

Handwritten Text Recognition using Deep Learning with CNN-RNN Architecture

A CAPSTONE PROJECT REPORT

*Submitted in partial fulfillment of the
requirement for the award of the
Degree of*

**BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING**

by
**GANTA TARUN (20BCD7251)
K SAI DEEPAK (20BCD7092)
DACHEPALLY VARUN (20BCE7092)
GHALI TRISHA (20BCI7206)**

Under the Guidance of

DR. NIHAR RANJAN PRADHAN



**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237**

DECEMBER 2023

CERTIFICATE

This is to certify that the Capstone Project work titled “**Handwritten Text Recognition using Deep Learning with CNN-RNN Architecture**” that is being submitted by **GANTA TARUN (20BCD7251), K SAI DEEPAK (20BCD7092), DACHEPALLY VARUN (20BCE7092), GHALI TRISHA (20BCI7206)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. NIHAR RANJAN PRADHAN

Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner 1

Internal Examiner 2

Approved by

HoD, Name of the Department
School of Computer Science and Engineering

ACKNOWLEDGEMENT

We wish to convey our sincere appreciation to VIT-AP University for its exceptional leadership, continuous oversight, and provision of crucial information, all of which were indispensable for the successful completion of our project. Throughout this endeavor, the university's unwavering commitment to academic excellence and support has served as a fundamental pillar.

A special recognition is extended to **DR. NIHAR RANJAN PRADHAN**, our esteemed project mentor, for his invaluable support and collaborative spirit. His expertise, guidance, and steadfast commitment have significantly influenced the positive outcomes of our project.

Our thanks also encompass the entire faculty of our institution for their timely encouragement and dedicated passion. Their mentorship has played a crucial role in cultivating an environment conducive to intellectual growth, enabling us to navigate our course of study successfully.

In conclusion, we express our gratitude to all those who have contributed, whether directly or indirectly, to the realization of this project. The collaborative efforts within the academic community and the support from various quarters have been essential in reaching this significant milestone.

ABSTRACT

In this paper an efficient system presents an Optical Character Recognition (OCR) system designed for recognizing handwritten text in images. The model architecture consists of a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), specifically Bidirectional Long Short-Term Memory (LSTM) layers. The model is trained using the Connectionist Temporal Classification (CTC) loss to align predicted character sequences with ground truth labels. The training process involves data loading, preprocessing, and the creation of TensorFlow datasets. The code also includes visualization functions for displaying images and their associated labels during training and evaluation.

Key features of the code include data augmentation, custom CTC loss layer, and decoding functions for interpreting model predictions. Additionally, a similarity matching function is provided to find the most similar image in a reference folder using Structural Similarity Index (SSIM). The model's architecture can be extended by incorporating attention mechanisms, transfer learning, ensemble models, and other enhancements to improve recognition accuracy. The abstract concludes by highlighting the importance of iterative experimentation to fine-tune the OCR system for specific applications.

TABLE OF CONTENTS

S.No.	Chapter	Title	Page Number
1.		Abstract	3
2.		List of Figures and Table	5
3.	1	Introduction	7
	1.1	Objectives	7
	1.2	Background and Literature Survey	9
	1.3	Organization of the Report	13
4.	2	Handwritten Text Recognition using Deep Learning	14
	2.1	Proposed System	14
	2.2	Working Methodology	15
	2.3	System Details	15
	2.4	Software Details	15
5.	3	Results and Discussion	27
6.	4	Conclusion & Future Works	29
7.	5	References	31
8.	6	Appendix	33

List of Tables

Table No.	Title	Page No.
1.	Literature Survey	10

List of Figures

Figure No.	Title	Page No.
1	System Block Diagram	14
2	Dataset size	16
3	Figure of Jupyter Notebook	17
4	Dataset head	17
5	Categorical distribution	18
6	2D categorical distributions	19
7	Unique characters in the dataset	19
8	Maximum length of a label	19
9	Head of the train dataset	20
10	categorical distribution of the train dataset	20
11	2D categorical distribution of the train dataset	21
12	size of each division of data	21
13	Figure of the train dataset	22

14	Figure of the test dataset	22
15	Figure of the validation dataset	22
16	Plot of the model	24
17	Example 1 of the test image	27
18	Example 2 of the test image	28
19	Example 3 of the test image	28
20	Deployed output	28

CHAPTER 1

INTRODUCTION

Handwritten Text Recognition (HTR) serves as a critical component in various applications, ranging from document digitization to automation of data entry tasks. This code presents an implementation of an Optical Character Recognition (OCR) system, a specialized form of HTR, leveraging deep learning techniques.

The model architecture is based on a combination of Convolutional Neural Networks (CNNs) for image feature extraction and Recurrent Neural Networks (RNNs), specifically Bidirectional Long Short-Term Memory (LSTM) layers, for sequence modeling. This hybrid CNN-RNN architecture proves effective in handling variable-length sequences inherent in handwritten text.

The training process involves the use of the Connectionist Temporal Classification (CTC) loss, enabling the model to learn to align predicted character sequences with ground truth labels. The code incorporates data augmentation strategies, visualization functions, and a custom CTC loss layer to enhance the training pipeline.

Beyond the core architecture, the code introduces a similarity matching function based on Structural Similarity Index (SSIM), allowing for the identification of the most similar image in a reference folder. Furthermore, the model's extensibility is emphasized, suggesting potential enhancements such as attention mechanisms, transfer learning, and ensemble models to boost recognition accuracy.

1.1 Objectives

1. Implement OCR Architecture:

- Develop a robust Optical Character Recognition (OCR) system using a hybrid architecture comprising Convolutional Neural Networks (CNNs) for image feature extraction and Bidirectional Long Short-Term Memory (LSTM) layers for sequence modeling.

2. Connectionist Temporal Classification (CTC) Training:

- Train the OCR model using the Connectionist Temporal Classification (CTC) loss to effectively align predicted character sequences with ground truth labels, enabling the model to learn from variable-length sequences inherent in handwritten text.

3. Data Loading and Preprocessing:

- Load and preprocess training, validation, and test datasets, incorporating strategies for data augmentation to enhance the diversity of training samples.

4. Visualization Functions:

- Develop functions for visualizing training and evaluation data, enabling insights into the model's performance and the recognition of handwritten text.

5. Similar Image Matching:

- Implement a function to find the most similar image in a reference folder using Structural Similarity Index (SSIM), providing a mechanism for assessing the model's performance against known images.

6. Extensibility and Modularity:

- Design the codebase with modularity in mind, allowing for easy extension and experimentation with additional components or enhancements such as attention mechanisms, transfer learning, and ensemble models.

7. Custom Loss Layer:

- Implement a custom CTC loss layer to facilitate efficient training of the model and to handle the intricacies of recognizing variable-length character sequences.

8. Inference Model Creation:

- Develop an inference model for making predictions on new images, allowing for real-world application of the trained OCR model.

9. Exploration of Enhancements:

- Encourage experimentation with various enhancements, such as attention mechanisms and transfer learning, to improve recognition accuracy and the model's adaptability to different scenarios.

1.2 Background and Literature Survey

Handwritten Text Recognition (HTR) has been a longstanding challenge in the field of computer vision and pattern recognition, with applications spanning document digitization, archival preservation, and automated data entry. The advent of deep learning techniques has significantly advanced the capabilities of HTR systems, enabling more accurate and robust recognition of handwritten text in various contexts.

Traditional HTR systems often relied on feature engineering and rule-based methods, which were limited in their ability to handle variations in handwriting styles and the inherent complexity of handwritten characters. The emergence of deep learning, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), has revolutionized HTR by allowing models to automatically learn hierarchical features and capture temporal dependencies within sequences.

The integration of CNNs in HTR architectures facilitates the extraction of relevant features from input images, enabling the model to discern intricate patterns and variations in handwritten text. Concurrently, Recurrent Neural Networks, specifically Bidirectional Long Short-Term Memory (LSTM) layers, are well-suited for modeling sequential data and handling variable-length character sequences, making them ideal for the diverse nature of handwritten text.

The Connectionist Temporal Classification (CTC) loss has proven to be instrumental in training deep learning models for sequence labeling tasks, such as HTR. CTC enables end-to-end training without the need for explicit alignment between input images and ground truth labels, aligning the predicted character sequences with the true labels during training.

While deep learning models have demonstrated remarkable success in HTR, the literature suggests ongoing exploration and refinement. Recent research has focused on enhancing model interpretability, addressing challenges posed by noisy or degraded handwriting, and extending HTR systems to handle multilingual and cursive scripts.

Furthermore, the flexibility of deep learning architectures allows for the incorporation of additional components such as attention mechanisms, transfer learning, and ensemble models. Attention mechanisms, inspired by their success in machine translation tasks, can improve the model's ability to focus on relevant image regions during recognition. Transfer learning leverages pre-trained models to enhance feature extraction, particularly useful when dealing with limited training data. Ensemble models, combining predictions from multiple models, offer potential gains in accuracy and robustness.

In summary, the integration of deep learning techniques, particularly CNNs and Bidirectional LSTMs, has propelled HTR systems to unprecedented levels of accuracy and adaptability. Ongoing research endeavors aim to address remaining challenges and explore avenues for further improvement, making HTR a dynamic and evolving field with promising applications in diverse domains.

