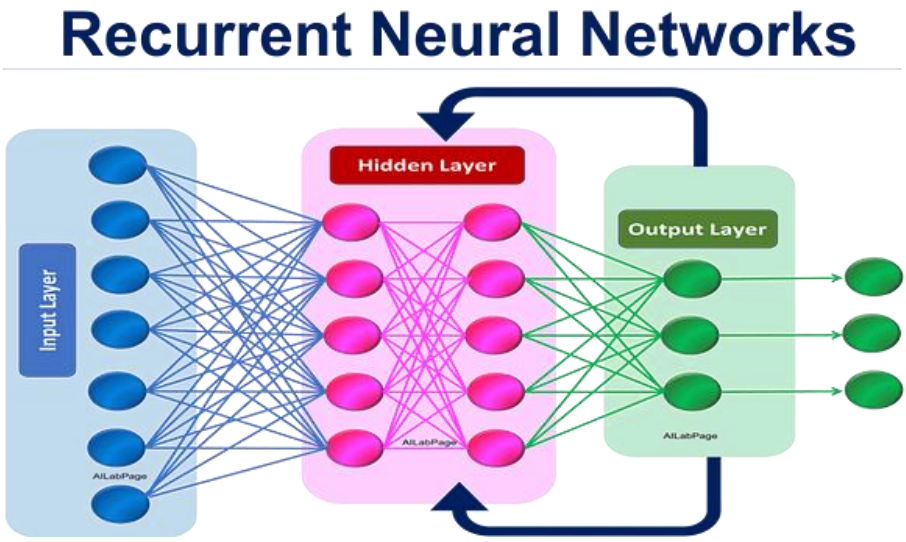


Figure 2.2: The architecture of RNN model

Figure 2.2 shows the architecture of RNN model. RNN models are designed to handle sequential data by keeping hidden state that retains information from prior inputs. it shows that the output that generated from the hidden layer is forwarded to the output layer. Their architecture typically includes an input layer, a hidden layer, and an output layer. Unlike feedforward neural networks like CNN, RNN possess recurrent connections which allow information to cycle within the network. At each time step, t , The RNN model will takes an input vector, X_t and updates its hidden state, h_t. This makes them particularly suitable for tasks like predicting the next word in a sentence, where the context of preceding words is essential (Parthiban, 2020). The equation of hidden states, h_t is shown below

$$h_t = \sigma_h(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$ Eq. (1) Hidden States, h_t

Where W_{xh} is weight matrix among the input and hidden layer, W_{hh} is weight matrix among the hidden layer, b_h is the bias vector, $\sigma_h$is the activation function. The equation of output layer, $y_t$is shown below

$$y_t = \sigma_y(W_{hy}x_t + b_y)$$ Eq. (2) Output Layer, y_t

Where W_{hy} is the weight matrix among hidden and output layers, $b_y$is the bias vector and $\sigma_y$is the activation function for output layer

Beside that, training recurrent neural networks (RNN) is challenging due to vanishing and exploding gradients. These issues arise during backpropagation through time as gradients can diminish or grow exponentially. This may hinder the learning of long-term dependencies or causing instability. Vanishing gradients occur when the Jacobian matrix eigenvalues are less than 1, while exploding gradients occur when they are greater than 1. To address these challenges, RNN variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) use gating mechanisms to manage information flow and gradients. Gradient clipping is also another technique to prevent exploding gradients through capping the flow of information at a maximum threshold during backpropagation. (Mienye, Swart, & Obaido, 2024)

2.1.3 Long Short-Term Memory

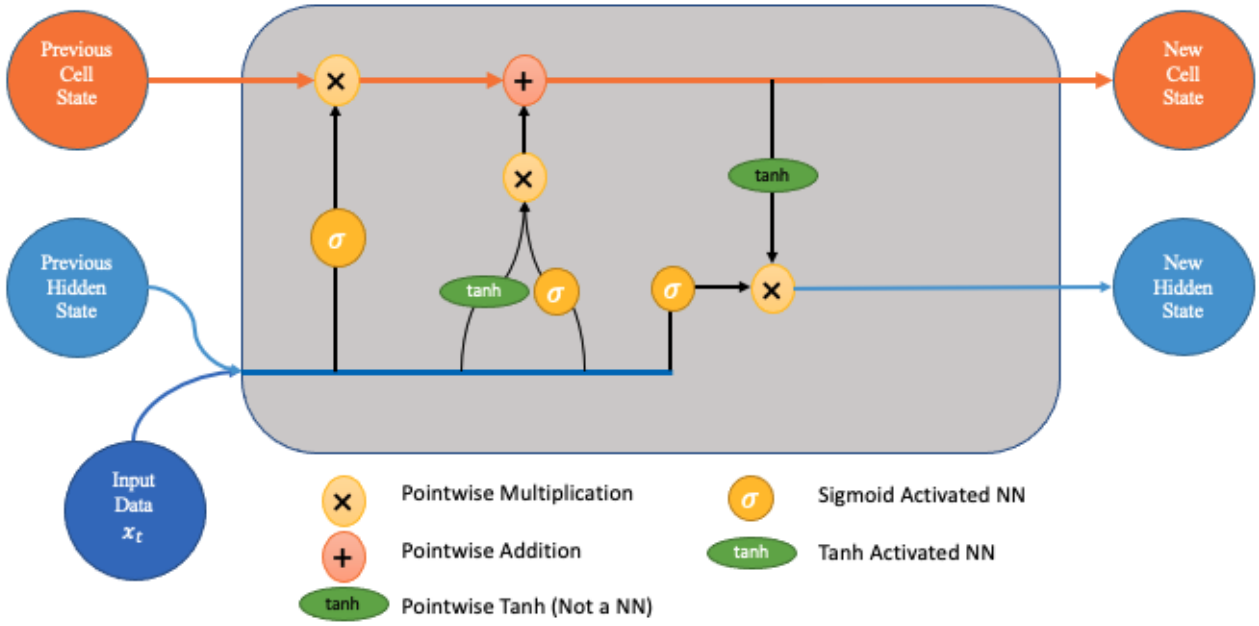


Figure 2.3: The architecture of the LSTM network

Figure 2.3 shows the architecture of the LSTM network. LSTM networks was introduced by Hochreiter and Schmidhuber. It was designed to solve the vanishing and exploding gradient problem seen in RNN model. Their main innovation lies in the gating mechanisms, which control the flow of information through the network. This functionality allows LSTM networks to maintain and refresh their internal state over long durations, making them ideal for tasks requiring long-term dependencies. An LSTM cell comprises three gates: input, forget, and output. These gates regulate the cell state (c_t) and hidden state (h_t) by controlling the input to be considered, the previous state to be discarded, and the cell state to be output. (Mienye, Swart, & Obaido, 2024) The update equations for LSTM are structured to facilitate this process.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \qquad \text{Eq. (3) Input Gate, } i_t$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) \qquad \text{Eq. (4) Forget Gate, } i_t$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \qquad \text{Eq. (5) Output Gate, } o_t$$

$$g_t = tanh\left(W_{xg}x_t + W_{hg}h_{t-1} + b_g\right) \qquad \text{Eq. (6): Cell Input, } g_t$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \qquad \text{Eq. (7) Cell State, } c_t$$

$$h_t = o_t \odot \tanh(c_t) \qquad \text{Eq. (8) Hidden State, } h_t$$

Where i_t is the equation of input gate, f_t is the forget gate, o_t is equation of the output gate, $g_t$is the equation of cell input, $c_t$is the equation of cell state, $h_t$is the equation of hidden state, σ is the sigmoid function, tanh is the hyperbolic tangent function, $\odot$ denotes to element-wise multiplication.

In the LSTM model, three gates control data flow within the cell, each serving a distinct function in controlling the flow of data within cell. The input gate regulates how much new input is added to the cell state. The forget gate decides the amount of previous cell state to retain. The output gate determines which part of the cell state contributes to the hidden state. Additionally, the cell input, serving as a candidate value, is modulated by the input gate before being integrated into the cell state. These gating mechanisms allow LSTM networks to selectively keep or discard information, thus managing long-term dependencies more effectively than traditional RNN (Yu, 2019).

2.1.4 Transformer Models

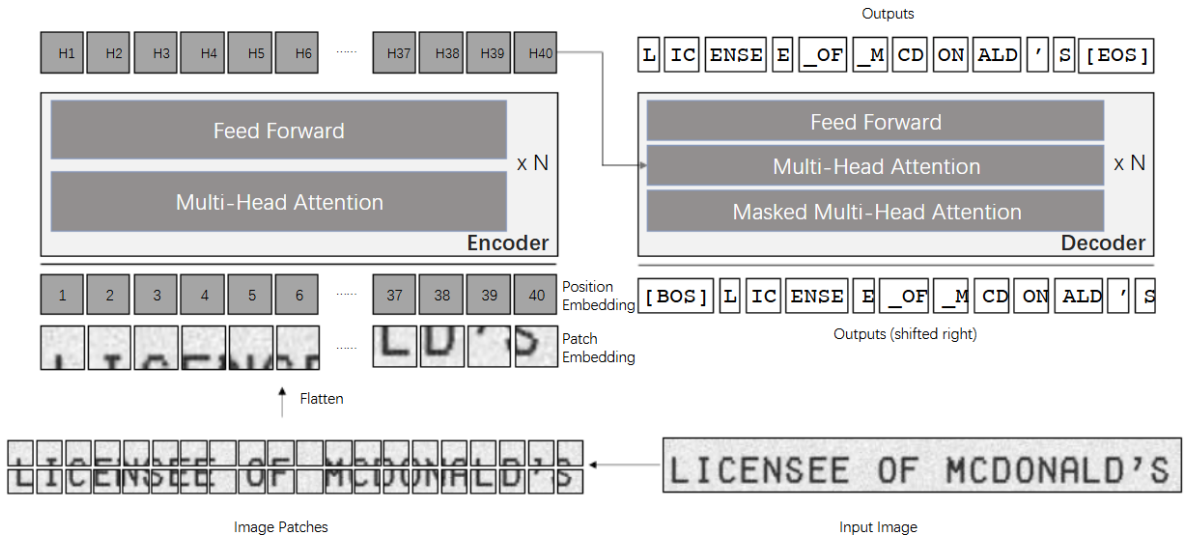


Figure 2.4: The architecture of TrOCR model

TrOCR (Transformer-based Optical Character Recognition) is an innovative end-to-end OCR model based on the Transformer model's architecture (Li, 2021). Based on figure 2.3, this model

features a pre-trained Vision Transformer (ViT) as the encoder and a pre-trained text Transformer, such as RoBERTa, as the decoder. The encoder processes resized input images (384x384 pixels) by dividing them into 16x16 patches. These patches will been embedded into a sequence of vectors and using a multi-head self-attention mechanism to extract rich visual features. The decoder employs a standard Transformer-based structure to generate text output, attending to both the encoder's features and previously generated tokens. This design eliminates the need for convolutional layers, commonly used in OCR systems, simplifying the architecture while improving performance. TrOCR is pre-trained on large scale synthetic datasets and fine-tuned with task-specific data for achieving state-of-the-art results on both printed and handwritten text recognition tasks

2.1.5 Xception Model

The Xception model stands for "Extreme Inception." It's an advanced version of the Inception model, with the primary difference being the use of depthwise separable convolutions instead of standard convolutions. This change significantly reduces the number of parameters and increases computational efficiency without sacrificing performance.

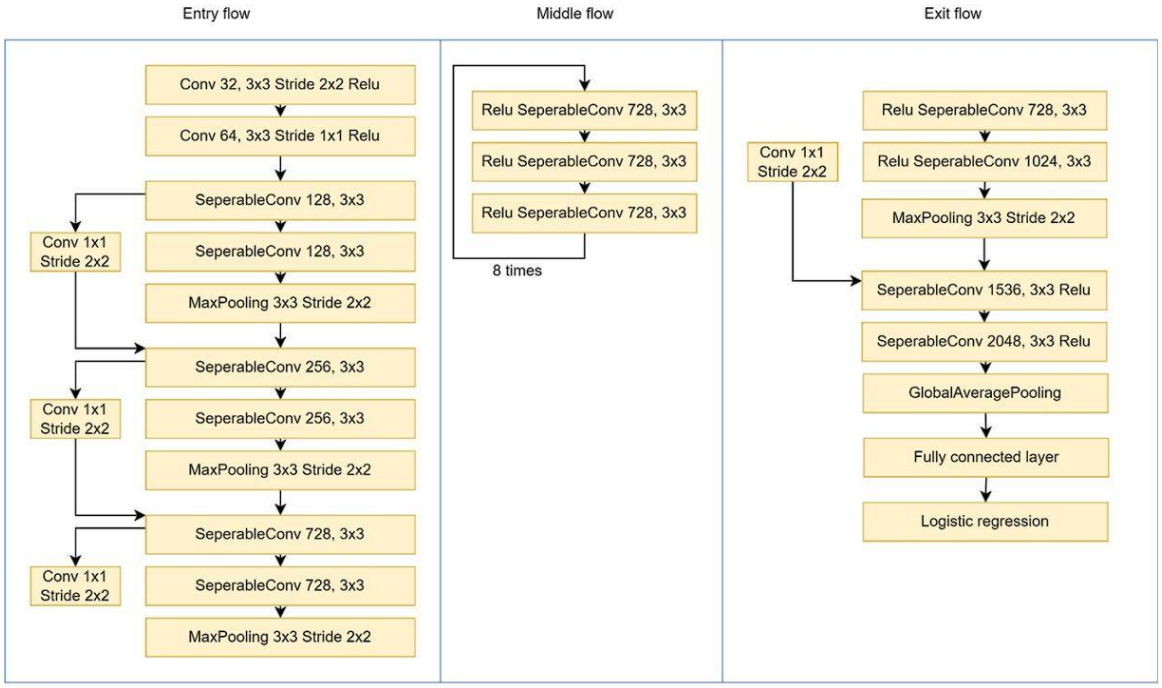


Figure 2.5: The architecture of Xception model

The Xception architecture consists of three main parts: the entry flow, the middle flow, and the exit flow, all connected by skip connections across its 36 layers (Gülmez, 2022). In the entry flow, an input image (299×299 pixels with RGB channels) goes through initial feature extraction using two 3×3 convolution layers with 32 and 64 filters, activated by ReLU. Next, the separable convolution layer combined with a 1×1 convolution layer further extracts the features, while 3×3 max pooling with a stride of 2 reduces the feature map's spatial dimensions. The middle flow contains eight repeated blocks, each featuring a depthwise separable convolution layer with 728 filters and ReLU activation. This structure allows for progressive extraction of high-level features. In the exit flow, separable convolution layers with filters increasing from 728 to 2048 capture more detailed features. Finally, global average pooling condenses the feature maps into a single vector, followed by a fully connected layer that uses logistic regression for classification.

2.1.6 Support Vector Machine (SVM) Model

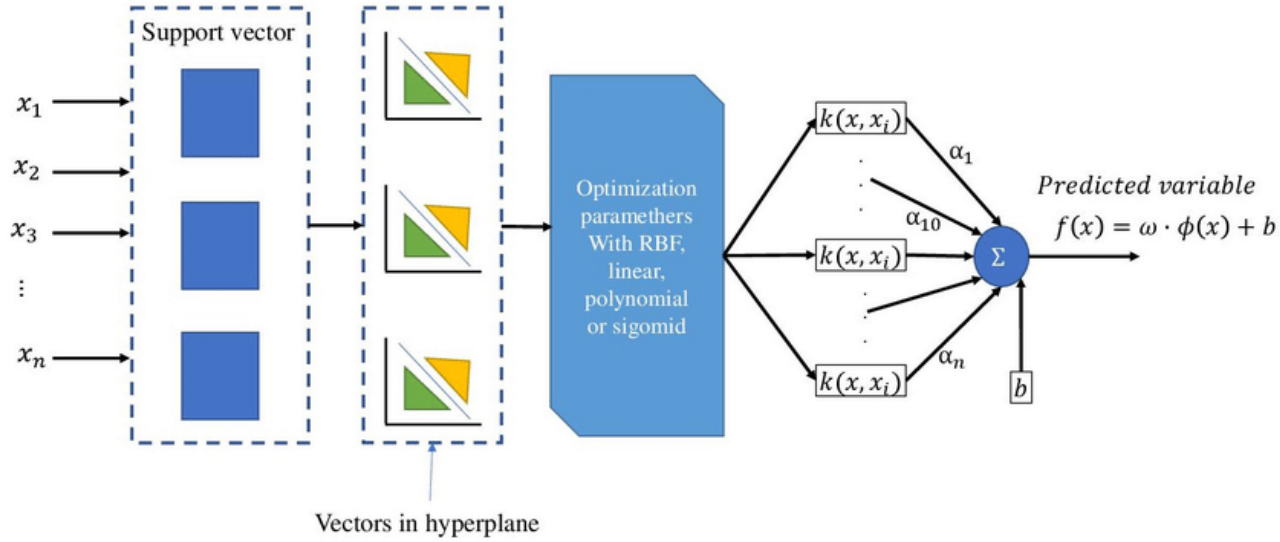


Figure 2.6: The architecture of Support Vector Machine model

A Support Vector Machine (SVM) is a powerful supervised learning algorithm primarily used for classification tasks, though it can also be applied to regression (José Manuel Álvarez-Alvarado, 2021). It works by identifying the optimal hyperplane that separates data into distinct classes. The main objective is to maximize the margin, the distance between the hyperplane and the closest data points from each class, known as support vectors. These support vectors are essential as they determine the hyperplane's position and orientation. When data is not linearly separable, SVM utilizes the kernel trick to map data into a higher-dimensional space, enabling effective class separation with a hyperplane. Common kernel functions include the polynomial kernel and the radial basis function (RBF) kernel. The SVM algorithm is depicted below

2.2 Related Study

Yang, et al. (2016) stated that DropSample, a model training method aimed to improve the efficiency and performance of CNN model for large-scale handwritten Chinese character recognition (HCCR). The method dynamically adjusts training sample quotas based on classification confidence to ensure that challenging samples are revisited more frequently while easy and noisy samples are gradually excluded. The paper also incorporates domain-specific knowledge to enhance CNN performance on Chinese Handwritten Character Recognition such as path signature features and deformation transformations,.

In a study conducted by Saqib, et al. (2022), CNN demonstrated high effectiveness in handwritten character recognition (HCR) due to their ability to automatically extract and classify image features with minimal preprocessing. However, challenges such as generalization across diverse datasets and computational complexity remain significant. The proposed lightweight CNN architecture addresses these challenges through a design comprising four convolutional layers, three max-pooling layers, and two dense layers. This structure balances computational efficiency with high accuracy, achieving 99.642% and 99.563% accuracy on the MNIST digit recognition dataset and the Kaggle English alphabets dataset, respectively.

This study also further highlights the importance of hyperparameter tuning and optimizer selection. The ADAM optimizer yielded superior results for alphabet recognition, while RMSprop performed better for digit recognition.

Parthiban (2020) discuss advanced variants of RNN which is Long Short-Term Memory (LSTM) model. This mode enhances its capability by handling long-term dependencies efficiently. LSTM can retain relevant information over extended sequences. This feature can mitigate issues like vanishing gradients that can affect standard RNNs. This study also concluded that the LSTM model can achieve the accuracy of 90%

Singh, et al. (2021) have research the performance of handwritten Gurmukhi characters using RNN model. The author mention RNN's ability in process sequence data. In HCR, points in a stroke or characters in a word are drawn in a time sequence and RNN excel at handling such temporal data for prediction. In this context, RNN can leverage the temporal relationship between the sequence of points with each point in the sequence acting as a timestep. The output of a hidden state in an RNN depends on the current input and the output of previous states,

creating a dependency on the history of inputs. This characteristic makes RNN well-suited for handwriting recognition systems.

Fang (2020) has Propose research on real-time handwritten character recognition on Chinese Handwritten character known as “HandiText”. The result indicate that the LSTM model has the best classification accuracy which reaching 99% while keeping relatively low false positive rate and false negative rate. The training time of LSTM model also short. The author also tested other datasets such as MNIST, IAM and CASIA, it found that the average accuracy of HCR reach around 98%, with strong generalization ability.

Khan & Nazir (2022) have done research on Pashto (official language of Afghanistan) Character Recognition using LSTM. The methodology done is followed by a sequence of data description, Feature Extraction, Classification Techniques and performance evaluations. The data description is done by collecting a total of 4488 handwritten Pashto sample that written by different people. For the feature extraction, they use two different way which are Zoning Technique and Invariant Moment Technique. Zoning Technique is done by counting the number of pixel within 8x8 grids while Invariant Moment Technique is done by calculate the following equation.

$$n_{p,q} = \frac{\mu_{p,q}}{(\mu_{0,0})^{\gamma}} \text{ where } \gamma = \frac{p+q+2}{2} \qquad \text{Eq. (10)}$$

3rd Order Normalised Spatial Moments

In the equation, “p” and “q” are the normalised moment in x and y axes where $n_{p,q}$ is the spatial moments of order (p,q). The Classification Techniques is done using LSTM model as classifier. An overall accuracy rate of 89.03% is calculated for the LSTM model. The 18.97% error rate is attributed to the similar shapes of many characters in the Pashto language, leading to misclassification. This suggests that the LSTM model may achieve higher accuracy when recognizing characters from other languages.

In the study conducted by (Li, 2021), TrOCR model get high performance for using Transformer-based architectures for OCR tasks. On the IAM Handwriting Database, TrOCR achieved a Character Error Rate (CER) of 2.97%, outperforming many existing methods, including those that utilize external language models. Similarly, on the SROIE dataset, TrOCR

models achieved high F1 scores of 96.34% (base) and 96.58% (large). The impact of pre-training is significant as models initialized with pre-trained parameters showed substantial improvements over those trained from scratch. Furthermore, the use of a Transformer-only architecture proved to be competitive, surpassing hybrid CNN + RNN models without the need for complex pre- or post-processing steps.

Fujitake (2024) stated that Decoder-only Transformer for Optical Character Recognition (DTrOCR). Unlike traditional optical character recognition (OCR) systems that rely on an encoder-decoder architecture, DTrOCR eliminates the need for a separate vision encoder while it use a pre-trained generative language model (LM) as the decoder. DTrOCR is characterized by its simplified architecture that eliminates the need for a separate vision encoder, relying solely on a decoder-only Transformer. It transforms input images into patch sequences, which are directly processed by the decoder. By leveraging a pre-trained generative language model (LM), DTrOCR incorporates linguistic knowledge to improve text recognition accuracy across printed, handwritten, and scene text.

Nanehkaran (2021) integrated the Xception model within a convolution bagging weighted majority ensemble (CBWME) learning framework to enhance recognition accuracy for challenging datasets like HODA, IFHCDB, and CENPARMI. When employed as a standalone classifier, the Xception model achieved a notable recognition accuracy of 95.9% on the HODA dataset, outperforming other CNN architectures such as ResNet18 (93.75%) and VGG16 (90.26%). This highlights Xception's superior feature extraction capabilities and adaptability to the cursive nature and complex digit shapes of Farsi handwriting. Furthermore, its inclusion in the CBWME ensemble model contributed to the ensemble's exceptional performance, achieving an accuracy of 97.65% on the same dataset, the highest among tested configurations.

Ahlawat & Choudhary (2020) proposed a hybrid model of CNN-SVM for handwritten digit recognition. This model involves the automatic feature generation using CNN and output prediction using SVM. CNN model can extract the most discriminating information from raw digit images while SVM model can minimize the generalization error on unseen data as classifier. Therefore, CNN-SVM model combine the advantage of CNN and SVM classifiers. Experimental results indicate that the hybrid CNN-SVM classifier achieves a recognition accuracy of 99.28% for handwritten digits, surpassing the 98.35% accuracy achieved by the SVM classifier alone.

2.3 Comparison of Machine Learning Model

Table 2.1 Table of Comparison between Machine Learning Model

Model With References	Description	Evaluation Metrics	Result
CNN Model (Saqib et al.,2022) (Alzubaidi, et al., 2021) (Yang, 2016)	• MNIST digit benchmark dataset and Kaggle alphabet data are used to train model. • 'ADAM' and 'RMSprop' optimizer are used to test their help in model efficiency • The feature like weight-sharing, feature extraction concurrently can also increase model performance	• Accuracy • Validation losses • F1 Score • Recall • Confusion Matrices	• The overall accuracy on the MNIST digit recognition dataset and the Kaggle English alphabets dataset are 99.642 % and 99.563% respectively • The hyperparameter tuning and optimizer can increase the performance of CNN model • "DownSampling" that proposed by (Yang, 2016) is also will been utilize
RNN model (Parthiban et al., 2020) (Singh, Sharma, Singh, & Kumar, 2021)	• The backpropagation feature make it can easily to predict the next word in a sentence • Long Short-Term Memory (LSTM) is used to mitigate the vanishing gradient problem	• Accuracy	• The RNN model manage to achieve 90% accuracy •
LSTM model (Fang, 2020) (Khan & Nazir, 2022)	• An advance variant of RNN model when it solve the Vanishing and exploding gradient problem by controlling the flow of data using different gate	• Accuracy	• The accuracy of LSTM model reaches 98% based on ((Fang, 2020) • Two Feature Extraction which are zoning and Invariant

			Moment Technique are introduced
Transformer Model (Li et al., 2021)	• Three Encoder ($DeiT_{BASE}$, $BEiT_{BASE}$ and ResNet50) and Two Decoder ($RoBERTa_{LARGE}$ and $RoBERTa_{BASE}$) are tested in this study • TrOCR eliminates the need for external language models, which are traditionally used in OCR pipelines • Since TrOCR need high computational resource to operate, the research of TrOCR's performance in low-resource or domain-specific OCR tasks is limited.	• Word-level Precision • Recall • F1 Score	$BEiT_{BASE}$ encoders show the best performance among three encoders. The $RoBERTa_{LARGE}$ perform better than base decoders
Xception model (Nanehkaran et al., 2021)	• This study proposed a pragmatic CBWME network structure model for Farsi handwritten digit recognition • The CBWME model combines multiple CNN architectures with a weighted bagging ensemble strategy • Some digits such as "3" consistently performed poorly across datasets, which show weaknesses in the model's handling of specific character shapes The CBWME model has higher training and	• Accuracy • Sensitivity • Specificity	• CBWME outperforms other models with average recognition accuracy of 97.65%

	inference times compared to individual CNN models		
CNN-SVM model SVM model (Ahlawat & Choudhary, 2020)	• This study proposed hybrid CNN-SVM model where CNN works as a feature extractor and SVM as a binary classifier. • This study used MNIST handwritten digit benchmark dataset for training and testing	• Accuracy	Experimental results indicate that the hybrid CNN-SVM classifier achieves a recognition accuracy of 99.28% for handwritten digits, surpassing the 98.35% accuracy achieved by the SVM classifier alone

2.4 Tools and Technologies Used

2.4.1 Handwritten Character Recognition

Handwriting is a common and systematic method for recording information, with everyone's handwriting being distinctive and unique. A Handwritten Character Recognition (HCR) system refers to the ability of software or devices to recognize and analyse handwriting in any language. (Saqib, Haque, Yanambaka, & Abdelgawad, 2022). Recognition can be performed on both online (digitally captured) and offline (scanned) handwriting. In recent years, HCR has seen widespread applications, including postal address reading, language translation, bank check processing, digital libraries, keyword spotting, and traffic sign detection.

An HCR system typically involves several stages: image acquisition, preprocessing, segmentation, feature extraction, and classification. The process starts with capturing an image of the handwritten text, which is then input into preprocessing. In preprocessing, distortions in the scanned image are corrected, and the image is converted to binary. During segmentation, the text is divided into individual characters, and features are extracted from each character. Finally, in the classification stage, the extracted features are used to identify the characters. Different classification techniques, such as Convolutional Neural Networks (CNNs), Support Vector Machines (SVMs), Recurrent Neural Networks (RNNs), Deep Belief Networks, Deep Boltzmann Machines, and K-Nearest Neighbour (KNN), are used depending on the approach and accuracy requirements. (AlKendi W. G., 2024)

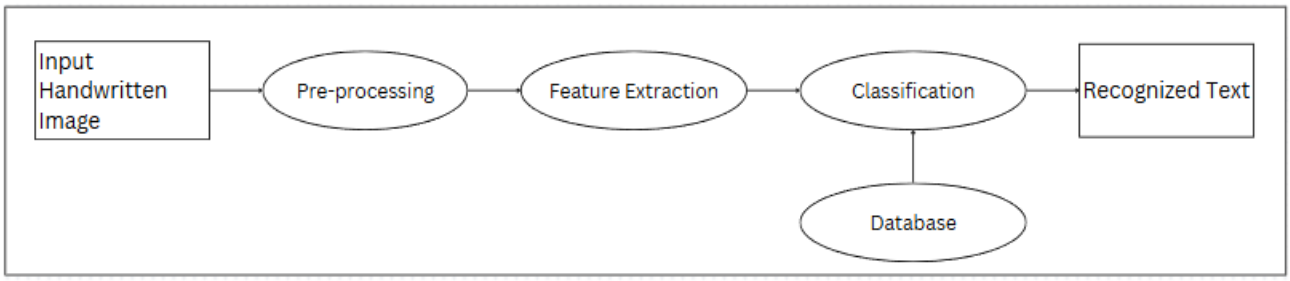


Figure 2.7: Stages of Handwritten Character Recognition System

2.4.2 Python

Python (https://www.python.org/) has knows as a leading programming language in the field of machine learning. Its simplicity, versatility and extensive ecosystem of libraries have strength its role in the field of machine learning. Its clean, easy-to-read syntax allows both beginners and experienced developers to focus more on solving machine learning problems than on dealing with complex code. The language's rich ecosystem includes powerful libraries such as NumPy for numerical computations, Pandas for data manipulation and Scikit-Learn for machine learning algorithms. Furthermore, Python integrates seamlessly with other languages and technologies making it ideal for building and deploying machine learning models in real-world applications.

2.4.3 Tensorflow

TensorFlow (https://www.tensorflow.org/) is an open-source deep learning framework. It's designed for building and training deep learning models with ease. TensorFlow allows for both CPU and GPU computations which make it suitable for large-scale machine learning tasks.