Table 1 showing the papers referenced:

Paper Title	Authors	Observation	Remarks
“An improved faster-RCNN model for handwritten character recognition” [1]	Saleh Albahli, Marriam Nawaz, Ali Javed & Aun Irtaza (2015)	Improved accuracy with noisy handwriting	Novel use of data augmentation for noise handling
“STARN: Spatial-Temporal Attention-aware Residual Network for Scene Text Recognition” [2]	Yaqiang Wu, Yaoxiong Huang, Wanli Ouyang, Jun Yu, Zhiyong Cheng (2021)	Demonstrated improved performance in scene text recognition through a novel spatial-temporal attention-aware residual network.	Introduces STARN, a spatial-temporal attention-aware residual network for scene text recognition.
“Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes” [3]	Pengyuan Lyu, Minghui Liao, Cong Yao, Wenhao Wu, Xiang Bai (2018)	Focuses on spotting text with arbitrary shapes using an end-to-end trainable neural network.	Addressed the challenge of spotting text with irregular shapes, contributing to improved performance in text detection.

“EAST: An Efficient and Accurate Scene Text Detector” [4]	Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, Jiajun Liang (2017)	Showcased an efficient and accurate text detection model specifically designed for scenes with varying orientations.	Proposes an efficient and accurate scene text detection method called EAST.
“Deep TextSpotter: An End-to-End Trainable Scene Text Localization and Recognition Framework” [5]	Minghui Liao, Baoguang Shi, Xiang Bai. (2018)	Introduced a comprehensive model capable of localizing and recognizing scene text in a single end-to-end trainable framework.	Presents an end-to-end trainable framework for scene text localization and recognition.
“Scene Text Detection and Segmentation Based on Cascaded Convolution Neural Networks” [6]	Yuan Yan Tang, Xiangqian Wu (2021)	Multi-language recognition using ensemble of decision trees	Adapting Random Forest for multilingual tasks
“LayoutLMv2: Multi-modal Pretraining for Visual Document Understanding” [7]	Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, Ming Zhou (2020)	Demonstrated significant advancements in visual document understanding by leveraging multi-modal pretraining methods.	Introduces LayoutLMv2, a multi-modal pretraining approach for visual document understanding.

“Handwriting Detection and Recognition Improvements Based on Hidden Markov Model and Deep Learning” [8]	M. H. Alkawaz, Cheng Chun Seong, Husniza Razalli. (2020)	Based on the Kohonen Network	Increases recognition accuracy by 31% for bigger sized pixels
“Detection of Forged Handwriting Through Analyzation of Handwritten Characters Using Support Vector Machine” [9]	Ma. Crisanta Q. Jasmin, Mark Jayson F. Dela Cruz, A. Yumang (2022)	Uses the Support Vector Machine	Achieved an F1 score of 0.9 for forged handwriting detection
“Emotional States Detection Model from Handwriting by using Machine Learning” [10]	Khadija Nadeem, Mudassar Ahmad, Muhammad Asif Habib (2022)	Utilizes feature extraction to classify an emotion from the detected handwriting	Had the highest accuracy up to date for the Emothaw dataset for the 3 main emotions
“Transfer Learning to Detect Age From Handwriting” [11]	Najla AL-Qawasmeh, C. Suen (2022)	Employs ResNet and GoogleNet CNN	Obtained an accuracy of 69.7% and 61.1%
“Transcript Anatomization with Multi-Linguistic and Speech Synthesis Features” [12]	Rohan Modi (2021)	Applies CRNN and Hidden Markov Model	Effectiveness of Optical Character Recognition with speech synthesis

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, the methodology applied in this model as well as the software details.
- Chapter 3 discusses the results obtained after the project was implemented.
- Chapter 4 concludes the report and provides several future scopes.
- Chapter 5 consists of references.

CHAPTER 2

Handwritten Text Recognition using Deep Learning

This Chapter describes the proposed system, working methodology and software details.

2.1 Proposed System

The following block diagram shows the system architecture of this project.

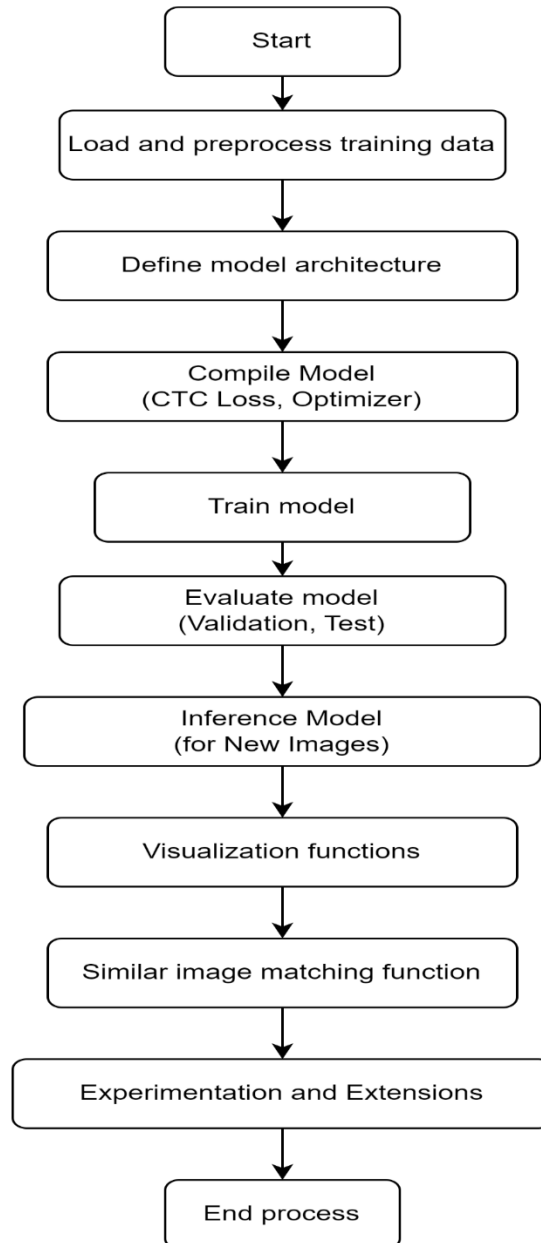


Figure 1: System Block Diagram

2.2 Working Methodology

The system is based only on software.

The software used to implement this software was Jupyter Notebook and Visual Studio Code. Jupyter Notebook is an open-source interactive web programme where you can share live codes, equations and even visualizations. There are multiple libraries that are integrated into Jupyter such as Pandas, Numpy, Tensorflow, Seaborn, Matplotlib, and Plotly for visualizations.

Visual Studio Code is a light weight open-source code editor where the deployment of our model was done.

The data which consisted of images was taken from Kaggle which was then split into training, validation, and test datasets respectively which ensures the robustness and reliability of the deep learning model.

2.3 System Details

This section describes the software details of the system:

2.4 Software Details

Kaggle and Jupyter Notebook and Visual studio code is used.

i) Kaggle

Kaggle stands as a pivotal platform in the realm of data science and machine learning, offering a dynamic environment for enthusiasts and professionals alike. Renowned for hosting data science competitions that tackle real-world challenges, Kaggle attracts a global community of participants aiming to develop cutting-edge models. Beyond competitions, Kaggle provides an extensive repository of datasets spanning diverse domains and supports collaborative projects through its integration of Jupyter Notebooks. The platform's kernels enable users to create, share, and iterate on code seamlessly, fostering a culture of collaboration and knowledge exchange. Kaggle's vibrant community engages in forums, discussions, and collaborative projects, enhancing the learning experience for individuals of varying skill levels. Additionally, Kaggle offers

educational resources, courses, and learning paths, making it an invaluable hub for honing data science and machine learning skills. With its cloud-based computing environment, Kaggle empowers users to experiment with GPUs and TPUs without the need for high-end hardware. Overall, Kaggle plays a central role in driving innovation, fostering collaboration, and providing continuous learning opportunities in the ever-evolving field of data science.

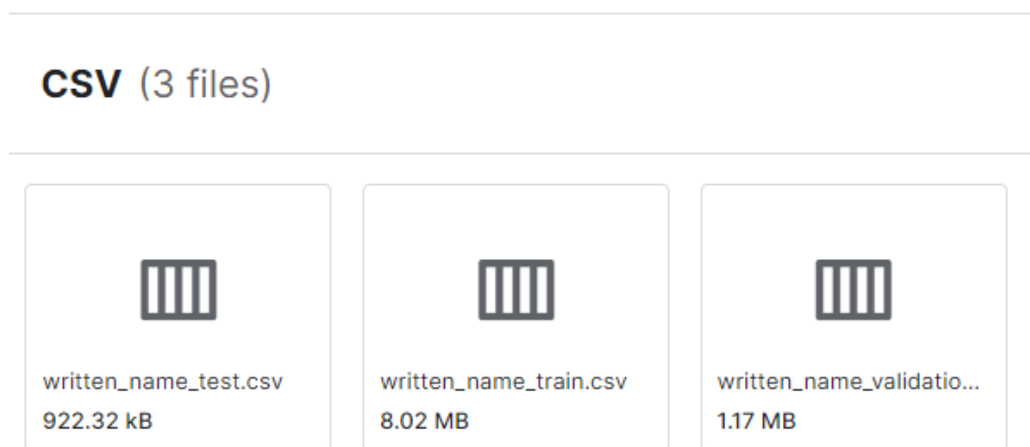


Figure 2: Dataset size

ii) JUPYTER NOTEBOOK

Jupyter Notebook stands out as a versatile and interactive web application that has become integral to the toolkit of data scientists and researchers. Functioning as an open-source platform, Jupyter Notebook enables users to seamlessly integrate live code, equations, visualizations, and narrative text within a single, shareable document. Its support for multiple programming languages, including Python, R, Julia, and Scala, adds to its adaptability, making it a preferred choice across diverse fields.