2.4.4 OpenCV

OpenCV (https://opencv.org/) stands for Open-Source Computer Vision Library. It is a powerful library for computer vision and image processing. It provides tools for real-time image and video processing. It can apply for tasks such as object detection, face recognition, and motion tracking.

2.4.5 Visual Studio Code

Visual Studio Code (https://code.visualstudio.com/) is a free, open-source code editor developed by Microsoft. It is highly versatile and widely used by developers for various programming tasks. One of its standout features is support for multiple programming languages, including JavaScript, Python, Java, and C++. The support for Python is particularly noteworthy, as Python is widely used in machine learning, data analysis, and general-purpose programming. VS Code enables seamless integration with Python through the ability to install extensions like IntelliSense for intelligent code suggestions, linting for error and style checking, debugging tools for troubleshooting, and code navigation for efficient editing and development.

2.5 Summary

The literature review explores various machine learning models used for handwritten character recognition. This review highlights the advancements in Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and emerging Transformer-based models like TrOCR. Among these, ensemble methods like the CBWME network demonstrate superior accuracy (97.65%) for Farsi digit recognition, surpassing standalone models such as Xception and ResNet18. However, significant gaps persist, including a lack of generalization across diverse and noisy datasets, limited research on multilingual recognition, and challenges with computational efficiency in high-performing models.

CHAPTER 3

Methodology

3.0 Overview

This chapter will discuss the methodology that proposed and will be implemented in chapter 4. The process and tools needed for the entire project will be described below.

3.1 Knowledge Discovery in Database

Knowledge Discovery in Database (KDD) is a multi-step process used to extract meaningful insights from large volumes of data (Maimon, 2005). It involves stages such as data integration, cleaning, selection, transformation, mining, pattern evaluation, and knowledge presentation. In the context of handwritten character recognition, KDD plays a pivotal role in processing the diverse and complex data involved such as scanned images and handwritten samples. The importance of KDD lies in its ability to help uncover hidden patterns in data which can lead to improved recognition accuracy and model performance. By applying KDD, the project can enhance the preprocessing, feature extraction, and evaluation stages, which are crucial in recognizing characters accurately. Furthermore, KDD's role in identifying trends and optimizing model parameters will significantly influence the future development of the project, allowing for more precise comparisons between different machine learning models and a better understanding of their strengths and limitations in recognizing handwritten characters.

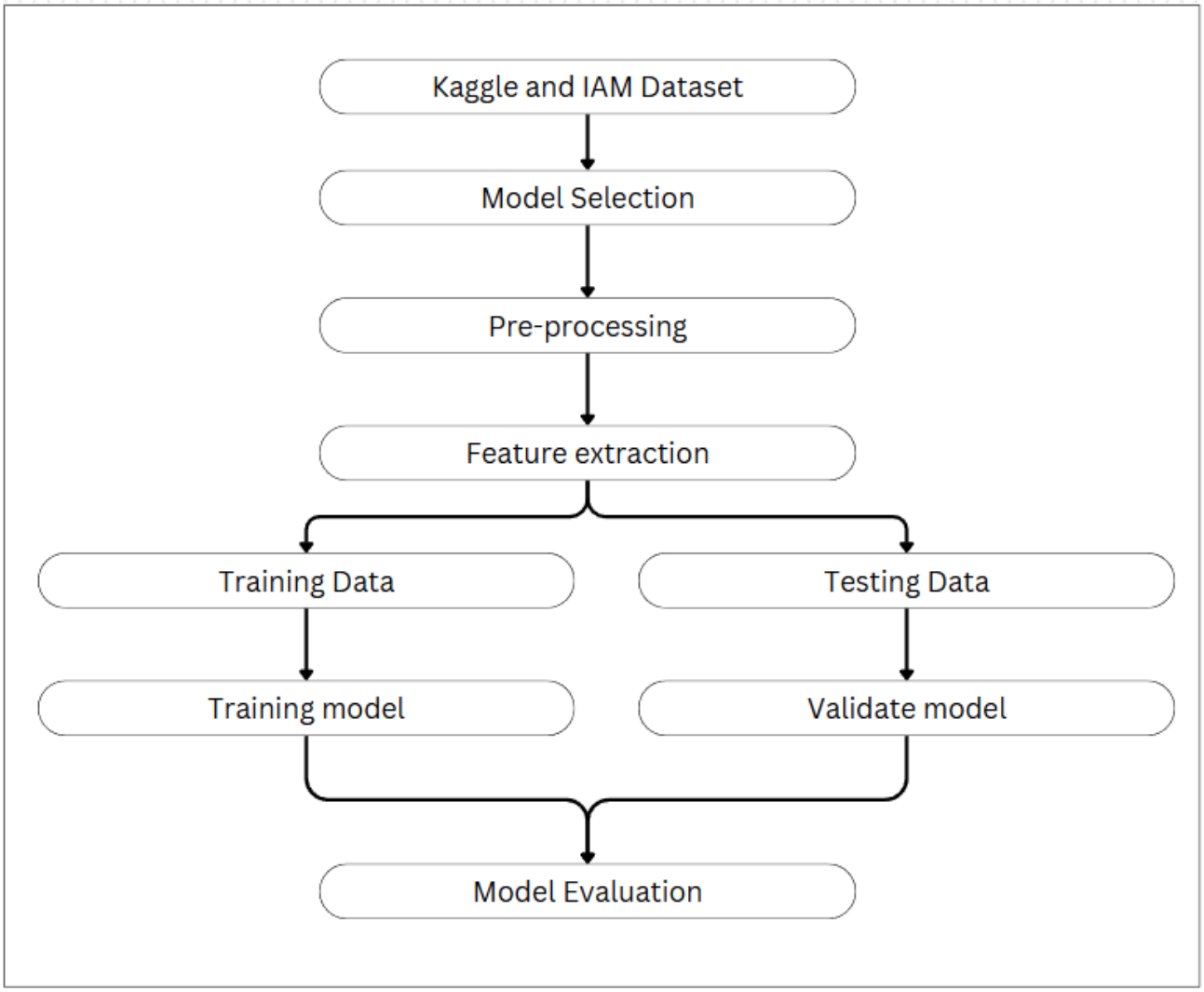


Figure 3.1: Flow diagram of the proposed methodology

3.1.1 Data Collection

The dataset used in the study are the "Handwritten Recognition" dataset from (Kaggle, n.d.) and "IAM Handwriting" dataset from (Marti & H., 2002). The Handwritten Recognition dataset consists of 413823 images that collected through clarity projects, The dataset comprises 206,799 first names and 207,024 surnames. It is divided by author into a training set (331,059), a testing set (41,382), and a validation set (41,382).

On the other hand, IAM Handwriting dataset consists of 13353 images of handwritten lines of text created by 657 writers. The texts transcribed by these writers are derived from the Lancaster-Oslo/Bergen Corpus of British English. The collection includes contributions from 657 writers, resulting in a total of 1,539 handwritten pages and 115,320 words. It is categorized as part of the modern collection. This dataset was chosen for its high-quality annotations,

variety in handwriting styles and its wide use in research which is more easier to benchmark the model performance against existing studies.

3.1.2 Model Selection

Model selection is critical in ensuring the success of handwritten character recognition (HCR) systems. For this project, Convolutional Neural Networks (CNN), Long Short Term Memory (LSTM), and the TrOCR model were selected due to their complementary strengths in handling diverse aspects of HCR tasks.

CNN were chosen for their proven ability to automatically extract meaningful hierarchical features from image data, making them particularly effective for identifying patterns such as edges, curves, and textures in handwritten characters. Their robustness, computational efficiency through weight sharing, and wide applicability to various image-based recognition tasks make CNNs a fundamental choice for this study.

LSTM networks are chosen as particularly effective in processing sequential data. The make them well-suited for handwriting recognition especially in cursive scripts or connected characters. By addressing issues such as vanishing and exploding gradients that happened on RNN, LSTMs can retain context over longer sequences and capture temporal dependencies in handwriting. This ability makes LSTMs especially valuable for recognizing handwriting styles that influenced by preceding or succeeding characters.

The TrOCR model, a Transformer-based OCR model, was selected for its innovative architecture that eliminates convolutional layers, relying instead on self-attention mechanisms to capture long-range dependencies and contextual relationships. TrOCR's ability to process input images as sequences and its high accuracy across printed and handwritten text benchmarks make it ideal for complex HCR tasks requiring precision and adaptability. Together, these models offer a diverse and robust framework for exploring and addressing the challenges of handwritten character recognition.

3.1.3 Image Preprocessing

In this study, the image preprocessing play a critical role in remove the feature that will disturb model recognition. The key step of preprocessing are Noise removal, segmentation, Binarization and Normalization

During scanning, input images may contain various types of noise, such as black pixels where white pixels should be or vice versa. This noise can interfere with the accuracy of character recognition. In the case of the input image shown in Figure 2, background noise was present. To address this, a median filter with a 3x3 filter size was applied to remove the noise. As demonstrated in Figure 4, this noise reduction algorithm effectively eliminated the unwanted background noise, leaving a cleaner image for further processing. (J, v, & S, 2011)

Segmentation is the process of separating words or individual characters within the image, which is essential for Handwritten Text Recognition (HTR). Since the framework can only recognize one word at a time, accurate segmentation is crucial. The basic connectivity principle was used, where pixels belonging to a word are considered connected. This principle is also used to separate the image from its background. To further refine the image, the color image is first converted to grayscale, and a fixed-size window, equal to the size of a single word, is used for clipping. This process ensures that each word is isolated and can be processed individually (Jaiswal, 2023)

Binarization is a vital step in image processing, where the image is converted into two distinct parts: the background (white) and the foreground (black). This step simplifies the recognition task by reducing the image to a binary format. A global grayscale intensity threshold is applied to classify the pixels into these two categories, producing a binary image that helps highlight the important features of the handwritten text (AIITK, 2023)

Since the classifier needs to handle different font sizes, normalization is required to ensure that all incoming characters are resized to a standard size. This step involves adjusting the image size to match the dimensions accepted by the classifier, allowing the neural network to process the image effectively. The input layer of the neural network receives these standardized image pixels, ensuring that the model can work with uniform input data (Mandal, 2019).

3.1.4 Feature Extraction

Feature extraction is a crucial step in handwritten character recognition as it involves identifying and isolating the key characteristics of the image that are necessary for accurate recognition. Feature extraction in handwritten character recognition involves transforming each character image into a vector representation by isolating and encoding significant characteristics. Broadly, feature extraction methods consider statistical features, structural features, and global transformations and moments (Vijay & Y, 2012).

Statistical features capture quantitative properties of the character image and include techniques such as zoning, projection histograms, profiles, and crossings and distances. Zoning divides the character image into a grid, and features are extracted from each zone, such as local pixel densities. Projection histograms count the number of foreground pixels along rows and columns, while profiles measure the distance between the character's bounding box and its edges. Crossings and distances measure transitions from background to foreground pixels along vertical and horizontal axes.

Structural features represent topological and geometrical properties of a character, offering robustness against distortions and style variations. Examples include aspect ratio, cross points and loops, branch points and strokes, projection histograms and radial histograms, and radial in-out profiles.

Global transformations and moments consider the entire image, capturing global patterns through mathematical transformations and moments. These techniques encode the character's overall distribution of pixels and spatial properties. In deep learning approaches, feature extraction is often automated through models like Convolutional Neural Networks (CNN), where layers learn hierarchical features directly from the input images. The extracted features are then assembled into a feature vector, which is used as input for machine learning or recognition algorithms.

3.1.5 Modelling and validation

The modelling phase in handwritten character recognition plays a pivotal role in crafting, training, and optimizing machine learning frameworks. In this stage, models are developed to

extract and learn intricate features, such as character shapes, textures, and pattern. The custom-built architectures and pre-trained models are utilized.

Following the feature extraction stage, the data is split into training and testing sets. The training data is used to train the model, allowing it to learn patterns and generalize effectively. Simultaneously, the testing data is reserved for validating the model's performance, ensuring it can handle unseen data. This dual-path approach with training and validation will facilitates high performance model.

3.1.6 Evaluation Metrics

Evaluation Metrics play a critical role in optimizing model and grading the model's performance. The evaluation metrics are usually used within the two main stage which are training and testing stage. This evaluation metric is utilized to measure the efficiency of a classifier.

Among these evaluation metrics, Accuracy, precession, recall and F1-score are most used to measure the model's performance. These metrics offer important information about how well the model recognizes and differentiates handwritten characters. They are computed using the confusion matrix, which compiles the actual class labels and the model's predictions.

a) Confusion Matrix

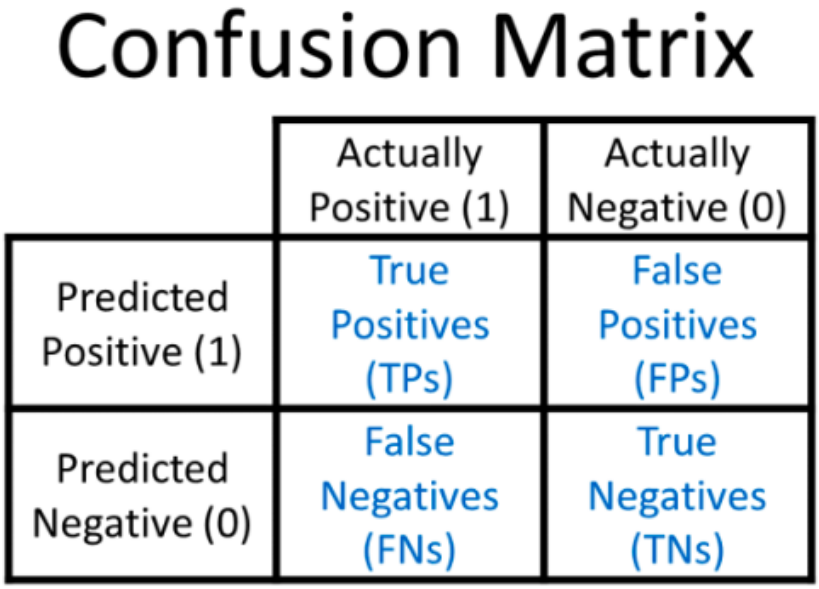


Figure 3.2: Confusion Matrix

"A confusion matrix, which is an N x N matrix where N represents the total number of target classes, is utilized to assess the performance of a classification model (Draelos, 2019). This matrix makes it easier for us to determine whether the model's predicted answer matches the actual answer. Additionally, it can determine the model's accuracy percentage using a few equations.

From this confusion matrix, true positives (TP) and true negatives (TN). denote the number of positive and negative instances that are correctly classified. Meanwhile, False positives (FP) and False negatives (FN) denote the number of misclassified negative and positive instances, respectively. (Hossin & Sulaiman, 2015)

In the context of character detection algorithms, correctly detected characters are classified as true positives (TP). Conversely, regions in the images that incorrectly identify characters are treated as true negatives (TN). False positives (FP) occur when the algorithm detects text in areas that do not contain any characters. False negatives (FN) arise when regions containing actual characters are not detected by the algorithm. (Md Rafiqul, Rashedul, & Kamrul Hasan,

2020) These typical scenarios that may face by HCR and confusion matrix will be used for model evaluation purposes.

b) Accuracy

The accuracy metric calculates the proportion of accurate predictions to all instances assessed.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$ Eq. (10) Accuracy

c) Sensitivity

The percentage of positive patterns that are accurately categorized is measured by this metric.

$$Sensitivity = \frac{TP}{TP+FN}$$ Eq. (11) Sensitivity

d) Specificity

The percentage of incorrectly categorized negative patterns is measured by this metric.

$$Specificity = \frac{TN}{FP+FN}$$ Eq. (12) Specificity

e) Precision

Out of all the expected patterns in a positive class, precision is the number of accurately predicted positive patterns.

$$Precision = \frac{TN}{FP+FN}$$ Eq. (13) Precision

f) Recall

The percentage of positive patterns that are accurately categorized is called recall.

$$Recall = \frac{TP}{TP+TN}$$ Eq. (14) Recall

g) $F1_{score}$

The F1 Score is the harmonic mean of recal and precision. It is employed when recollection and precision fail to maximize or offer significance. Therefore, a more complete picture of the result is obtained by combining the F1 Score with other performance measurements.

$$F1_{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \qquad \text{Eq. (15) } F1_{score}$$

3.2 Wireframe

Wireframe is a visual blueprint that provides a clear structure and layout for the project. It outline the essential components of the Handwritten Character Recognition system and their interconnections between each wireframe.

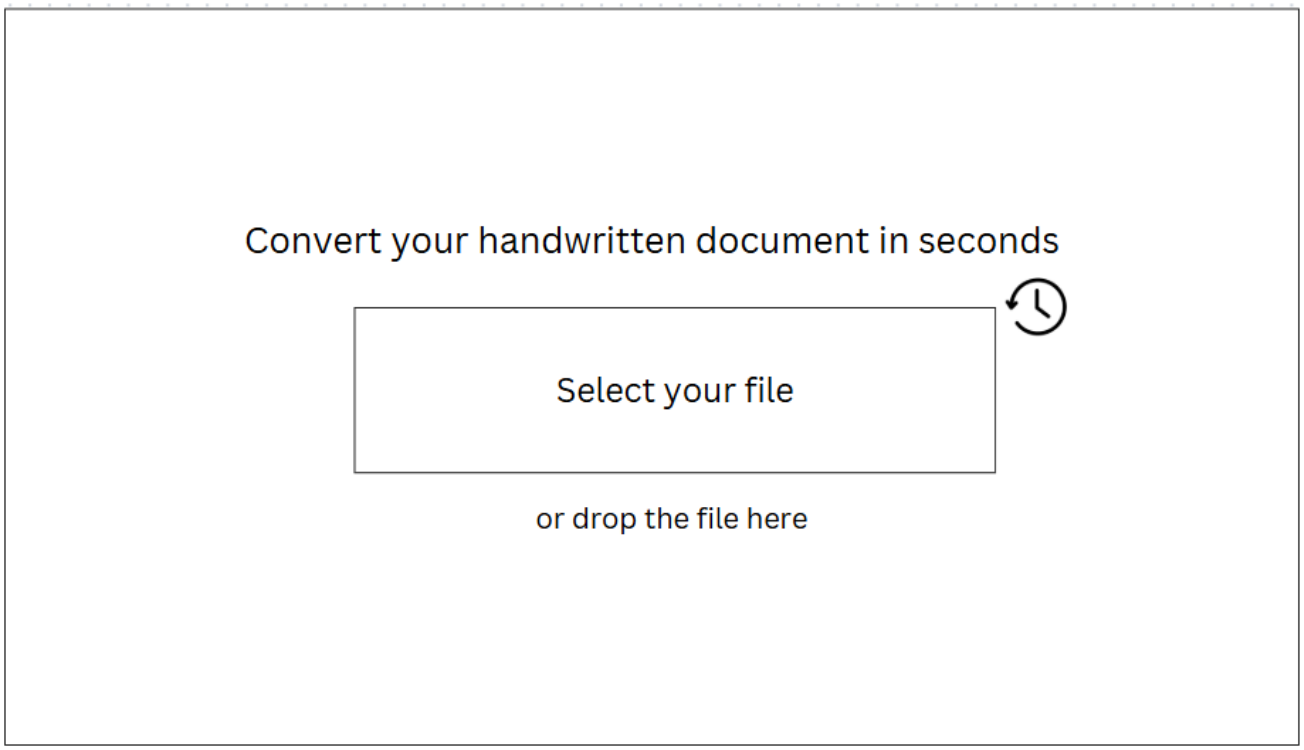


Figure 3.3: Main Screen Interface for File Upload and History Access

Figure 3.3 illustrates the main screen of the Handwritten Character Recognition system. Before using this software, user need to shoot their handwritten documents clearly to increase the accuracy of recognition. In this page, user can select the file from the folder by clicking "Select your file "button or drag their file into the screen to upload the handwritten document into system. User can also access their previous uploaded document through clicking the history icon.

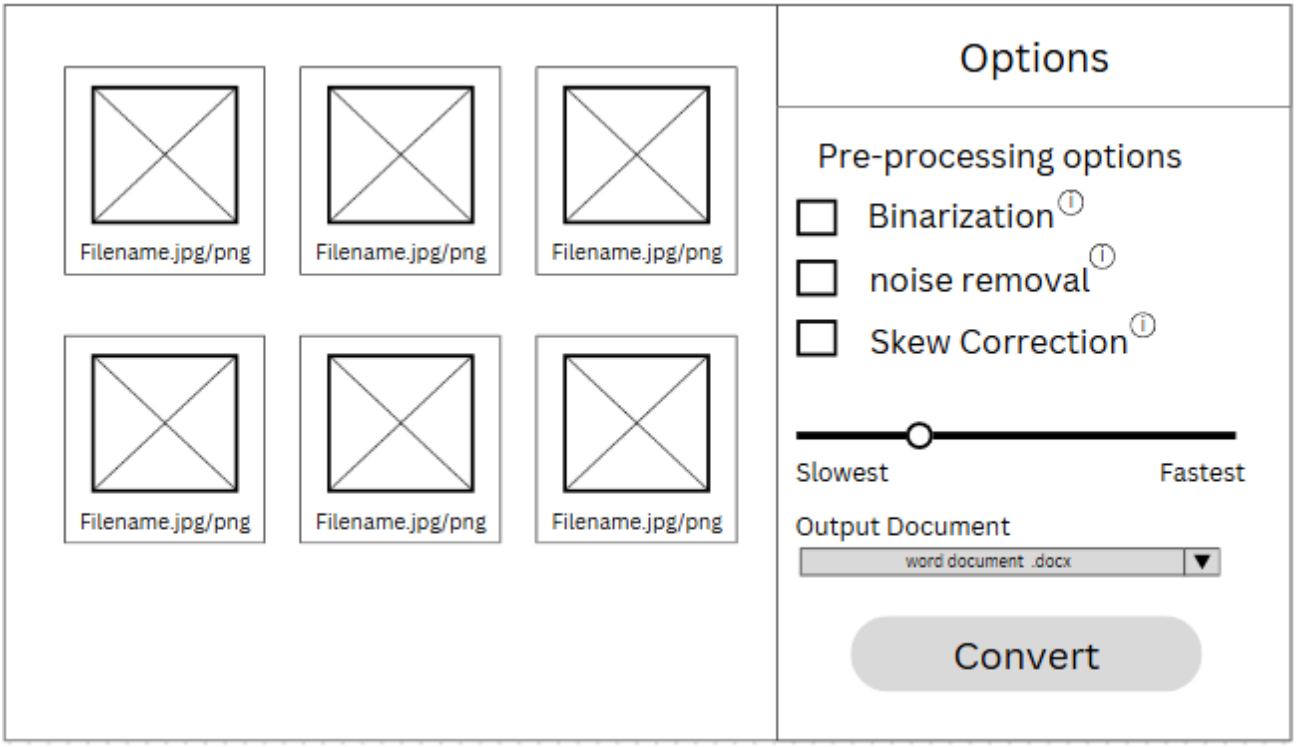


Figure 3.4: Pre-Processing Screen with Image Management and Customizable Options

Figure 3.4 illustrates the pre-processing screen of the system. It is designed to give users control and flexibility over their uploaded images. On this screen, users can preview the images they have uploaded and manage them easily by removing or rearranging the order of documents. On the right-hand side, users can customize their pre-processing options. They can select specific pre-processing steps, such as noise removal or skew correction. A tooltip or information icon is provided for each option which allow users to understand its functionality before enabling it. However, users are advised to select options judiciously as choosing more options can slow down the overall processing speed. The screen also features a slider to adjust the processing speed. The slider determines the size of the dataset that the model will use, balancing between speed and accuracy depending on the user's preference. Additionally, users can specify the desired output file format for the recognized text, such as PDF, Word, or plain text. Once all settings are configured, the user can click the "Convert" button to initiate the model, which will process the document based on the selected options.

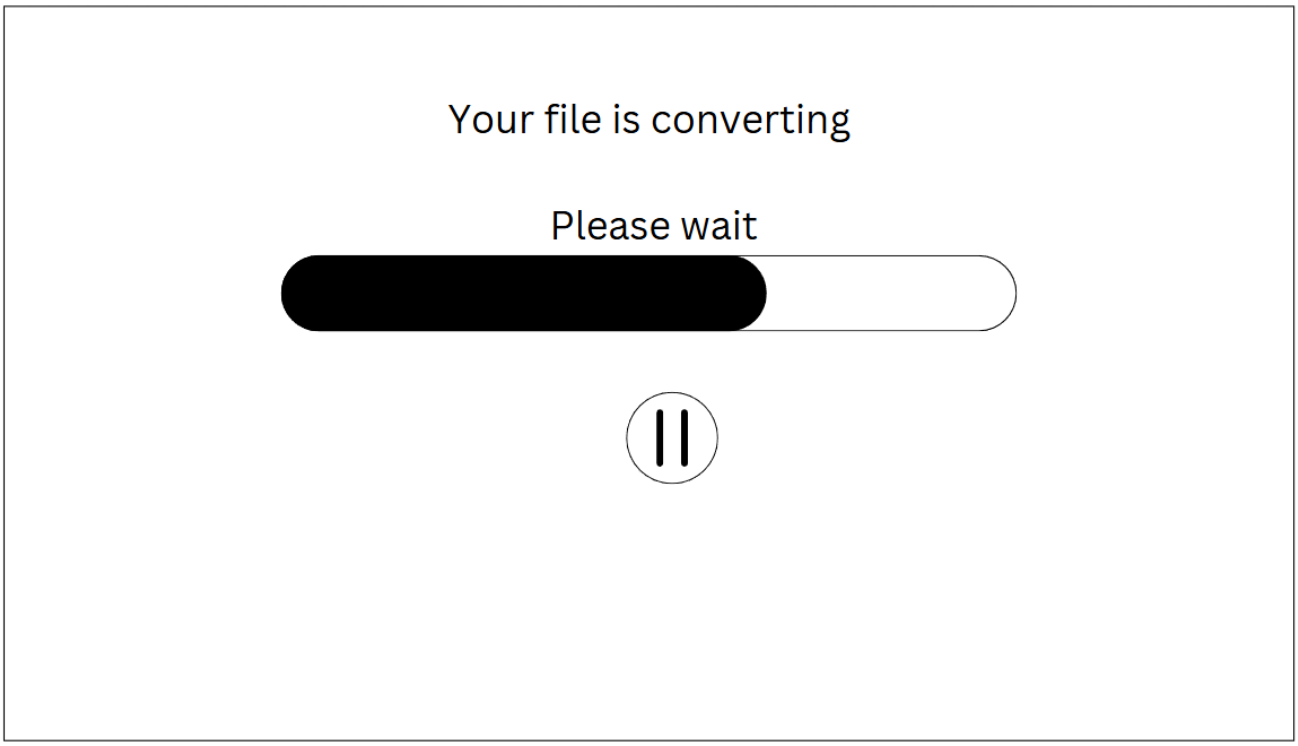


Figure 3.5: Loading Screen with Progress Tracking and Pause Functionality

Figure 3.5 illustrates the loading screen of the system. The next screen is the loading screen, designed to provide feedback to users during the document processing phase. The centrepiece of the screen is a prominent loading bar, which visually indicates the progress of the process, allowing users to track how much of the task has been completed. Below the loading bar, a pause button is available, enabling users to temporarily halt the processing if needed. This feature offers flexibility, such as when users need to adjust their system resources or make changes to the process. The screen also includes status messages or tooltips to keep users informed about the current stage of processing

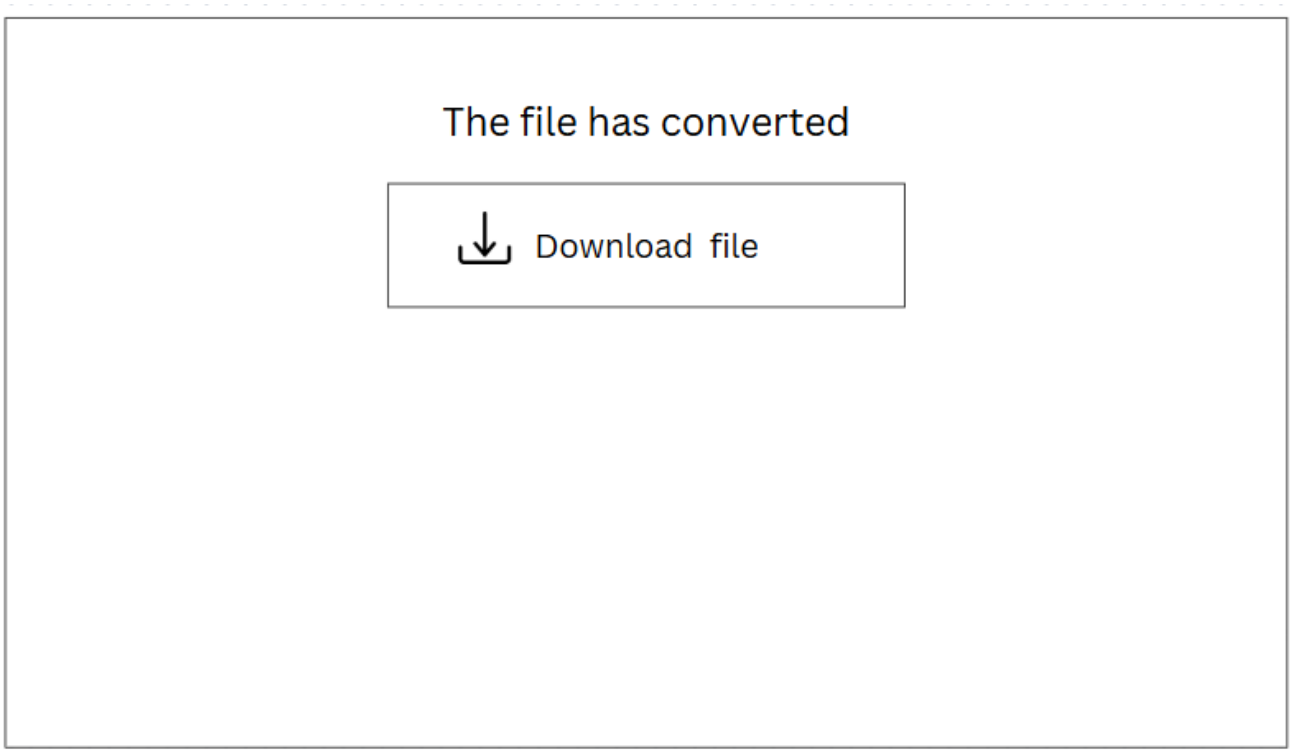


Figure 3.6: Result Screen for File Download

Figure 3.6 illustrates the result screen of the system. The wireframe for the result screen is designed to provide users with a clear and straightforward interface to download their converted file after processing. At the top of the screen, a confirmation message, "The file has converted," informs the user that the conversion process is complete. Beneath this message, a prominent download button labelled "download file" allows users to easily access their processed file.