Jupyter's distinctive feature lies in its organization of work into notebooks, composed of cells that can contain code snippets, Markdown-formatted text, equations, and visual outputs. This facilitates an iterative and collaborative approach to coding and analysis. With a plethora of libraries integrated, such as Pandas, Numpy, Seaborn, Matplotlib, and Plotly, Jupyter Notebook provides a comprehensive environment for data manipulation, analysis, and visualization. Whether used for educational purposes, research, or data exploration, Jupyter Notebook stands as a powerful and user-friendly tool, fostering transparency, reproducibility, and interactive computing experiences.

iii) VISUAL STUDIO CODE

Visual Studio Code, commonly known as VS Code, stands as a versatile and free-source code editor designed by Microsoft for developers across Windows, macOS, and Linux platforms. This editor has gained widespread adoption owing to its rich feature set, including IntelliSense for smart code completion, syntax highlighting, and navigation aids. Its strength lies in its extensibility through a vast library of extensions that cater to diverse programming languages, frameworks, and tools. Offering an integrated terminal, built-in debugger, Git integration, and customizable interface, VS Code empowers developers to efficiently write, debug, and manage their code while providing a seamless and personalized coding experience.

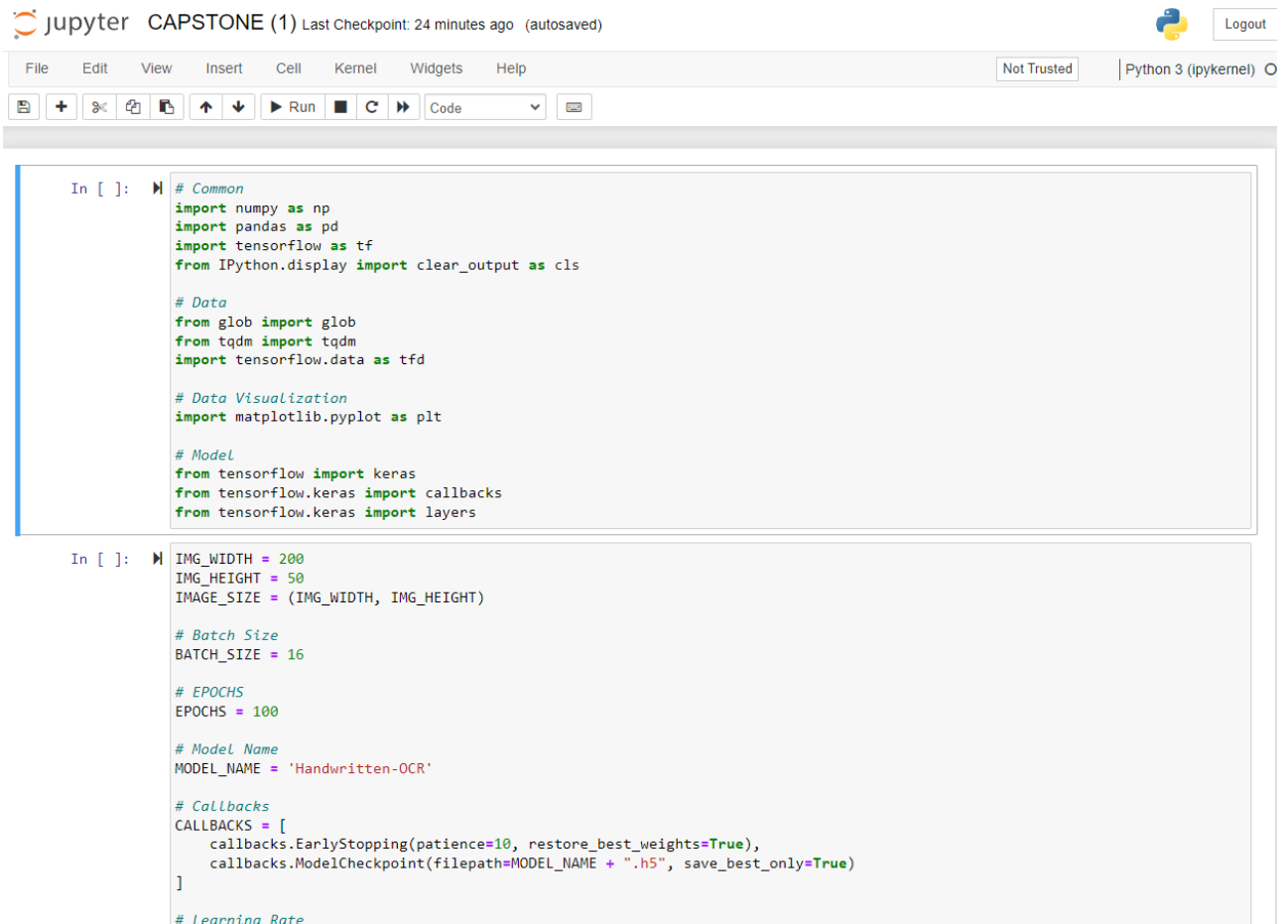


Figure 3: Figure of Jupyter Notebook

	FILENAME	IDENTITY
0	TRAIN_00001.jpg	BALTHAZAR
1	TRAIN_00002.jpg	SIMON
2	TRAIN_00003.jpg	BENES
3	TRAIN_00004.jpg	LA LOVE
4	TRAIN_00005.jpg	DAPHNE

Figure 4: Dataset head

Categorical distributions

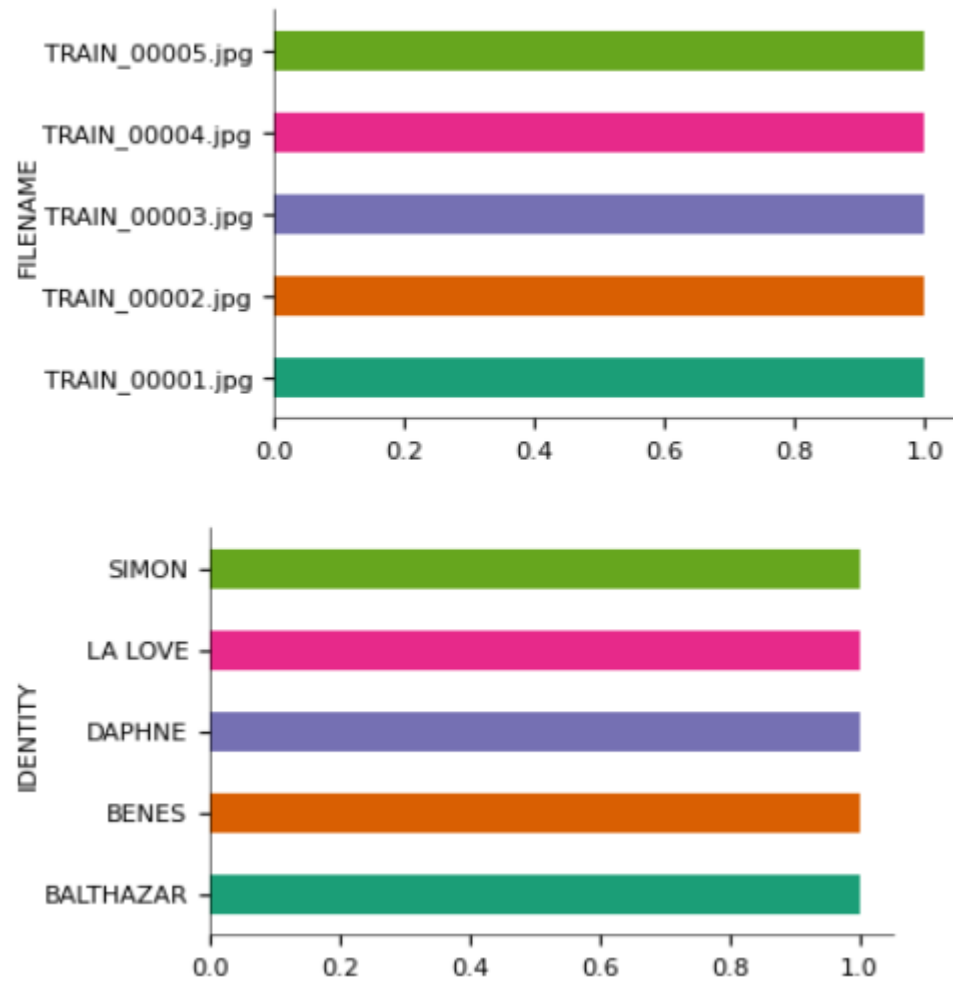


Figure 5: Categorical Distribution

2-d categorical distributions

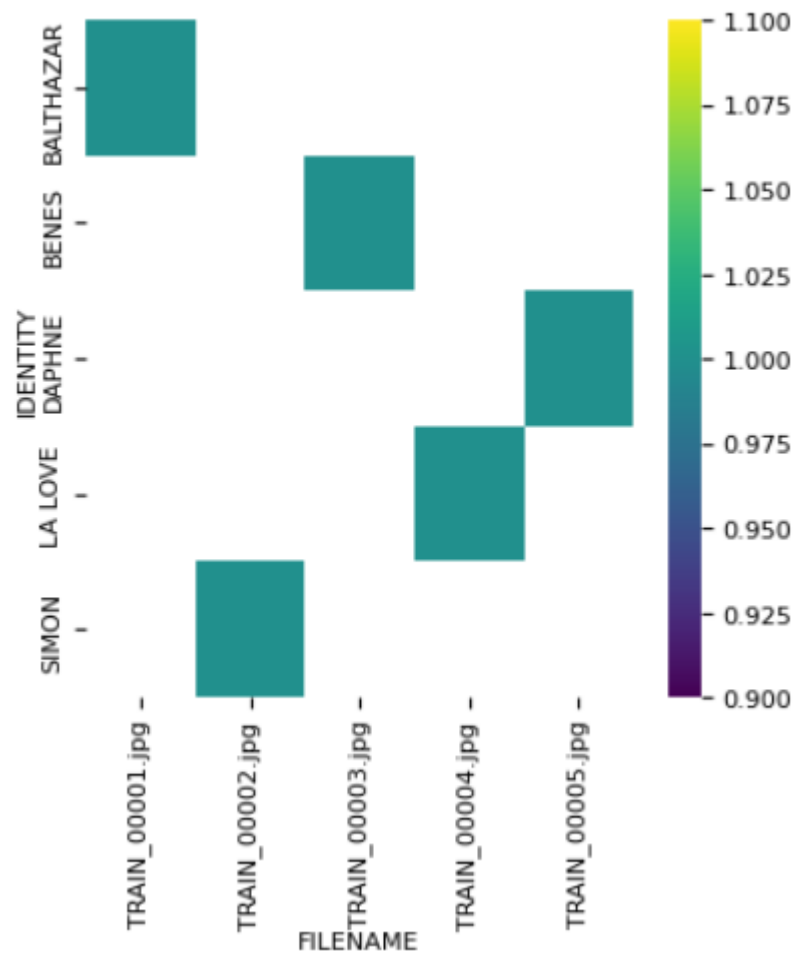


Figure 6: 2-D categorical distribution

```
Total number of unique characters : 28
Unique Characters :
{'C', 'F', 'R', 'Z', 'D', 'Y', 'B', 'K', 'A', 'L', 'H', 'V', 'G', 'M', 'U',
'T', 'W', 'O', 'J', ' ', 'S', 'N', 'X', 'Q', 'E', 'P', 'I', '-'}
```

Figure 7: Unique characters in the dataset

Maximum length of a label : 15

Figure 8: maximum length of a label

	FILENAME	IDENTITY
0	/content/train/TRAIN_00001.jpg	BALTHAZAR
1	/content/train/TRAIN_00002.jpg	SIMON
2	/content/train/TRAIN_00003.jpg	BENES
3	/content/train/TRAIN_00004.jpg	LA LOVE
4	/content/train/TRAIN_00005.jpg	DAPHNE

☐
☐

Figure 9: head of the train dataset

Categorical distributions

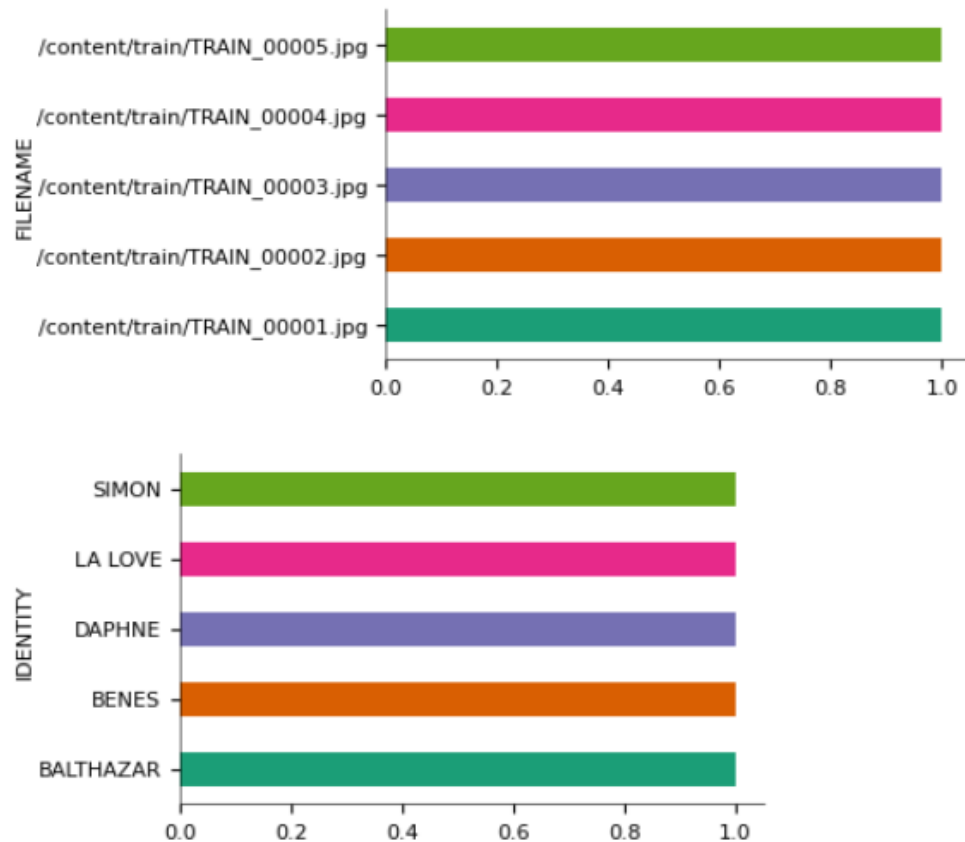


Figure 10: categorical distribution of the train dataset

2-d categorical distributions

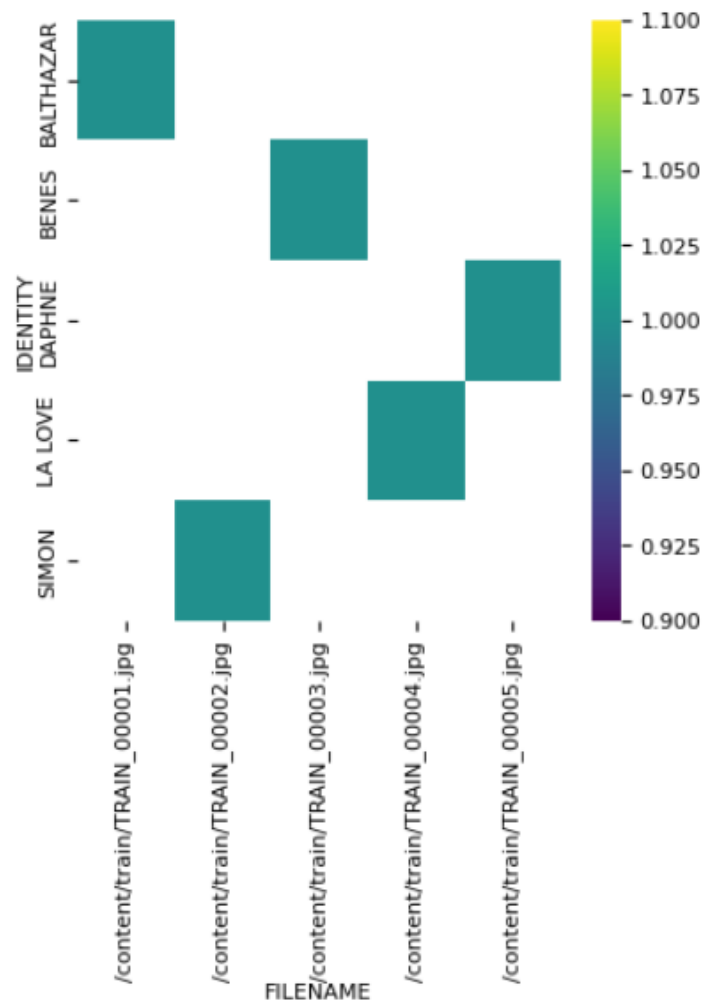


Figure 11: 2D categorical distribution of the train dataset

```
Training Data Size    : 160
Validation Data Size  : 160
Testing Data Size     : 160
```

Figure 12: size of each division of data



Figure 13: Figure of the train dataset



Figure 14: Figure of the test dataset



Figure 15: Figure of the validation dataset

Summary of the OCR model:

Model: "model"