3.3 Summary

In summary, Chapter 3 provides a comprehensive overview of the methodology for implementation in chapter 4 and the accompanying system wireframe The chapter systematically outlines the key stages of the project including data collection, preprocessing, feature extraction, and model development. By employing this structured methodology, the study can improve the precision and efficiency of handwritten character recognition.

CHAPTER 4
Implementation

4.0 Overview

This chapter discusses the procedure done to implement the machine learning model (CNN, CNN-LSTM and TrOCR) for Handwritten Character Recognition.

4.1 Hardware and Software Specification

The model training was performed on a system equipped with 12.7 GB allocated RAM and NVIDIA Tesla T4 GPU (15.0 GB VRAM) under Google Colab. Miniconda 23.1.0 was used to manage isolated Python environments, ensuring dependency control. Jupyter Notebook 6.5.2 facilitated interactive coding, real-time visualization, and documentation during training. Careful attention was given to version compatibility between hardware drivers and software libraries to maximize computational efficiency.

This project also utilise bunch of powerful stacks of Python libraries to facilitate efficient data processing, machine learning, and deep learning.

TensorFlow serves as the core framework for building and training neural networks. The TensorFlow enables efficient computation, multicore parallel processing for rapid model training. The TensorFlow modules also integrated with Keras's high-level API. This Api streamlines model development and experimentation.

For numerical computations and array manipulations, NumPy facilitates efficient manipulation of multi-dimensional arrays and matrices. It underpins preprocessing tasks, tensor operations, and data transformations across the pipeline.

Pandas Used for structured data management, including loading, cleaning, and organizing metadata associated with HCR images. Pandas streamlines the handling of image data, such as label, ensuring seamless integration with image datasets.

OpenCV is critical for image preprocessing tasks, including resizing, normalization, noise reduction, and augmentation. OpenCV's robust computer vision tools ensure retinal images are optimized for input into deep learning models.

The Matplotlib module is used for generate visualizations for exploratory data analysis (EDA), training curves, and performance metrics.

For the implementation of TrOCR model, The transformers library provides state-of-the-art pre-trained models, such as Vision Transformers (ViT), which are fine-tuned for retinal image classification. This enables experimentation with transformer-based architectures alongside traditional CNNs.

Additionally, Scikit-learn (Sklearn) Offers utilities for dataset splitting, metric calculation, and statistical evaluation. Key functions include train test split for partitioning data, classification report for precision/recall/F1-score summaries, and confusion matrix for visualizing model performance.

Streamlit is an open-source Python framework that transforms data scripts into shareable web apps in minutes. It eliminates frontend complexity so you can build interactive tools, dashboards, and visualizations with pure Python.

Together, these libraries create a versatile ecosystem for developing and deploying advanced data-driven solutions.

4.2 Data Collection

```
train = pd.read_csv('C:/Users/User/.cache/kagglehub/datasets/landlord/handwriting-recognition/versions/1/written_name_train_v2.csv')
valid = pd.read_csv('C:/Users/User/.cache/kagglehub/datasets/landlord/handwriting-recognition/versions/1/written_name_validation_v2.csv')

plt.figure(figsize=(15, 10))

for i in range(9):
    ax = plt.subplot(3,3,i+1)
    img_dir = 'C:/Users/User/.cache/kagglehub/datasets/landlord/handwriting-recognition/versions/1/train_v2/train/'+train.loc[i, 'FILENAME']
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    plt.imshow(image, cmap = 'gray')
    plt.title(train.loc[i, 'IDENTITY'], fontsize=12)
    plt.axis('off')

plt.subplots_adjust(wspace=0.2, hspace=-0.8)
```

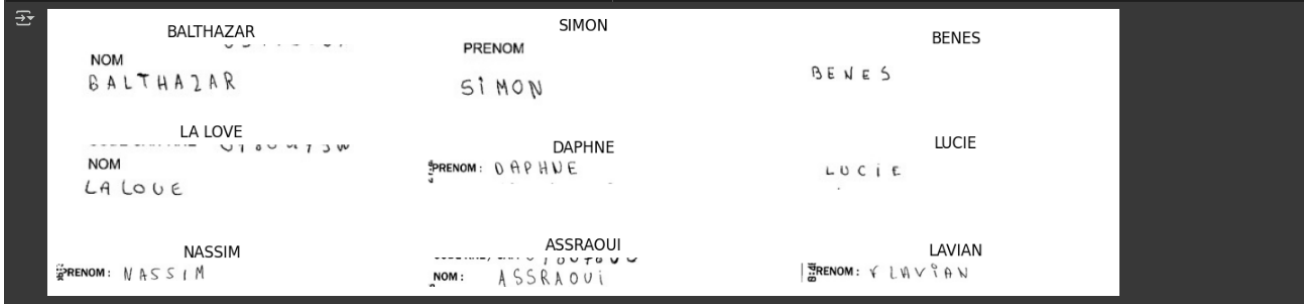


Figure 4.1 Code for Installing the Dataset and check the data

Figure 4.2 illustrates the process of loading the training and validation datasets and visualizing handwritten images alongside their labels. The code begins by importing the metadata from CSV files using ‘pd.read_csv()’,which includes two key columns: FILENAME (the image’s file path) and IDENTITY (the transcribed text label). The training and validation data is loaded

into the DataFrame. Subsequently, a 3x3 grid of subplots is generated to display nine sample images from the training set. For each image, the code constructs the full file path by combining a base directory with the filename from the DataFrame, reads the image in grayscale using OpenCV's cv2.imread(), and displays it with its corresponding label as the title. This visualization serves to validate the dataset's integrity by confirming that filenames align correctly with their transcribed labels, while also providing insight into the handwritten text's structure and variability.

```
[ ] print("Number of NaNs in train set          : ", train['IDENTITY'].isnull().sum())
    print("Number of NaNs in validation set : ", valid['IDENTITY'].isnull().sum())

    Number of NaNs in train set      : 565
    Number of NaNs in validation set : 78

    train.dropna(axis=0, inplace=True)#axis =0, removing rows otherwisw axis =1. removing columns
    valid.dropna(axis=0, inplace=True) #true means dropping
```

Figure 4.2: Snippet code for cleaning empty dataset

Before proceeding with model training, the datasets are inspected and cleaned to ensure data integrity. The code first checks for missing labels in the IDENTITY column of both the training and validation datasets using 'train['IDENTITY'].isnull().sum()' and 'valid['IDENTITY'].isnull().sum()', which print the number of incomplete entries. Rows containing missing labels are then removed. This step is critical to prevent errors during model training, as incomplete entries with no transcribed text labels would compromise learning accuracy. By filtering out these rows, the pipeline retains only valid, annotation-complete samples, ensuring the model trains on reliable and properly structured data. The empty data in the dataset within the train and validation are dropped.

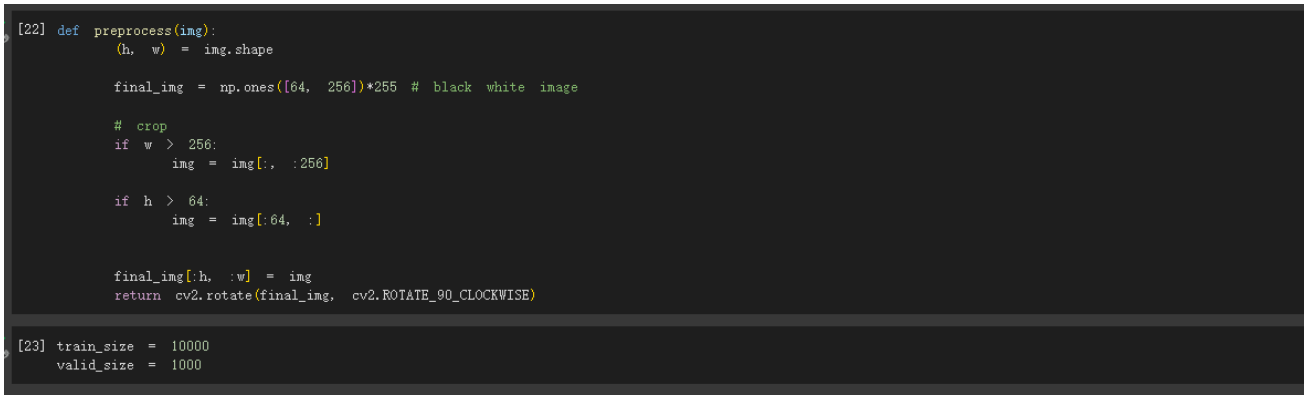
Figure 4.3 Code for cleaning unreadable dataset

After cleaning missing labels, the code explicitly investigates entries marked as "UNREADABLE" to analyse problematic cases in the training data. A dedicated DataFrame is created by filtering rows where the IDENTITY column equals "UNREADABLE", and its index is reset for easier iteration. A 3x3 grid of subplots then visualizes these samples using the same methodology. Images are loaded in grayscale from their file paths, displayed with the "UNREADABLE" label as titles, and axes are removed for clarity. This visualization serves two purposes: (1) to verify that the "UNREADABLE" label is consistently applied to genuinely illegible handwriting such as smudged text, incomplete characters, or ambiguous strokes. These samples might require special handling or removal to avoid introducing noise into the model. By explicitly isolating and reviewing these challenging examples, the pipeline ensures data quality standards are maintained before model training.

Figure 4.4: Code for convert case of image label and reset order of index

To ensure consistency in text labels for model training, all entries in the IDENTITY column of the training and validation datasets are converted to uppercase This normalization step eliminates case-sensitive variations, which simplifies the model's learning task by reducing unnecessary complexity in character recognition. After modifying the labels, the DataFrames indices are reset using 'reset_index(drop=True)' to remove gaps caused by earlier data removal steps which is dropping empty and unreadable data and ensure contiguous row numbering. This reorganization guarantees that subsequent operations, such as batch iteration or data splitting, function correctly with orderly index-aligned samples. Together, these steps enforce uniformity in label formatting and structural integrity in dataset organization, both critical for reliable model training and evaluation.

4.3 Image Preprocessing



```
[22] def preprocess(img):
        (h, w) = img.shape

        final_img = np.ones([64, 256])*255 # black white image

        # crop
        if w > 256:
            img = img[:, :256]

        if h > 64:
            img = img[:64, :]


        final_img[:h, :w] = img
        return cv2.rotate(final_img, cv2.ROTATE_90_CLOCKWISE)

[23] train_size = 10000
     valid_size = 1000
```

Figure 4.5 Function to normalize the images

Figure 4.6 illustrates the function to normalize the image. The function standardizes handwritten images to a fixed dimension of 64x256 pixels to ensure uniformity for model input. First, the input grayscale image's height (h) and width (w) are retrieved. A blank white canvas of size 64x256 is created. Images exceeding the target width or height are cropped to fit within these bounds, preserving the top-left region. The cropped image is then placed onto the blank canvas, effectively padding smaller images with white to maintain consistent dimensions. Finally, the image is rotated 90 degrees clockwise to correct orientation, as handwriting samples in the dataset are stored in a vertical layout incompatible with horizontal model input requirements.

To manage computational resources, the number of training dataset is subset to 10000 sample and number of validation dataset is subset to 1000 samples. This step balances dataset diversity

with practical training efficiency, particularly for prototyping or hardware-constrained environments. The preprocessing ensures all images are spatially aligned, normalized, and oriented correctly, addressing critical prerequisites for training a robust handwriting recognition model.

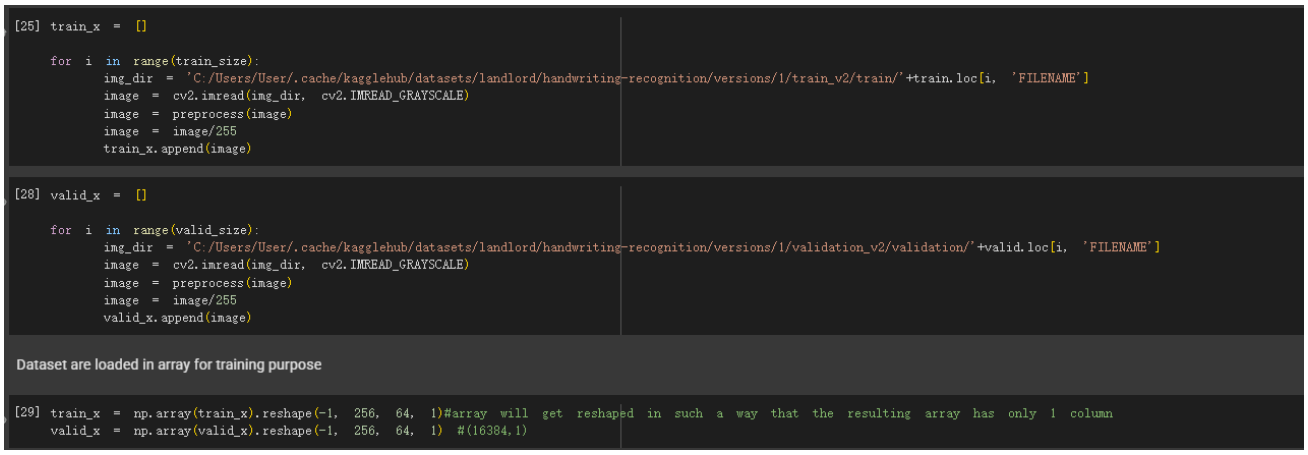
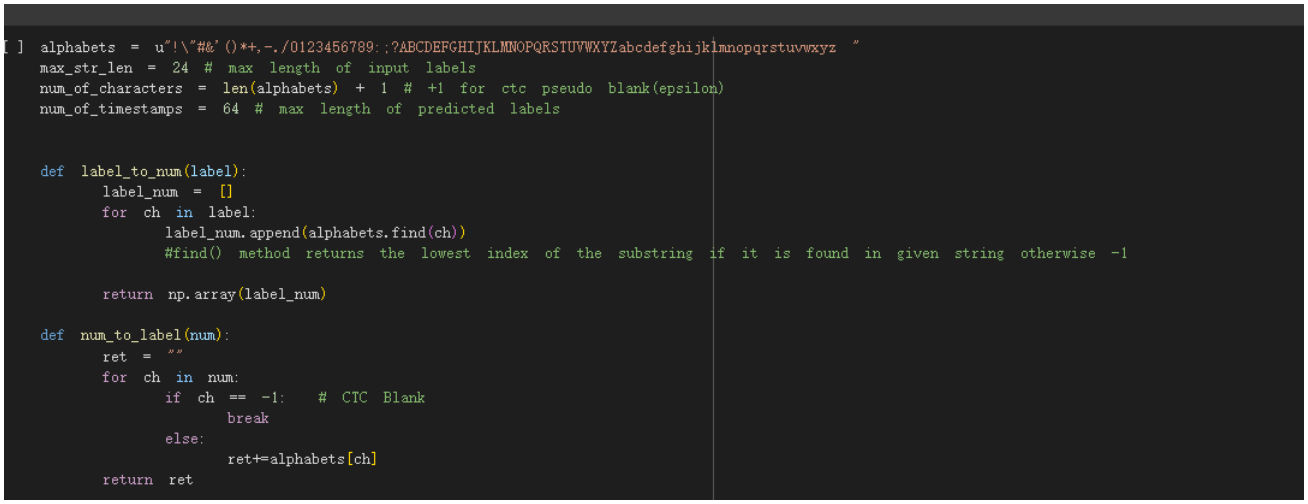