Layer (type) connected to	Output Shape	Param #	Connect
image (InputLayer)	[(None, 200, 50, 1)]	0	[]
conv2d (Conv2D)	(None, 200, 50, 32)	320	['image [0][0]']
max_pooling2d (MaxPooling2D)	(None, 100, 25, 32)	0	['conv2 d[0][0]']
conv2d_1 (Conv2D)	(None, 100, 25, 64)	18496	['max_p ooling2d[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 50, 12, 64)	0	['conv2 d_1[0][0]']
reshape (Reshape)	(None, 50, 768)	0	['max_p ooling2d_1[0][0]']
dense (Dense)	(None, 50, 64)	49216	['resha pe[0][0]']
dropout (Dropout)	(None, 50, 64)	0	['dense [0][0]']
bidirectional (Bidirectional)	(None, 50, 256)	197632	['dropo ut[0][0]']
bidirectional_1 (Bidirectional)	(None, 50, 128)	164352	['bidir ectional[0][0]']
label (InputLayer)	[(None, None)]	0	[]
dense_1 (Dense)	(None, 50, 30)	3870	['bidir ectional_1[0][0]']
ctc_layer (CTCLayer)	(None, 50, 30)	0	['label [0][0]', 'dense _1[0][0]']
=====			
Total params: 433886 (1.66 MB)			
Trainable params: 433886 (1.66 MB)			
Non-trainable params: 0 (0.00 Byte)			

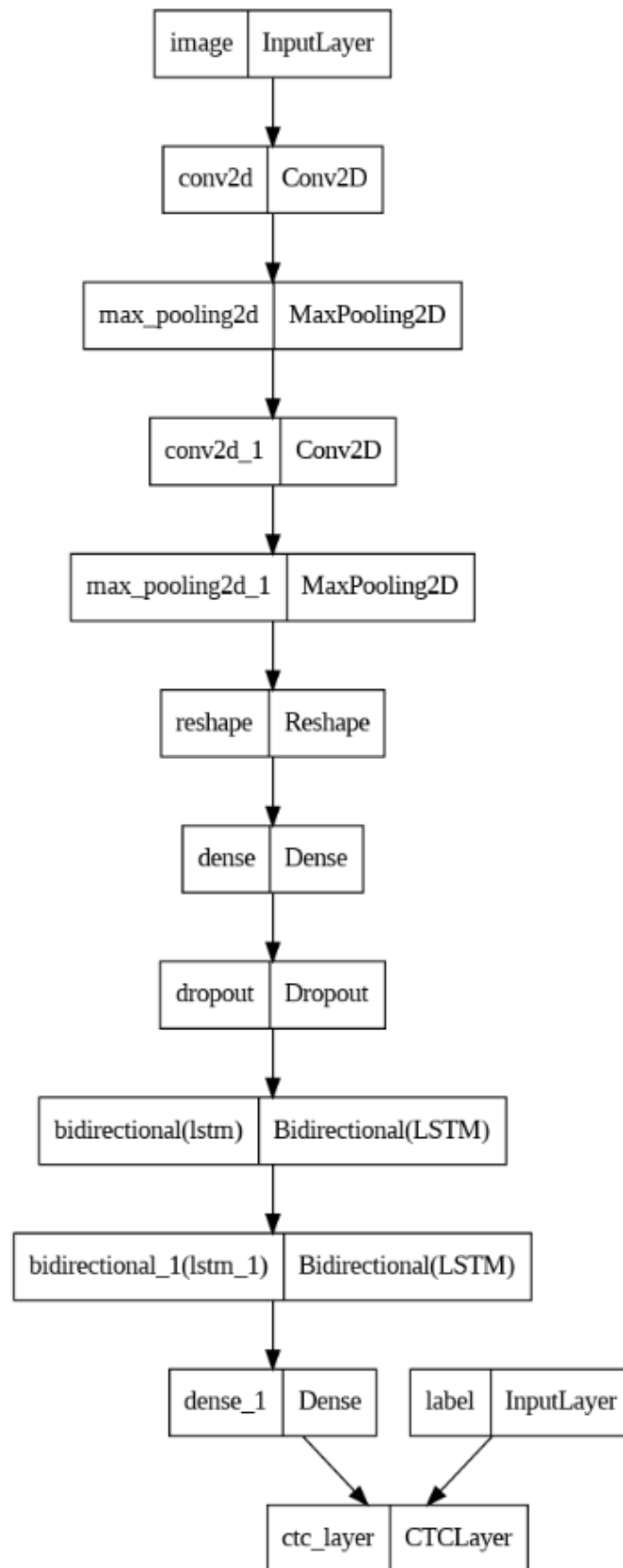


Figure 16: Plot of the model

Summary of the inference 1 model:

Model: "model_1"

Layer (type)	Output Shape	Param #
image (InputLayer)	[(None, 200, 50, 1)]	0
conv2d (Conv2D)	(None, 200, 50, 32)	320
max_pooling2d (MaxPooling2D)	(None, 100, 25, 32)	0
conv2d_1 (Conv2D)	(None, 100, 25, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 50, 12, 64)	0
reshape (Reshape)	(None, 50, 768)	0
dense (Dense)	(None, 50, 64)	49216
dropout (Dropout)	(None, 50, 64)	0
bidirectional (Bidirectional)	(None, 50, 256)	197632
bidirectional_1 (Bidirectional)	(None, 50, 128)	164352
dense_1 (Dense)	(None, 50, 30)	3870
Total params: 433886 (1.66 MB)		
Trainable params: 433886 (1.66 MB)		
Non-trainable params: 0 (0.00 Byte)		

Summary of the inference 2 model:

Model: "model_3"

Layer (type)	Output Shape	Param #
image (InputLayer)	[(None, 200, 50, 1)]	0
conv2d_2 (Conv2D)	(None, 200, 50, 32)	320
conv2d_3 (Conv2D)	(None, 200, 50, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 100, 25, 32)	0
conv2d_4 (Conv2D)	(None, 100, 25, 64)	18496

conv2d_5 (Conv2D)	(None, 100, 25, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 50, 12, 128)	0
reshape_1 (Reshape)	(None, 50, 1536)	0
dense_2 (Dense)	(None, 50, 64)	98368
dense_3 (Dense)	(None, 50, 128)	8320
dropout_1 (Dropout)	(None, 50, 128)	0
bidirectional_2 (Bidirectional)	(None, 50, 512)	788480
bidirectional_3 (Bidirectional)	(None, 50, 256)	656384
dense_4 (Dense)	(None, 50, 30)	7710

=====

Total params: 1661182 (6.34 MB)
Trainable params: 1661182 (6.34 MB)
Non-trainable params: 0 (0.00 Byte)

iii) Integration

The integration of Kaggle and Jupyter Notebook in this code exemplifies a synergistic approach to data science and machine learning development. Kaggle, as the dataset source, provides a centralized platform for hosting and sharing datasets, offering convenient access to diverse datasets for various tasks. The code, executed within Jupyter Notebook, capitalizes on the interactive and iterative nature of the notebook environment. Through Jupyter, the code unfolds in a sequential and comprehensible manner, combining Python scripts with Markdown cells for explanatory text. The dataset is read in Jupyter using the Pandas library. This integration allows for step-by-step exploration, code execution, and documentation within a single, cohesive environment. Leveraging Kaggle for dataset storage and retrieval and Jupyter Notebook for code development results in a seamless workflow, fostering clarity, collaboration, and effective model development.

CHAPTER 3

RESULTS AND DISCUSSIONS

a. SSIM Score

The Structural Similarity Index (SSIM) is a valuable metric in image processing that goes beyond simple pixel-wise comparisons, offering a more holistic assessment of image quality. It addresses challenges associated with traditional metrics by considering luminance, contrast, and structure. The SSIM metric produces a numerical score between -1 and 1, where 1 signifies perfect similarity. This makes it a robust tool for applications such as image compression, image restoration, and quality assessment, where a comprehensive understanding of image fidelity is crucial. The SSIM index's ability to account for both local and global structural information makes it particularly effective in scenarios where subtle visual details and spatial arrangements play a significant role.

In the provided code, SSIM is employed as a critical component for image comparison. The code utilizes SSIM to find the most similar image in a given folder by comparing the uploaded image with each reference image based on structural characteristics. By calculating the SSIM score, the algorithm assesses the similarity in luminance, contrast, and structure, enabling the identification of the most visually analogous image. This showcases the practical application of SSIM in real-world scenarios, where the metric's nuanced evaluation contributes to more accurate and meaningful image analysis.

The model was able to detect multiple uploaded images delivering an SSIM score of range from 0.0 to 1.0. When it identifies the writing on the image, it also provides the identity of said image.