Figure 4.6: pipeline of preprocessing image and the dataset are loaded into array

The training and Validation data is loaded into pipeline for preprocessing image. The image file is taken from the folder, then the 'IMREAD_GRAYSCALE' function will load the image into black-and-white. Next, the normalize function that stated above will normalize the image. The pixel brightness of image also will be divide by 255 from 0-255 to 0-1. This can help the model learn more efficiently. The pre-processed image will be stored into a list. After loading all images, both lists are converted into NumPy arrays and reshaped into a 4D format. the first dimension represents the number of images (calculated automatically using **-1**), followed by the height (256 pixels), width (64 pixels), and a single channel (since the images are grayscale). This reshaping ensures the data matches the input structure expected by machine learning frameworks like TensorFlow or Keras. By standardizing the image size, normalizing pixel values, and formatting the data correctly, the code transforms raw images into a clean, consistent format ready for the model to process during training and evaluation.



```
[ ]  alphabets = u"!\"#&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz "
     max_str_len = 24 # max length of input labels
     num_of_characters = len(alphabets) + 1 # +1 for ctc pseudo blank(epsilon)
     num_of_timestamps = 64 # max length of predicted labels


     def label_to_num(label):
           label_num = []
           for ch in label:
                 label_num.append(alphabets.find(ch))
                 #find() method returns the lowest index of the substring if it is found in given string otherwise -1

           return np.array(label_num)

     def num_to_label(num):
           ret = ""
           for ch in num:
                 if ch == -1:    # CTC Blank
                       break
                 else:
                       ret+=alphabets[ch]
           return ret
```

Figure 4.7: The function for CTC Loss Labelling

Figure 4.8 illustrate the function of CTC Loss Labelling. Connectionist Temporal Classification (CTC) Loss is an objective function that allows a neural network to be trained for sequence transcription tasks without requiring any prior alignment between the input and output. (Huang, 2022). The CTC Loss Labelling is suitable for train sensing task such as image recognition, posture recognition and speech recognition. In the code, all character that include in a label is defined such as special symbols, digits, Uppercase letter, Lowercase letter and space with total of 95 characters. The maximum length of labels, total number of character and total number of timestamps also defined. The purpose of ‘label_to_num’ function is to convert text labels to numerical indices for CTC loss. Next, the “num_to_label” function is used to convert model outputs back to readable text.

```
train_y = np.ones([train_size, max_str_len]) * -1
train_label_len = np.zeros([train_size, 1])
train_input_len = np.ones([train_size, 1]) * (num_of_timestamps-2)
train_output = np.zeros([train_size])

for i in range(train_size):
    train_label_len[i] = len(train.loc[i, 'IDENTITY'])
    train_y[i, 0:len(train.loc[i, 'IDENTITY'])]= label_to_num(train.loc[i, 'IDENTITY'])
```

```
valid_y = np.ones([valid_size, max_str_len]) * -1
valid_label_len = np.zeros([valid_size, 1])
valid_input_len = np.ones([valid_size, 1]) * (num_of_timestamps-2)
valid_output = np.zeros([valid_size])

for i in range(valid_size):
    valid_label_len[i] = len(valid.loc[i, 'IDENTITY'])
    valid_y[i, 0:len(valid.loc[i, 'IDENTITY'])]= label_to_num(valid.loc[i, 'IDENTITY'])
```

```
print('True label : ',train.loc[100, 'IDENTITY'] , '\ntrain_y : ',train_y[100],'\ntrain_label_len : ',train_label_len[100],
      '\ntrain_input_len : ', train_input_len[100])
```

Figure 4.8: The pipeline for labelling CTC and test the output of label

The code prepares labels for the Connectionist Temporal Classification (CTC) loss, which is essential for training sequence-to-sequence models on variable-length handwriting recognition tasks. For both training and validation datasets, three critical arrays are initialized: train_y and valid_y, which store numerical indices of characters padded with -1 to a fixed length of max_str_len (24 characters), ensuring uniform input dimensions; train_label_len and valid_label_len, which record the actual length of each text label to distinguish valid characters from padding during loss calculation; and train_input_len and valid_input_len, set to num_of_timestamps - 2, representing the number of time steps the model uses to align predictions. During processing, each text label (e.g., "DOG") is converted to numerical indices via label_to_num, with its length stored to exclude padding from loss computation. However, a critical mismatch arises in the handling of the CTC blank symbol: the code incorrectly uses -1 (reserved for padding) as the blank, whereas the blank should be assigned a dedicated index (e.g., 94, corresponding to len(alphabets)). This conflation may cause decoding errors, as padding positions are erroneously interpreted as blanks. The formatted arrays (train_y, train_label_len, train_input_len) structure the data for CTC loss compatibility, enabling the model to learn alignments between variable-length text labels and fixed-length image features. Identical steps are applied to the validation set to ensure consistency in evaluation.

4.4 Model Specifications

4.4.1 CNN Model Parameter Settings

Table 4.1: CNN model Parameters Setting

Training Detail	CNN Model
Input Layers	256 x 64 x 1
Convolutional Layers	Conv2D (32, 3, padding=’same’, activation= ‘relu'), Maxpool2D ((2, 2)), Conv2D (64, 3, padding=’same’, activation= ‘relu'), Maxpool2D ((2, 2)), Conv2D (128, 3, padding=’same’, activation= ‘relu'), Maxpool2D ((2, 2)),
Fully Connected Layers	Flatten () Dense(64, activation='relu'), Dense(4, activation = ‘softmax’)
Number of Trainable Parameters	2,432,080
Number of Non-Trainable Parameters	448
Total Number of Parameters	2,432,528
Loss Function	lambda function
Optimizer	Adam
Learning Rate	0.0001
Epoch	1000

The CNN model was designed with an input layer of 256x64x1, representing the spatial dimensions and single-channel grayscale structure of the input images. The architecture begins with three sequential convolutional blocks for feature extraction: The first block employs a Conv2D layer with 32 filters (3×3 kernel, same padding, ReLU activation) followed by 2×2 max-pooling. This pattern repeats in the second block with 64 filters and in the third block with 128 filters, each followed by identical max-pooling. This hierarchical design progressively extracts higher-level features while reducing spatial dimensions through down-sampling. The output then transitions to fully connected layers via flatten into a 1D vector, processed by a Dense layer with 64 units (ReLU activation, He normal initialization). The model contains 2,432,528 total parameters, with 2,432,080 trainable weights updated during training and 448 non-trainable parameters. For optimization, Adam was used with a low learning rate of 0.0001 to enable precise weight adjustments, while a custom lambda function served as the loss function, indicating adaptation to specialized task requirements. The extended training regimen of 1000 epochs suggests complex pattern learning from

substantial data, though no early stopping mechanism was documented. This configuration prioritizes detailed feature extraction from grayscale inputs while maintaining computational efficiency through progressive down-sampling.

4.4.2 CNN-LSTM Model Parameter Settings

Table 4.2: CNN-LSTM Model Parameter Settings

Training Detail	CNN Model
Input Layers	256 x 64 x 1
Convolutional Layers	Conv2D (32, 3, padding=’same’, activation= ‘relu'), Maxpool2D ((2, 2)), Conv2D (64, 3, padding=’same’, activation= relu'), Maxpool2D ((2, 2)), Conv2D (128, 3, padding=’same’, activation= ‘relu'), Maxpool2D ((2, 2)),
Fully Connected Layers	Reshape () Dense(64, activation='relu', kernel_initializer='he_normal')
Recurrent Layers	Bidirectional (LSTM (256, return_sequences=True) Bidirectional (LSTM (256, return_sequences=True)
Number of Trainable Parameters	2,432,080
Number of Non-Trainable Parameters	448
Total Number of Parameters	2,432,528
Loss Function	lambda function
Optimizer	Adam
Learning Rate	0.0001
Epoch	1000

The CNN-LSTM model was designed with an input layer of 256 x 64 x 1, representing the dimensions of the input images. The model architecture consists of three convolutional layers for feature extraction and two Recurrent Layers for sequence labelling. The convolutional layers include Conv2D with 32 filters, a kernel size of 3, the same padding, and a ReLu activation function, followed by Maxpool2D layers for down 58 sampling. This pattern is

repeated with Conv2D layers of 64 and 128 filters. The fully connected layers consist of a Reshape layer to convert the dimension of output into 64 x 1024. It followed by a Dense layer with 64 units and a ReLU activation function with he_normal as kernel. Next, the data will be forward to two recurrent layer which has 256 unit for each. The layer is used to predict the character based on the alphabet with the highest predicted value given by the model. Finally, the model ends with a Dense layer of 4 units and a SoftMax activation function. It utilizes the SoftMax activation function to produce a probability distribution across all alphabets.

The model has a total of 2,432,528 parameters, including 2,432,080 parameters are trainable and 448 parameters are non-trainable. The loss function used is CTC (Connectionist Temporal Classification) lambda Function. The CTC lambda function is a specialized loss function used in sequence-to-sequence learning tasks where the alignment between the input such as image) and the output such as text sequence is unknown or variable. The Lambda function is used because it can computes the loss efficiently across a batch of sequences output. The model was trained for 1000 epochs with Adam as the optimizer, which has a learning rate of 0.0001. The early stopping callback is implemented into it, with a patient of 5. This means that the training process is continuously monitored based on the validation loss to prevent overfitting.

4.4.3 TrOCR Model Parameter Settings

Table 4.3: TrOCR Model Parameter Settings

Component	Layer Type	Key Parameters	Purpose
Encoder (ViT)	Patch Embeddings	Conv2d(3→768, kernel=(16,16), stride=(16,16)	Extracts 16×16 image patches → 768D embeddings
	ViT Layer (×12)	- SelfAttention(768→768) - Intermediate(768→3072) - Output(3072→768)	Transformer processing with 12 layers
	LayerNorm	eps=1e-12, 768D	Normalizes encoder outputs
	EmbedTokens	Embedding(50265→1024, padding_idx=1)	Input token embeddings (vocab size 50265)

Decoder	Sinusoidal Pos Embedding	1024D	Adds positional information to embeddings
	TrOCR Decoder Layer (×12)	- Self Attention(1024→1024) - Encoder --- -Attention(768→1024) - FFN(1024→4096→1024)	Processes tokens with self/cross-attention
	Output Projection	Linear(1024→50265)	Generates logits over vocabulary

TrOCR model employs a transformer-based encoder-decoder architecture tailored for optical character recognition. The encoder is named as Vision Language Transformer (VLT). It begins by segmenting input images into 16×16 patches, converting them into 768D embeddings via a convolutional layer. Next, the VLT layers then process these embeddings using self-attention and feed-forward networks (FFNs). The process will repeat 12 time by going through 12 ViT layers, The purpose of ViT Layer is capturing spatial relationships and hierarchical visual features progressively. Layer normalization stabilizes the encoder outputs before they are passed to the decoder.

The decoder starts by embedding input tokens into 1024D vectors, augmented with sinusoidal positional encodings to preserve sequence order. Twelve TrOCR layers follow, each combining self-attention (for token-token interactions), encoder cross-attention (to focus on image regions), and an expanded FFN for feature transformation. Finally, an output projection layer generates logits over a 50,265-token vocabulary, enabling text prediction.

4.4.4 Modelling

```
input_data = Input(shape=(256, 64, 1), name='input')

inner = Conv2D(32, (3, 3), padding='same', name='conv1', kernel_initializer='he_normal')(input_data)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max1')(inner)

inner = Conv2D(64, (3, 3), padding='same', name='conv2', kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max2')(inner)
inner = Dropout(0.3)(inner)

inner = Conv2D(128, (3, 3), padding='same', name='conv3', kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(1, 2), name='max3')(inner)
inner = Dropout(0.3)(inner)

# CNN to RNN
inner = Reshape(target_shape=((64, 1024)), name='reshape')(inner)
inner = Dense(64, activation='relu', kernel_initializer='he_normal', name='dense1')(inner)

## RNN
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm1')(inner)
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm2')(inner)

## OUTPUT
inner = Dense(num_of_characters, kernel_initializer='he_normal',name='dense2')(inner)
y_pred = Activation('softmax', name='softmax')(inner)

model = Model(inputs=input_data, outputs=y_pred)
model.summary()
```

Figure 4.9: Snippet Code for building the CNN-LSTM model

Figure 4.10 shows the code for building CNN-LSTM model. The model built based on the parameter setting stated above. The model combines convolutional neural networks (CNN) for spatial feature extraction and bidirectional LSTM for sequence modelling and optimized using Connectionist Temporal Classification (CTC) loss. At the output layer, the logits are converted to probabilities for CTC loss computation using Softmax as activation layer.

4.4.5 Compile model

```
[ ] # the ctc loss function
    def ctc_lambda_func(args):
        y_pred, labels, input_length, label_length = args
        # the 2 is critical here since the first couple outputs of the RNN
        # tend to be garbage
        y_pred = y_pred[:, 2:, :]

        # Wrap the ctc_batch_cost call with tf.function to ensure it's executed eagerly
        # and any variables are created during the first call only.
        @tf.function(jit_compile=False)
        def ctc_batch_cost_wrapper(labels, y_pred, input_length, label_length):
            return K.ctc_batch_cost(labels, y_pred, input_length, label_length)

        return ctc_batch_cost_wrapper(labels, y_pred, input_length, label_length)
```

```
[ ] labels = Input(name='gtruth_labels', shape=[max_str_len], dtype='float32')
    input_length = Input(name='input_length', shape=[1], dtype='int64')
    label_length = Input(name='label_length', shape=[1], dtype='int64')

    ctc_loss = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([y_pred, labels, input_length, label_length])
    model_final = Model(inputs=[input_data, labels, input_length, label_length], outputs=ctc_loss)
```

```
# the loss calculation occurs elsewhere, so we use a dummy lambda function for the loss
file_path_best = "C_LSTM_best.keras"

model_final.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=Adam(learning_rate=0.0001))

checkpoint = ModelCheckpoint(filepath=file_path_best,
                             monitor='val_loss',
                             verbose=1,
                             save_best_only=True,
                             mode='min')

callbacks_list = [checkpoint]
```

Figure 4.10: Snippet Code for Compiling the model

Compiling a model is necessary after building it. By compiling the model, we can configure additional settings for training the model. In this code, the ctc loss function, optimizer, checkpoints and callback are include. The CTC loss function quantifies the difference between the predicted output and the true output labels, representing the objective the model seeks to minimize. The Adam optimizer determines the algorithm and updates the rules used to adjust the model's weights and biases during training. Callbacks such as model checkpoint, early stopping, and learning rate scheduler provided by TensorFlow offer convenient tools for monitoring and managing the model during the training process. In this project, the loss will continue to use in sparse categorical cross-entropy, while the metrics will track accuracy.