Match: The uploaded image is similar to 'TEST_0001.jpg' with SSIM score 1.00
Identity: KEVIN

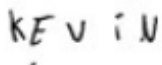


Figure 17 Example 1 of the test image

Match: The uploaded image is Matched to 'TEST_0006.jpg' with SSIM score 1.000
Identity: MARTIN

PRENOM: MARTIN

Figure 18 Example 2 of the test image

Match: The uploaded image is Matched to 'TEST_0090.jpg' with SSIM score 0.864
Identity: DE SA

DE SA

Figure 19 Example 3 of the test image

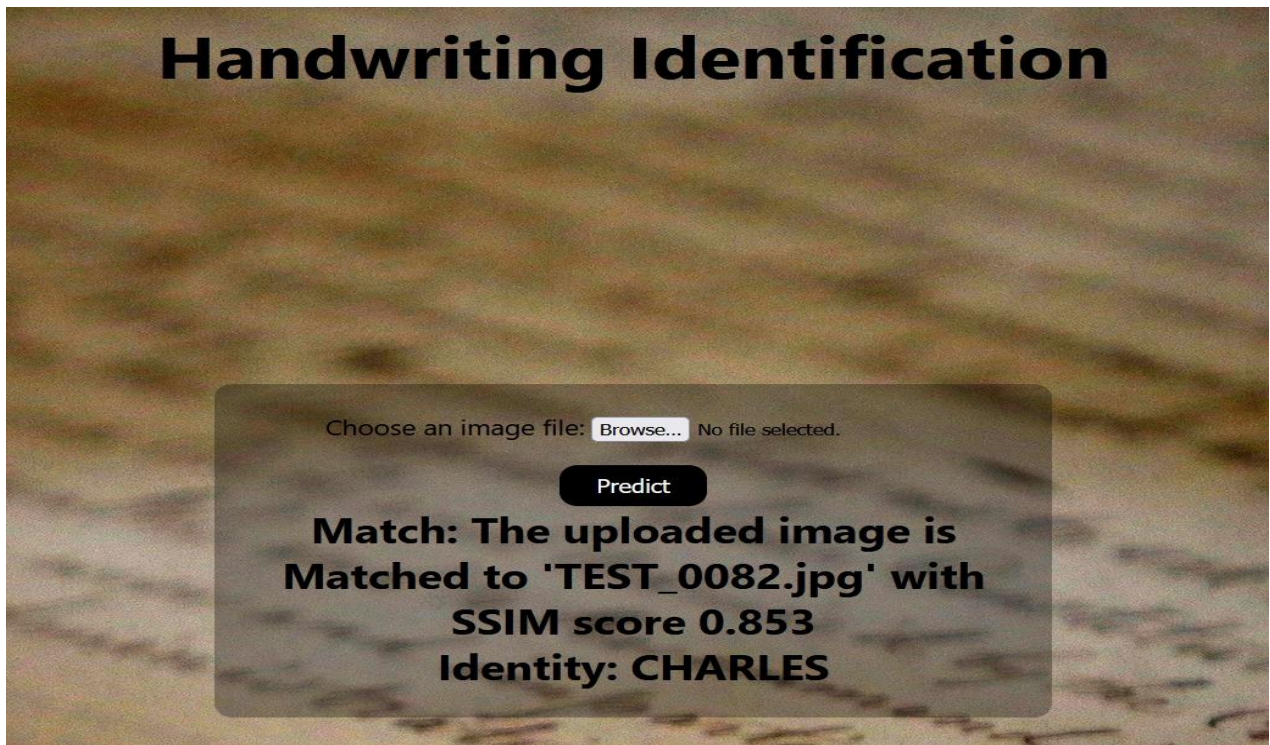


Figure 20 Deployed output

CHAPTER 4

CONCLUSION AND FUTURE WORK

The provided code implements a Handwritten OCR (Optical Character Recognition) system using deep learning techniques. The model is trained on a dataset obtained from Kaggle, with the implementation done in a Jupyter Notebook. The code involves data preprocessing, model architecture design, and training using TensorFlow and Keras. The trained model is then used for inference on test and validation datasets, with visualizations to assess its performance. The use of CTC (Connectionist Temporal Classification) loss and bidirectional LSTMs contributes to the model's ability to recognize handwritten text.

There are various future scopes for this project, such as:

1. **Enhanced Model Architectures:** Explore more complex neural network architectures and experiment with different combinations of layers to improve the model's accuracy and robustness.
2. **Data Augmentation:** Implement data augmentation techniques to artificially increase the diversity of the training dataset, which can enhance the model's generalization to different handwriting styles.
3. **Transfer Learning:** Investigate the applicability of transfer learning by using pre-trained models on larger text datasets. This can be particularly beneficial when dealing with limited annotated handwritten text data.
4. **Hyperparameter Tuning:** Conduct a thorough search for optimal hyperparameters, including learning rates, dropout rates, and the number of LSTM units, to fine-tune the model's performance.
5. **Deployment:** Consider deploying the OCR model to practical applications, such as mobile applications or web services, making it accessible for a wider audience.

6. Real-time Inference: Adapt the model for real-time inference, enabling it to process handwritten text from live input sources like cameras or stylus devices.
7. Support for Multiple Languages: Extend the OCR system to recognize and transcribe handwritten text in multiple languages, broadening its applicability.
8. Integration with Other Technologies: Explore integration with other technologies, such as natural language processing (NLP) models, to enable more advanced processing of recognized text, including language understanding and semantic analysis.

CHAPTER 5

REFERENCES

- [1] Albahli, S., Nawaz, M., Javed, A., & Irtaza, A. (2021). An improved faster-RCNN model for handwritten character recognition. *Arabian Journal for Science and Engineering*, 46, 8509 - 8523.
- [2] Liu, Wei & Chen, Chaofeng & Wong, Kwan-Yee Kenneth & Su, Zhizhong & Han, Junyu. (2016). STAR-Net: A SpaTial Attention Residue Network for Scene Text Recognition. 43.1-43.13. 10.5244/C.30.43.
- [3] Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary ShapesMinghui Liao, Pengyuan Lyu, Minghang He, Cong Yao, Wenhao Wu, Xiang Bai
<https://doi.org/10.48550/arXiv.1908.08207>
- [4] X. Zhou et al., "EAST: An Efficient and Accurate Scene Text Detector," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 2642-2651, doi: 10.1109/CVPR.2017.283.
- [5] M. Buřta, L. Neumann and J. Matas, "Deep TextSpotter: An End-to-End Trainable Scene Text Localization and Recognition Framework," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 2223-2231, doi: 10.1109/ICCV.2017.242.
- [6] Tang, Yuan & Wu, Xiangqian. (2017). Scene Text Detection and Segmentation Based on Cascaded Convolution Neural Networks. IEEE transactions on image processing : a publication of the IEEE Signal Processing Society. PP. 10.1109/TIP.2017.2656474.
- [7] LayoutLMv2: Multi-modal Pre-training for Visually-Rich Document Understanding
<https://doi.org/10.48550/arXiv.2012.14740>
- [8] M. H. Alkawaz, C. C. Seong and H. Razalli, "Handwriting Detection and Recognition Improvements Based on Hidden Markov Model and Deep Learning," 2020 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA), Langkawi, Malaysia, 2020, pp. 106-110, doi: 10.1109/CSPA48992.2020.9068682.

- [9]Jasmin, M., Cruz, M., & Yumang, A. (2022). Detection of Forged Handwriting Through Analyzation of Handwritten Characters Using Support Vector Machine. 2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAET), 1-5. <https://doi.org/10.1109/IICAET55139.2022.9936769>.
- [10] Nadeem, K., Ahmad, M., & Habib, M. (2022). Emotional States Detection Model from Handwriting by using Machine Learning. 2022 International Conference on Frontiers of Information Technology (FIT), 284-289. <https://doi.org/10.1109/FIT57066.2022.00059>.
- [11] AL-Qawasmeh, N., & Suen, C. (2022). Transfer Learning to Detect Age From Handwriting. Adv. Artif. Intell. Mach. Learn., 2. <https://doi.org/10.54364/aaiml.2022.1126>
- [12] Modi, R. (2021). Transcript Anatomization with Multi-Linguistic and Speech Synthesis Features. International Journal for Research in Applied Science and Engineering Technology. <https://doi.org/10.22214/ijraset.2021.35371>.
- [13] How to install Jupyter Notebook <https://noteable.io/jupyter-notebook/install-jupyter-notebook/>
- [14] dataset - <https://www.kaggle.com/datasets/ssarkar445/handwriting-recognitionocr/data>

CHAPTER 6

APPENDIX

Jupyter Notebook Code

```
# Common

import numpy as np
import pandas as pd
import tensorflow as tf
from IPython.display import clear_output as cls


# Data

from glob import glob
from tqdm import tqdm
import tensorflow.data as tfd


# Data Visualization

import matplotlib.pyplot as plt


# Model

from tensorflow import keras
from tensorflow.keras import callbacks
from tensorflow.keras import layers

IMG_WIDTH = 200
IMG_HEIGHT = 50
IMAGE_SIZE = (IMG_WIDTH, IMG_HEIGHT)


# Batch Size

BATCH_SIZE = 16
```

```

# EPOCHS
EPOCHS = 100

# Model Name
MODEL_NAME = 'Handwritten-OCR'

# Callbacks
CALLBACKS = [
    callbacks.EarlyStopping(patience=10, restore_best_weights=True),
    callbacks.ModelCheckpoint(filepath=MODEL_NAME + ".h5", save_best_only=True)
]

# Learning Rate
LEARNING_RATE = 1e-3

# Random Seed
np.random.seed(2569)
tf.random.set_seed(2569)

# File Paths
train_csv_path = '/content/written_name_train.csv'
valid_csv_path = '/content/written_name_validation.csv'
test_csv_path = '/content/written_name_test.csv'

train_image_dir = '/content/train'
valid_image_dir = '/content/validation'
test_image_dir = '/content/test'

# Data Size