4.4.6 Train Model

```
from tensorflow.keras.callbacks import EarlyStopping


early_stopping = EarlyStopping(
        monitor='val_loss',    # Metric to monitor
        patience=5,                 # Number of epochs to wait before stopping
        verbose=1,                  # Show messages when stopping
        mode='min',                 # Look for decreasing values
        restore_best_weights=True   # Keep best weights when stopping
)

callbacks_list = [
        checkpoint,               # Your existing ModelCheckpoint
        early_stopping          # New early stopping callback
]

history = model_final.fit(
        x=[train_x, train_y, train_input_len, train_label_len],
        y=train_output,
        validation_data=(
                [valid_x, valid_y, valid_input_len, valid_label_len],
                valid_output
        ),
        callbacks=callbacks_list,
        verbose=1,
        epochs=90,
        batch_size=128,
        shuffle=True
)
```

```
78/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step - loss: 2.4358
Epoch 77: val_loss did not improve from 3.12051
79/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 40s 263ms/step - loss: 2.4348 - val_loss: 3.1246
Epoch 78/90
78/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step - loss: 2.2884
Epoch 78: val_loss improved from 3.12051 to 3.01747, saving model to C_LSTM_best.keras
79/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 42s 277ms/step - loss: 2.2894 - val_loss: 3.0175
Epoch 79/90
78/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step - loss: 2.2241
Epoch 79: val_loss did not improve from 3.01747
79/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 40s 266ms/step - loss: 2.2247 - val_loss: 3.0555
Epoch 80/90
78/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 0s 259ms/step - loss: 2.2014
Epoch 80: val_loss did not improve from 3.01747
79/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 41s 273ms/step - loss: 2.2019 - val_loss: 3.0573
Epoch 81/90
78/79 ━━━━━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step - loss: 2.1388
Epoch 81: val_loss improved from 3.01747 to 3.01015, saving model to C_LSTM_best.keras
```

Figure 4.11: Snippet Code for training the model

The final model was trained over 50 epochs with a batch size of 128, using a shuffled dataset to enhance generalization and reduce the risk of overfitting. The training process employed multiple input streams, including train_x, train_y, train_input_len, and train_label_len, alongside a corresponding target output, train_output. Validation was performed on a separate set comprising valid_x, valid_y, valid_input_len, and valid_label_len, with valid_output as the validation target. This configuration suggests the use of a multi-input architecture, potentially aligned with models incorporating Connectionist Temporal Classification (CTC) loss or sequence-based learning. A set of callbacks was also utilized to optimize the training process, including mechanisms for early stopping, best model checkpointing, and adaptive learning rate adjustment. The patience for early stopping is 5 where the model will stop if the validation loss do not improve within 5 epoch. In this training, the callback was stopped at 90 epoch with the

best epoch is 85, best loss of 1.7952 and best validation loss is 2.89242. These components collectively ensured effective training convergence, safeguarded against overfitting, and facilitated the retention of the most performant model based on validation metrics.

4.5 Performance Evaluation

4.5.1 Learning Curve

```
[ ]  plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('Model  Loss')
     plt.ylabel('Loss')
     plt.xlabel('Epoch')
     plt.legend(['Train',  'Test'],  loc='upper  left')
     plt.gca().invert_yaxis()    #  Invert  y-axis
     plt.show()
```

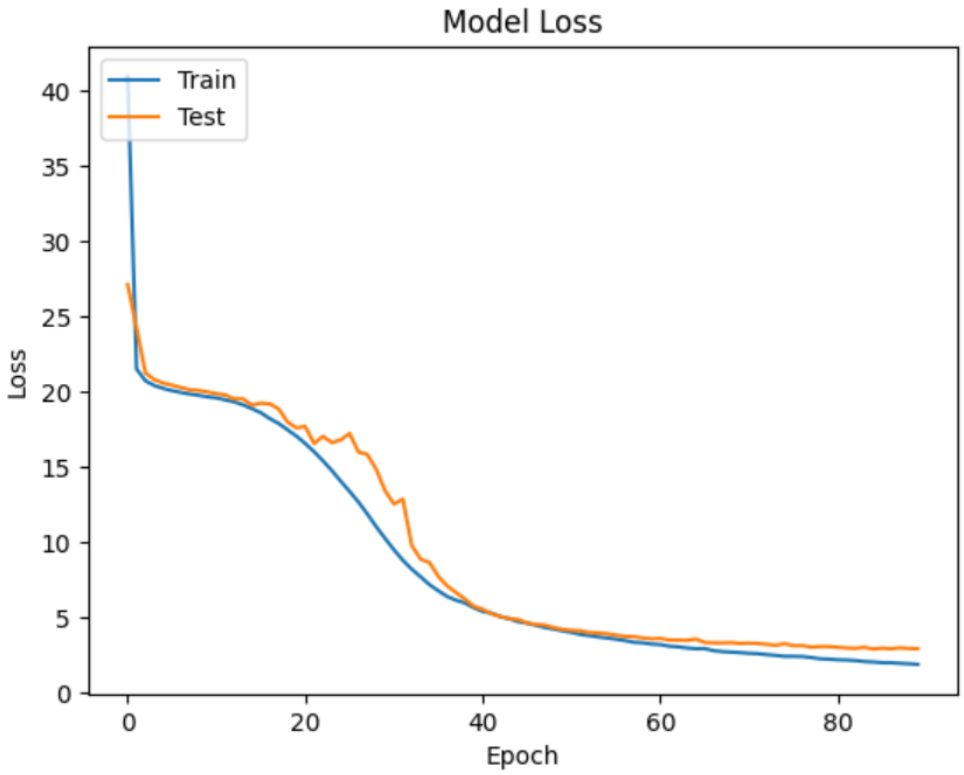


Figure 4.12 Learning Curve of the CNN-LSTM model

Figure 4.13 shows the learning curve of CNN-LSTM model. By using the loss and validation loss data from training history, the plot of the loss of the model over training and validation epochs is generated. This plot provides a visual representation of how well the model is learning over time and can be used to identify issues such as overfitting or underfitting.

4.5.2 Evaluate Model

```
y_true = valid.loc[0:valid_size, 'IDENTITY']
correct_char = 0
total_char = 0
correct = 0

for i in range(valid_size):
        pr = prediction[i]
        tr = y_true[i]
        total_char += len(tr)

        for j in range(min(len(tr), len(pr))):
                if tr[j] == pr[j]:
                        correct_char += 1

        if pr == tr :
                correct += 1

print('Character Error Rate (CER) : %.2f%%' %(100-(correct_char*100/total_char)))
print('Words Error Rate (WER)      : %.2f%%' %(100-(correct*100/valid_size)))
```

```
Character Error Rate (CER) : 25.48%
Words Error Rate (WER)     : 43.10%
```

Figure 4.13 Evaluate the model

Figure 4.14 illustrates how to evaluate the performance of CNN-LSTM model on the test dataset. The method comparing the prediction of model with the original label of dataset. This evaluate output the error rate of the model. In this case, the model achieved a Character Error Rate (CER) of 25.48% and Word Error Rate (WER) of 43.10%.

4.5.3 Predict Model

```
test = pd.read_csv('/kaggle/input/handwriting-recognition/written_name_test_v2.csv')

plt.figure(figsize=(15, 10))
for i in range(16):
        ax = plt.subplot(4, 4, i+1)
        img_dir = '/kaggle/input/handwriting-recognition/test_v2/test/'+test.loc[i, 'FILENAME']
        image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
        plt.imshow(image, cmap='gray')

        image = preprocess(image)
        image = image/255.
        pred = model.predict(image.reshape(1, 256, 64, 1))
        decoded = K.get_value(K.ctc_decode(pred, input_length=np.ones(pred.shape[0])*pred.shape[1],
                                           greedy=True)[0][0])
        plt.title(num_to_label(decoded[0]), fontsize=12)
        plt.axis('off')

plt.subplots_adjust(wspace=0.2, hspace=-0.8)
```

Figure 4.14 Predicts the model

Figure 4.19 shows the prediction phase of the CNN-LSTM model. This code output the prediction of the model and pre-processed image from test dataset. This will be used to compare the prediction of model with the test dataset to evaluate the model's performance

4.6 System Implementation

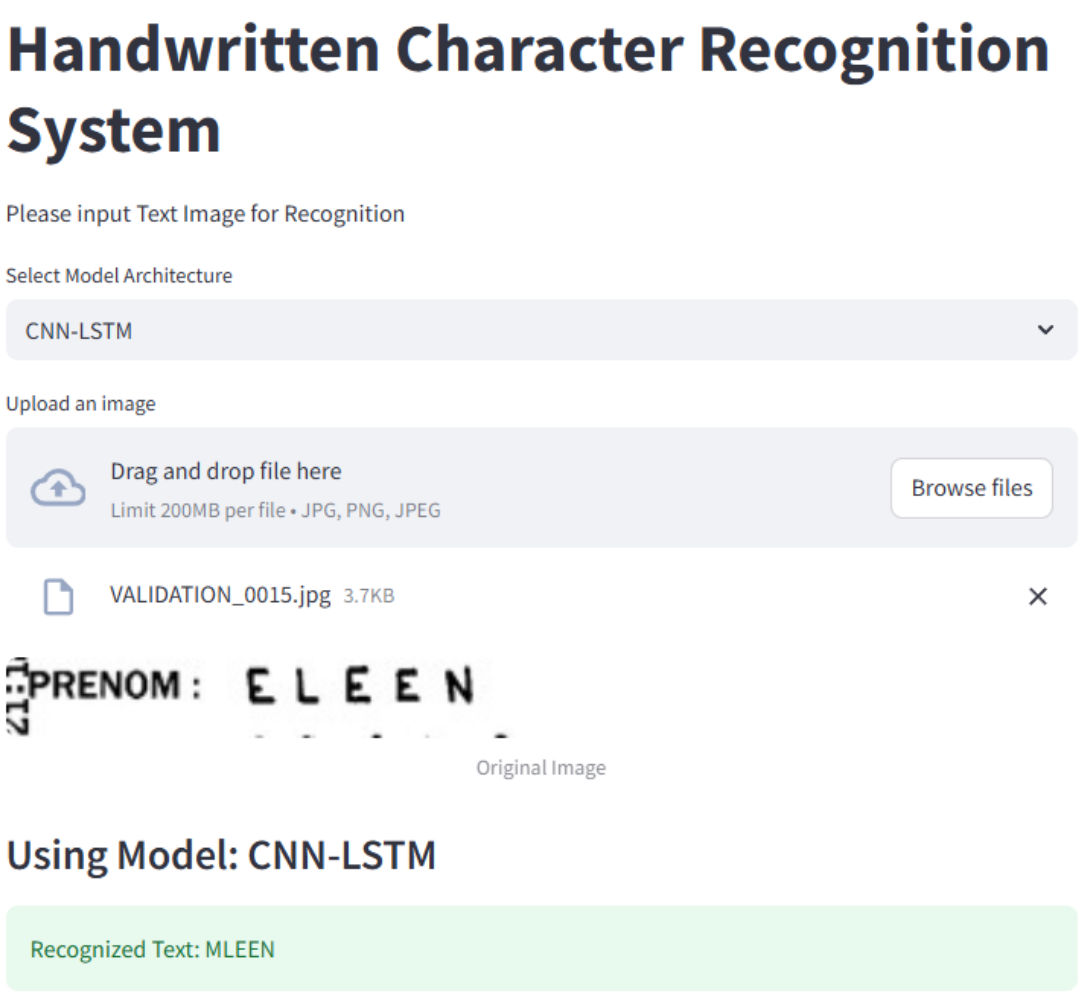


Figure 4.14 User Interface of HCR system

Figure 4.15 illustrates the user interface of the Handwritten Character Recognition system. Upon entering the system, users select a model architecture from CNN, CNN-LSTM, or TrOCR. After model selection, users drag and drop a locally stored handwritten character image for processing. The system then outputs the recognized text. For example, when processing the 15th image from the validation set whose label is 'ELEEN' but the model incorrectly recognized it as 'MLEEN. This error likely arises from visual ambiguity between 'E' and 'M' due to connected strokes or ink artifacts, combined with limitations in the model's capacity to generalize to atypical handwriting variations.

4.7 Summary

The chapter describe the implementation of Handwritten Character Recognition using Machine Learning Model. It Started from data distribution, data cleaning, image preprocessing, CTC Loss labelling, build Model, compile model, training model and model Evaluation.

CHAPTER 5

Result and Discussion

5.0 Overview

In this Chapter, we explore and compare the performance of there different model with are CNN, LSTM-CNN and TrOCR model. Each model offer unique architectural features and significants. The chapter investigate the performance matrics for each model and conduct a comprehensive evaluation of each model's performance.

5.1.1 Result comparison and analysis

Table 5.1: Table of model against performance metrics

Model	**Training Accuracy**	**Validation Accuracy**	**Word Error Rate (WER)**	**Character Error Rate (CER)**	**Training Duration**
CNN model	0.8585	0.8544	0.6290	0.4976	1 hour 14 minutes
CNN-LSTM model	0.8799	0.8663	0.2548	0.4310	1 hour 45 minutes
TrOCR Model	0.9021	0.7924	0.1845	0.3624	2 hour 13 minutes

Based on the training accuracy results in the table above, the TrOCR model achieved the highest performance (0.9021), followed by the CNN-LSTM model (0.8799), and the CNN model (0.8585). This indicates that TrOCR demonstrated the strongest ability to learn patterns from the training dataset, though its higher complexity may have contributed to this result.

Regarding validation accuracy, the CNN-LSTM model yielded the best generalization capability (0.8663), with the CNN model performing comparably (0.8544). The TrOCR model showed a significant drop (0.7924), suggesting potential overfitting to the training data despite its initial high training performance.

In terms of training duration, the CNN model was the most efficient (1 hour 18 minutes), while the CNN-LSTM model required moderately more time (1 hour 45 minutes). The TrOCR model

demanded the longest training period (2 hours 13 minutes), reflecting its computational complexity relative to the other architectures.

For Word Error Rate(WER) and Character Error Rate (CER), the TrOCR model achieved lowest WER(0.1845) and CER(0.3624), followed by CNN-LSTM model with 0.2548 for WER and 0.4310 for CER. The CNN model show the highest error rate with 0.6290 for WER and 0.4976 for CER.

5.1.2 Classification Report

Table 5.2 Table of precision, recall and F1-Score for each models

Model	Precision	Recall	F1-Score
CNN model	0.5172	0.4677	0.4872
CNN-LSTM model	0.6904	0.6895	0.6858
TrOCR Model	0.8234	0.8342	0.8236

Table 5.2 shows the precision, recall and F1-score of CNN, CNN-LSTM and TrOCR model. After analysing the results presented in the table, it is evident that the CNN model had the lowest overall performance among the three models. It achieved a precision of 0.5172, indicating that when the CNN predicted a character, it was correct approximately 51.72% of the time. The recall was 0.4677, meaning it correctly identified 46.77% of the actual character instances. The relatively low F1-score of 0.4872 reflects an imbalanced trade-off between precision and recall, with a considerable presence of both false positives and false negatives. This suggests that the CNN struggled to consistently and accurately classify handwritten characters, particularly when faced with character-level confusion or out-of-distribution samples.

In contrast, the CNN-LSTM model showed substantial improvement across all metrics. It achieved a precision of 0.6904 and a recall of 0.6895, both indicating that the model was able to correctly predict and identify approximately 69% of the characters. The F1-score of 0.6858 further highlights its balanced performance, suggesting that the integration of temporal sequence modelling via LSTM enhanced the model's ability to capture the spatial patterns in handwritten character sequences more effectively than the standalone CNN. This is supported

by the confusion matrix, which revealed a reduction in misclassifications and fewer predictions categorized under the "Other" class.

The TrOCR model outperformed both CNN-based models by a significant margin. It achieved a precision of 0.8234 and a recall of 0.8342, meaning the model was both highly accurate and sensitive in identifying characters, correctly recognizing over 83% of all actual instances while maintaining a low false positive rate. With an F1-score of 0.8236, the TrOCR model demonstrated the best balance between precision and recall. This superior performance can be attributed to the model's transformer-based architecture, which excels at capturing long-range dependencies and contextual relationships in image-to-text tasks.

Overall, the TrOCR model is the most effective in classifying handwritten characters, followed by the CNN-LSTM, while the baseline CNN model exhibits relatively weak performance. The consistent improvement from CNN to CNN-LSTM to TrOCR suggests that incorporating sequential modeling and transformer architectures significantly enhances recognition accuracy and robustness in handwritten character recognition tasks.

5.2 Discussion

The performance differences among the CNN, CNN-LSTM, and TrOCR models is differs due to fundamentally from their architectural capabilities and their suitability for the sequential nature of handwritten text recognition.

The CNN model achieved the lowest accuracy and highest error rates (WER: 0.6290, CER: 0.4976), along with poor precision (0.5172), recall (0.4677), and F1-score (0.4872). This weakness is attributable to its exclusive focus on spatial feature extraction while effective for isolated character recognition. CNNs lack any mechanism to model the crucial temporal dependencies and contextual relationships between consecutive characters in handwriting. This limitation leads to high confusion rates, especially for ambiguous characters or contextual variations. This may explain its significant number of false positives/negatives and poor sequence-level accuracy. Its primary advantage was computational efficiency which evidenced by the shortest training time (1h 18m).

The CNN-LSTM model demonstrated a substantial improvement, achieving the best validation accuracy (0.8663) and significantly better classification metrics (Precision: 0.6904, Recall:

0.6895, F1-score: 0.6858) and lower error rates (WER: 0.2548, CER: 0.4310) than the CNN. This enhancement is due to the synergistic combination: the CNN extracts robust visual features, while the LSTM layers explicitly model the sequential dependencies between these features. This allows the model to resolve ambiguities using contextual information from neighbouring characters, leading to better generalization on unseen data (as shown by the high validation accuracy) and a significant reduction in sequence-level errors.

Finally, the TrOCR model exhibited the highest raw recognition power on the training data (accuracy: 0.9021) and achieved the best classification scores (Precision: 0.8234, Recall: 0.8342, F1-score: 0.8236) and lowest error rates (WER: 0.1845, CER: 0.3624). This superior performance stems from its Transformer architecture, whose self-attention mechanism excels at capturing complex, long-range contextual dependencies across the entire input sequence simultaneously. However, this power comes with challenges: the model's high complexity led to the longest training time (2h 13m) among all models. This suggests TrOCR, with its vast capacity may have memorized training specifics rather than learning robust general patterns, likely due to insufficient data volume/diversity relative to its parameter count. This indicate this model need for strategies like stronger regularization or more data to fully leverage its potential on this task.

The progression from CNN to CNN-LSTM to TrOCR clearly underscores the critical importance of effectively modeling sequence context for accurate handwritten text recognition.

5.3 Summary

In summary, this chapter introduces the performance metrics and analysis for three different machine learning model. Their performance are evaluated for handwritten character recognition. The metrics compared are training, validation and testing accuracy, word error rate, character error rate, training duration, confusion matrix and classification report.

CHAPTER 6

Conclusion

6.0 Overview

The important results and contributions of the study on handwritten character recognition by using a machine learning approach are summarized in Chapter 6 to bring the thesis to a close. This chapter also discusses the study's weaknesses and recommends other research directions.

6.1 Contribution

This project makes significant contributions to multilingual handwritten character recognition by directly achieving the objectives stated as section 1.2.

First, it successfully establishes critical baseline accuracy metrics for three distinct deep learning architectures (CNN, CNN-LSTM, TrOCR) on a standardized dataset. The benchmarking was evaluated through comprehensive metrics including precision, recall, F1-score, WER, and CER. These Performance metrics can provides essential reference points for future research. The clear performance hierarchy established fulfils the primary objective of defining fundamental accuracy benchmarks for model comparison.

Second, the project directly addresses its objective to examine model performance under varying data conditions. Through detailed confusion matrix analysis and comparative evaluation, it systematically identifies how handwriting style variations (particularly visually similar characters) and inherent image ambiguities impact robustness. The study reveals how each architecture handles these challenges. For instance, the CNN's susceptibility to confusion under variation, the CNN-LSTM's improved contextual resilience leading to superior generalization and the TrOCR's exceptional contextual power offset by sensitivity to overfitting without sufficient regularization. This analysis fulfils the objective of identifying unique model strengths and limitations regarding handwriting diversity and noise, providing crucial insights for real-world application suitability in tasks like noisy document digitization or diverse handwriting transcription.

Beyond these core objectives, this project bridges the gap between theoretical research and practical application by deploying the machine learning models in an interactive web application. This tool enables real-time conversion of user-uploaded handwritten

characters/notes into digital text which demonstrating the tangible utility of the evaluated architectures.

6.2 Limitation

Despite the valuable contributions of this study to the field of handwritten character recognition, several limitations should be acknowledged. One major constraint was the limited computational power available for model training and evaluation. The use of modest hardware resources restricted the ability to train more complex architectures with larger batch sizes or longer training epochs, particularly for transformer-based models such as TrOCR. As a result, the full potential of some models may not have been realized, possibly impacting their final performance scores and training stability.

The model evaluation process also relied primarily on internal validation using the same dataset from which training and testing subsets were drawn. This approach limits the assessment of model robustness against external or real-world data. Handwritten input varies significantly between individuals and cultural contexts, and further evaluation using external datasets is essential to verify the models' effectiveness in diverse real-life applications.

Lastly, the scalability and deployment of the models in practical settings remain an concern. Real-time applications, such as mobile handwriting recognition tools or embedded systems, demand lightweight models with fast inference capabilities. Due to the computational requirements of the current models, especially TrOCR, deploying them in constrained environments would require further model optimization, such as pruning or quantization. Future work should explore these directions to enable broader accessibility and application of the developed recognition systems.

6.3 Future Works

Building upon the findings and limitations of this study, several avenues for future research in handwritten character recognition can be pursued. Firstly, to address the constraint of limited character diversity, future work should focus on collecting a larger and more comprehensive dataset that includes a wider variety of handwritten characters across different scripts, writing styles, and languages. This would not only improve model training but also enhance the

generalizability of the system to recognize a broader spectrum of handwriting patterns in real-world scenarios.

In terms of improving model performance, future studies could explore the use of transfer learning by leveraging pre-trained models trained on large-scale document image or character datasets. Fine-tuning these models specifically for handwritten character recognition tasks can significantly boost performance, especially for models like TrOCR and CNN-LSTM that benefit from contextual understanding and sequential features.

Additionally, future work could investigate the use of ensemble learning techniques, where predictions from multiple models (e.g., CNN, CNN-LSTM, and TrOCR) are combined to form a more robust recognition system. Ensemble approaches can capitalize on the unique strengths of each model and potentially mitigate individual weaknesses, resulting in improved overall accuracy and stability in character classification.

Moreover, further research could focus on model optimization techniques such as pruning, quantization, or knowledge distillation to reduce the computational cost of large models. This would support deployment in resource-constrained environments, such as mobile applications or embedded systems, making the recognition system more accessible and efficient for real-time usage.

6.5 Summary

This project successfully explored and evaluated various deep learning for the task of handwritten character recognition. Through comprehensive analyses, the study demonstrated the strengths and limitations of each model in recognizing a diverse set of handwritten characters. Overall, the project contributes valuable insights toward building more accurate, efficient, and adaptable handwriting recognition systems.

References

Ahlawat, S., & Choudhary, A. (2020). Hybrid CNN-SVM Classifier for Handwritten Digit Recognition. *Procedia Computer Science, 167*, 2554-2560. doi:10.1016/j.procs.2020.03.309

AIITK, A. I. (2023). Handwritten Character Recognition. *Journal of Computer Vision and Pattern Recognition, 15*(2), 112-125.

AlKendi, W. G. (2024). Advancements and Challenges in Handwritten Text Recognition: A Comprehensive Survey. *Journal of Imaging, 10*(1), 18. doi:https://doi.org/10.3390/jimaging10010018

AlKendi, W., Gechter, F., Heyberger, L., & Guyeux, C. (2024). Advancements and Challenges in Handwritten Text Recognition: A Comprehensive Survey. *Journal of Imaging, 10*(1), 18. doi:https://doi.org/10.3390/jimaging10010018

AlKendi, W., Gechter, F., Heyberger, L., & Guyeux, C. (2024). Advancements and Challenges in Handwritten Text Recognition: A Comprehensive Survey. *Journal of Imaging, 10*(1), 18. doi:https://doi.org/10.3390/jimaging10010018

Alzubaidi, L., Zhang, J., Humaidi, A., Al-Dujaili, A., & Duan, Y. A.-S.-A. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data, 8*(1). doi:https://doi.org/10.1186/s40537-021-00444-8

Draelos, R. (2019). Measuring performance: The confusion matrix.

Fang, L. e. (2020). HandiText. *ACM/IMS Transactions on Data Science, 1*(4), 1-18. doi:https://doi.org/10.1145/3385189.

Fujitake, M. (2024). DTrOCR: Decoder-Only Transformer for Optical Character Recognition. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (pp. 8025-8035). Computer Vision Foundation.

Gülmez, B. (2022). A novel deep neural network model based Xception and genetic algorithm for detection of COVID-19 from X-ray images. *Annals of Operations Research*. doi:https://doi.org/10.1007/s10479-022-05151-y

Hossin, M., & Sulaiman, M. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining and Knowledge Management Process, 5*(2), 01-11. doi:10.5121/ijdkp.2015.5201

Huang, G. Z. (2022). Training RNN-T with CTC Loss in Automatic Speech Recognition. *Journal of Applied Mathematics and Computation, 6*(2), 256–262. doi:https://doi.org/10.26855/jamc.2022.06.010

J, P., v, S., & S, H. (2011). Neural Network based Handwritten Character Recognition system without feature extraction.

Jaiswal, K. A. (2023). Preprocessing Low Quality Handwritten Documents for OCR Models. *International Journal for Research in Applied Science and Engineering Technology, 11*(4), 2980–2985. doi:https://doi.org/10.22214/ijraset.2023.50664

José Manuel Álvarez-Alvarado, e. a. (2021). Hybrid Techniques to Predict Solar Radiation Using Support Vector Machine and Search Optimization Algorithms: A Review. *Applied Sciences, 11*(3), 1044–1044. doi:https://doi.org/10.3390/app11031044.

Kaggle. (n.d.). Retrieved from Handwriting Recognition.

Khan, S., & Nazir, S. (2022). Deep Learning Based Pashto Characters Recognition: LSTM-Based Handwritten Pashto characters recognition system. *Physical and Computational Sciences, 58*(3), 49–58. doi:https://doi.org/10.53560/PPASA(58-3)743

Li, M. T. (2021). TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models. doi:https://doi.org/10.48550/arxiv.2109.10282

Maimon, O. (2005). Introduction to Knowledge Discovery in Databases. *Data Mining and Knowledge Discovery Handbook*.

Mandal, G. S. (2019). A Novel Approach of Normalization for Online Handwritten Characters Using Geometrical Parameters on a 2D Plane. *International Journal of Computational Intelligence & IoT, 2*(2), 45-49.

Marti, U.-V., & H., B. (2002). The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition, 5*(1), 39-46. doi:10.1007/s100320200071

Md Rafiqul, I., Rashedul, I., & Kamrul Hasan, T. (2020). An efficient method for extraction and recognition of bangla characters from vehicle license plates. *Multimedia Tools and Applications, 79*(27-28), 20107-20132. doi:10.1007/s11042-020-08629-8

Mienye, I., Swart, T., & Obaido, G. (2024). Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications. *Information, 15*(9), 517. doi:https://doi.org/10.3390/info15090517

Monica, P., & Shital P., T. (2015). Handwritten Character Recognition in English: A Survey. *IJARCCE*, 345-350. doi:10.17148/ijarcce.2015.4278

Nanehkaran, Y. A. (2021). A pragmatic convolutional bagging ensemble learning for recognition of Farsi handwritten digits. *The Journal of Supercomputing, 77*(11), 13474–13493. doi:https://doi.org/10.1007/s11227-021-03822-4

Parthiban, R. E. (2020). Optical Character Recognition for English Handwritten Text Using Recurrent Neural Network. *2020 International Conference on System, Computation, Automation and Networking (ICSCAN)*. doi:https://doi.org/10.1109/icscan49426.2020.9262379

Saqib, N. H. (2022). Convolutional-Neural-Network-Based Handwritten Character Recognition: An Approach with Massive Multisource Data. *Algorithms, 15*(4), 129. doi:https://doi.org/10.3390/a15040129

Saqib, N., Haque, K., Yanambaka, V., & Abdelgawad, A. (2022). Convolutional-Neural-Network-Based Handwritten Character Recognition. *An Approach with Massive Multisource Data. Algorithms*(15), 129. doi: https://doi.org/10.3390/a15040129

Singh, H., Sharma, R., Singh, V., & Kumar, M. (2021). Recognition of online handwritten Gurmukhi characters using recurrent neural network classifier. *Soft Computing, 25*(8), 6329–6338. doi:https://doi.org/10.1007/s00500-021-05620-9

Vijay, P., & Y, S. (2012). A study on structural method of feature extraction for Handwritten Character Recognition. *Indian Journal of Science and Technology, 6*(S3), 174-178.

Vinjit, B. M., Bhojak, M. K., Kumar, S., & Chalak, G. (2020). A Review on Handwritten Character Recognition Methods and Techniques. *2020 International Conference on Communication and Signal Processing (ICCSP).* Chennai, India: IEEE. doi:10.1109/ICCSP48568.2020.9182129

Yang, W. e. (2016). DropSample : A new training method to enhance deep convolutional neural networks for large-scale unconstrained handwritten Chinese character recognition. *Pattern Recognition, 58*, 190-203. doi:https://doi.org/10.1016/j.patcog.2016.04.007.

Yu, Y. S. (2019). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation, 31*(7), 1235–1270. doi:https://doi.org/10.1162/neco_a_01199