```

```

TRAIN_SIZE = BATCH_SIZE * 1000
VALID_SIZE = BATCH_SIZE * 500
TEST_SIZE = BATCH_SIZE * 100

# AUTOTUNE
AUTOTUNE = tfd.AUTOTUNE

# Train CSV
train_csv = pd.read_csv("/content/written_name_train.csv")[:TRAIN_SIZE]

# Validation CSV
valid_csv = pd.read_csv("/content/written_name_validation.csv")[:VALID_SIZE]

# Test CSV
test_csv = pd.read_csv("/content/written_name_test.csv")[:TEST_SIZE]

train_csv.head()

# Get all train labels
train_labels = [str(word) for word in train_csv['IDENTITY'].to_numpy()]
train_labels[:10]

# Unique characters
unique_chars = set(char for word in train_labels for char in word)
n_classes = len(unique_chars)

# Show
print(f"Total number of unique characters : {n_classes}")
print(f"Unique Characters : \n{unique_chars}")

```

```

MAX_LABEL_LENGTH = max(map(len, train_labels))

print(f"Maximum length of a label : {MAX_LABEL_LENGTH}")

train_csv['FILENAME'] = [train_image_dir + f"/{filename}" for filename in
train_csv['FILENAME']]

valid_csv['FILENAME'] = [valid_image_dir + f"/{filename}" for filename in
valid_csv['FILENAME']]

test_csv['FILENAME'] = [test_image_dir + f"/{filename}" for filename in
test_csv['FILENAME']]

train_csv.head()

# Character to numeric value dictionary
char_to_num = layers.StringLookup(
    vocabulary = list(unique_chars),
    mask_token = None
)

# Reverse dictionary
num_to_char = layers.StringLookup(
    vocabulary = char_to_num.get_vocabulary(),
    mask_token = None,
    invert = True
)

def load_image(image_path : str):
    """
    This function loads and preprocesses images. It first receives the image path, which is used to
    decode the image as a JPEG using TensorFlow. Then, it converts the image to a tensor and
    applies
    two processing functions: resizing and normalization. The processed image is then returned by
    the function.

```

Argument :

image_path : The path of the image file to be loaded.

Return:

image : The loaded image as a tensor.

'''

Read the Image

image = tf.io.read_file(image_path)

Decode the image

decoded_image = tf.image.decode_jpeg(contents = image, channels = 1)

Convert image data type.

cnvt_image = tf.image.convert_image_dtype(image = decoded_image, dtype = tf.float32)

Resize the image

resized_image = tf.image.resize(images = cnvt_image, size = (IMG_HEIGHT, IMG_WIDTH))

Transpose

image = tf.transpose(resized_image, perm = [1, 0, 2])

Convert image to a tensor.

image = tf.cast(image, dtype = tf.float32)

Return loaded image

return image

```
def encode_single_sample(image_path : str, label : str):
```

```
'''
```

The function takes an image path and label as input and returns a dictionary containing the processed image tensor and the label tensor.

First, it loads the image using the load_image function, which decodes and resizes the image to a specific size. Then it converts the given

label string into a sequence of Unicode characters using the unicode_split function. Next, it uses the char_to_num layer to convert each

character in the label to a numerical representation. It pads the numerical representation with a special class (n_classes)

to ensure that all labels have the same length (MAX_LABEL_LENGTH). Finally, it returns a dictionary containing the processed image tensor

and the label tensor.

Arguments :

image_path : The location of the image file.

label : The text to present in the image.

Returns:

dict : A dictionary containing the processed image and label.

```
'''
```

```
# Get the image
```

```
image = load_image(image_path)
```

```
# Convert the label into characters
```

```
chars = tf.strings.unicode_split(label, input_encoding='UTF-8')
```

```
# Convert the characters into vectors
```



```

vecs = char_to_num(chars)

# Pad label
pad_size = MAX_LABEL_LENGTH - tf.shape(vecs)[0]
vecs = tf.pad(vecs, paddings = [[0, pad_size]], constant_values=n_classes+1)

return {'image':image, 'label':vecs}

# Training Data
train_ds = tf.data.Dataset.from_tensor_slices(
    (np.array(train_csv['FILENAME'].to_list()), np.array(train_csv['IDENTITY'].to_list())))
    ).shuffle(1000).map(encode_single_sample,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

# Validation data
valid_ds = tf.data.Dataset.from_tensor_slices(
    (np.array(valid_csv['FILENAME'].to_list()), np.array(valid_csv['IDENTITY'].to_list())))
    ).map(encode_single_sample,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

# Testing data.
test_ds = tf.data.Dataset.from_tensor_slices(
    (np.array(test_csv['FILENAME'].to_list()), np.array(test_csv['IDENTITY'].to_list())))
    ).map(encode_single_sample,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

print(f"Training Data Size : {tf.data.Dataset.cardinality(train_ds).numpy() * BATCH_SIZE}")
print(f"Validation Data Size : {tf.data.Dataset.cardinality(valid_ds).numpy() * BATCH_SIZE}")
print(f"Testing Data Size : {tf.data.Dataset.cardinality(test_ds).numpy() * BATCH_SIZE}")

"pip install matplotlib tensorflow"

```

```

def show_images(data, GRID=[4,4], FIGSIZE=(25, 8), cmap='binary_r', model=None,
decode_pred=None):

    # Plotting configurations
    plt.figure(figsize=FIGSIZE)
    n_rows, n_cols = GRID

    # Loading Data
    data = next(iter(data))
    images, labels = data['image'], data['label']

    # Iterate over the data
    for index, (image, label) in enumerate(zip(images, labels)):

        # Label processing
        text_label = num_to_char(label)
        text_label = tf.strings.reduce_join(text_label).numpy().decode('UTF-8')
        text_label = text_label.replace("[UNK]", " ").strip()

        # Create a sub plot
        plt.subplot(n_rows, n_cols, index+1)
        plt.imshow(tf.transpose(image, perm=[1,0,2]), cmap=cmap)
        plt.axis('off')

        if model is not None and decode_pred is not None:
            # Make prediction
            pred = model.predict(tf.expand_dims(image, axis=0))
            pred = decode_pred(pred)[0]
            title = f"True : {text_label}\nPred : {pred}"
            plt.title(title)

```

```

else:
    # add title
    plt.title(text_label)

# Show the final plot
cls()
plt.show()
plt.pause(0.1) # Add a small delay before closing the plot

plt.show()
show_images(data=train_ds, cmap='gray')
class CTCLayer(layers.Layer):

    def __init__(self, **kwargs) -> None:
        super().__init__(**kwargs)

        self.loss_fn = keras.backend.ctc_batch_cost

    def call(self, y_true, y_pred):

        batch_len = tf.cast(tf.shape(y_true)[0], dtype='int64')

        input_len = tf.cast(tf.shape(y_pred)[1], dtype='int64') * tf.ones(shape=(batch_len, 1),
dtype='int64')

        label_len = tf.cast(tf.shape(y_true)[1], dtype='int64') * tf.ones(shape=(batch_len, 1),
dtype='int64')

        loss = self.loss_fn(y_true, y_pred, input_len, label_len)

        self.add_loss(loss)

```

```

        return y_pred

# Input Layer
input_images = layers.Input(shape=(IMG_WIDTH, IMG_HEIGHT, 1), name="image")

# Labels : These are added for the training purpose.
target_labels = layers.Input(shape=(None, ), name="label")

# CNN Network
x = layers.Conv2D(
    filters=32,
    kernel_size=3,
    strides=1,
    padding='same',
    activation='relu',
    kernel_initializer='he_normal'
)(input_images)

x = layers.MaxPool2D(pool_size=(2,2), strides=(2,2))(x)

x = layers.Conv2D(
    filters=64,
    kernel_size=3,
    strides=1,
    padding='same',
    activation='relu',
    kernel_initializer='he_normal'
)(x)

x = layers.MaxPool2D(pool_size=(2,2), strides=(2,2))(x)

```

```

# Encoding Space
encoding = layers.Reshape(target_shape=((IMG_WIDTH//4), (IMG_HEIGHT//4)*64))(x)
encoding = layers.Dense(64, activation='relu', kernel_initializer='he_normal')(encoding)
encoding = layers.Dropout(0.2)(encoding)

# RNN Network
x = layers.Bidirectional(layers.LSTM(128, return_sequences=True, dropout=0.25))(encoding)
x = layers.Bidirectional(layers.LSTM(64, return_sequences=True, dropout=0.25))(x)

# Output Layer
output = layers.Dense(len(char_to_num.get_vocabulary())+1, activation='softmax')(x)

# CTC Layer
ctc_layer = CTCLayer()(target_labels, output)

# Model
ocr_model = keras.Model(
    inputs=[input_images, target_labels],
    outputs=[ctc_layer]
)
ocr_model.summary()
tf.keras.utils.plot_model(ocr_model)
def decode_pred(pred_label):

```

'''

The `decode_pred` function is used to decode the predicted labels generated by the OCR model. It takes a matrix of predicted labels as input, where each time step represents the probability for each character. The function uses CTC decoding to decode the numeric labels back into their

character values. The function also removes any unknown tokens and returns the decoded texts as a

list of strings. The function utilizes the num_to_char function to map numeric values back to their

corresponding characters. Overall, the function is an essential step in the OCR process, as it allows

us to obtain the final text output from the model's predictions.

Argument :

pred_label : These are the model predictions which are needed to be decoded.

Return:

filtered_text : This is the list of all the decoded and processed predictions.

'''

Input length

input_len = np.ones(shape=pred_label.shape[0]) * pred_label.shape[1]

CTC decode

decode = keras.backend.ctc_decode(pred_label, input_length=input_len, greedy=True)[0][0][:,MAX_LABEL_LENGTH]

Converting numerics back to their character values

chars = num_to_char(decode)

Join all the characters

texts = [tf.strings.reduce_join(inputs=char).numpy().decode('UTF-8') for char in chars]

Remove the unknown token

filtered_texts = [text.replace('[UNK]', " ").strip() for text in texts]

```

    return filtered_texts

# Model required for inference
inference_model = keras.Model(
    inputs=ocr_model.get_layer(name="image").input,
    outputs=ocr_model.get_layer(name='dense_1').output
)

# Model summary
inference_model.summary()

show_images(data=test_ds, model=inference_model, decode_pred=decode_pred, cmap='binary')

show_images(data=valid_ds, model=inference_model, decode_pred=decode_pred,
cmap='binary')

# Model required for inference
inference_model_2 = keras.Model(
    inputs=ocr_model_2.get_layer(name="image").input,
    outputs=ocr_model_2.get_layer(name='dense_4').output
)

# Model summary
inference_model_2.summary()

show_images(data=test_ds, model=inference_model_2, decode_pred=decode_pred,
cmap='binary')

show_images(data=valid_ds, model=inference_model_2, decode_pred=decode_pred,
cmap='binary')

from tensorflow.keras.preprocessing.image import load_img, img_to_array

def preprocess_uploaded_image(image_path, target_size=(IMG_WIDTH, IMG_HEIGHT)):
    # Load the uploaded image
    img = load_img(image_path, target_size=target_size)

    # Convert to NumPy array

```

```

img_array = img_to_array(img)
# Normalize the image (if needed)
# img_array = img_array / 255.0 # Example normalization
# Add batch dimension
img_array = img_array.reshape((1,) + img_array.shape)
return img_array

pip install pytesseract
pip install fuzzywuzzy

#testing
import cv2
import os
from skimage import io
from skimage.metrics import structural_similarity as ssim

# Define the folder containing reference images
image_folder = '/content/Test'

# Load the uploaded image
uploaded_image_path = '/content/TEST_0001.jpg'
uploaded_image = io.imread(uploaded_image_path)

# Function to find the most similar image in the folder
def find_most_similar_image(uploaded_image, image_folder):
    most_similar_image = None
    highest_ssim = -1 # Initialize with a very low value

    for filename in os.listdir(image_folder):
        if filename.endswith(('jpg', 'jpeg', 'png')):
            image_path = os.path.join(image_folder, filename)
            reference_image = io.imread(image_path)

```



```

    # Ensure both images have the same dimensions by resizing
    reference_image = cv2.resize(reference_image, (uploaded_image.shape[1],
uploaded_image.shape[0]))

    # Calculate the SSIM (Structural Similarity Index) between images
    similarity = ssim(uploaded_image, reference_image, multichannel=True)

    if similarity > highest_ssim:
        highest_ssim = similarity
        most_similar_image = filename

return most_similar_image, highest_ssim

# Find the most similar image in the folder
similar_image, similarity_score = find_most_similar_image(uploaded_image, image_folder)

if similar_image is not None:
    print(f"Match: The uploaded image is similar to '{similar_image}' with SSIM score
{similarity_score:.2f}")
else:
    print("Not Matched: No similar image found in the folder.")

import os
import pandas as pd
from skimage import io
from skimage.metrics import structural_similarity as ssim
from google.colab.patches import cv2_imshow

# Define the folder containing reference images
image_folder = '/content/Test'

```

```

# Load the uploaded image
uploaded_image_path = '/content/TEST_0001.jpg'
uploaded_image = io.imread(uploaded_image_path)

# Load the CSV file containing filename and identity information
csv_filename = '/content/written_name_test.csv'
df = pd.read_csv(csv_filename)

# Function to find the most similar image in the folder
def find_most_similar_image(uploaded_image, image_folder):
    most_similar_image = None
    highest_ssim = -1 # Initialize with a very low value

    for filename in os.listdir(image_folder):
        if filename.endswith(('jpg', 'jpeg', 'png')):
            image_path = os.path.join(image_folder, filename)
            reference_image = io.imread(image_path)

            # Ensure both images have the same dimensions by resizing
            reference_image = cv2.resize(reference_image, (uploaded_image.shape[1],
uploaded_image.shape[0]))

            # Calculate the SSIM (Structural Similarity Index) between images
            similarity = ssim(uploaded_image, reference_image, multichannel=True)

            if similarity > highest_ssim:
                highest_ssim = similarity
                most_similar_image = filename

```

```

return most_similar_image, highest_ssim

# Find the most similar image in the folder
similar_image, similarity_score = find_most_similar_image(uploaded_image, image_folder)

if similar_image is not None:
    # Retrieve the identity corresponding to the matching image from the CSV file
    matching_row = df[df['FILENAME'] == similar_image]

    if not matching_row.empty:
        matching_identity = matching_row.iloc[0]['IDENTITY']
        print(f"Match: The uploaded image is similar to '{similar_image}' with SSIM score
        {similarity_score:.2f}")
        print(f"Identity: {matching_identity}")

        # Display the matching image in Colab using cv2_imshow
        matching_image_path = os.path.join(image_folder, similar_image)
        matching_image = io.imread(matching_image_path)
        cv2_imshow(matching_image)
    else:
        print("No matching identity found in the CSV file.")
else:
    print("Not Matched: No similar image found in the folder.")

import os
import pandas as pd
from skimage import io
from skimage.metrics import structural_similarity as ssim
from google.colab.patches import cv2_imshow

# Define the folder containing reference images

```

```

image_folder = '/content/Test'

# Load the uploaded image
uploaded_image_path = '/content/TEST_0006.jpg'
uploaded_image = io.imread(uploaded_image_path)

# Load the CSV file containing filename and identity information
csv_filename = '/content/written_name_test.csv'
df = pd.read_csv(csv_filename)

# Function to find the most similar image in the folder
def find_most_similar_image(uploaded_image, image_folder):
    most_similar_image = None
    highest_ssim = -1 # Initialize with a very low value

    for filename in os.listdir(image_folder):
        if filename.endswith(('jpg', 'jpeg', 'png')):
            image_path = os.path.join(image_folder, filename)
            reference_image = io.imread(image_path)

            # Ensure both images have the same dimensions by resizing
            reference_image = cv2.resize(reference_image, (uploaded_image.shape[1],
uploaded_image.shape[0]))

            # Calculate the SSIM (Structural Similarity Index) between images
            similarity = ssim(uploaded_image, reference_image, multichannel=True)

            if similarity > highest_ssim:
                highest_ssim = similarity
                most_similar_image = filename

```

```

    return most_similar_image, highest_ssim

# Find the most similar image in the folder
similar_image, similarity_score = find_most_similar_image(uploaded_image, image_folder)

if similar_image is not None:
    # Retrieve the identity corresponding to the matching image from the CSV file
    matching_row = df[df['FILENAME'] == similar_image]

    if not matching_row.empty:
        matching_identity = matching_row.iloc[0]['IDENTITY']
        print(f"Match: The uploaded image is Matched to '{similar_image}' with SSIM score {similarity_score:.3f}")
        print(f"Identity: {matching_identity}")

        # Display the matching image in Colab using cv2_imshow
        matching_image_path = os.path.join(image_folder, similar_image)
        matching_image = io.imread(matching_image_path)
        cv2_imshow(matching_image)
    else:
        print("No matching identity found in the CSV file.")
else:
    print("Not Matched: No similar image found in the folder.")

import os
import pandas as pd
from skimage import io
from skimage.metrics import structural_similarity as ssim
from google.colab.patches import cv2_imshow

```

```

# Define the folder containing reference images
image_folder = '/content/Test'

# Load the uploaded image
uploaded_image_path = '/content/handwriting name.jpg'
uploaded_image = io.imread(uploaded_image_path)

# Load the CSV file containing filename and identity information
csv_filename = '/content/written_name_test.csv'
df = pd.read_csv(csv_filename)

# Function to find the most similar image in the folder
def find_most_similar_image(uploaded_image, image_folder):
    most_similar_image = None
    highest_ssim = -1 # Initialize with a very low value

    for filename in os.listdir(image_folder):
        if filename.endswith(('jpg', 'jpeg', 'png')):
            image_path = os.path.join(image_folder, filename)
            reference_image = io.imread(image_path)

            # Ensure both images have the same dimensions by resizing
            reference_image = cv2.resize(reference_image, (uploaded_image.shape[1],
uploaded_image.shape[0]))

            # Calculate the SSIM (Structural Similarity Index) between images
            similarity = ssim(uploaded_image, reference_image, multichannel=True)

            if similarity > highest_ssim:
                highest_ssim = similarity

```

```

        most_similar_image = filename

    return most_similar_image, highest_ssim

# Find the most similar image in the folder
similar_image, similarity_score = find_most_similar_image(uploaded_image, image_folder)

if similar_image is not None:
    # Retrieve the identity corresponding to the matching image from the CSV file
    matching_row = df[df['FILENAME'] == similar_image]

    if not matching_row.empty:
        matching_identity = matching_row.iloc[0]['IDENTITY']
        print(f"Match: The uploaded image is Matched to '{similar_image}' with SSIM score {similarity_score:.3f}")
        print(f"Identity: {matching_identity}")

        # Display the matching image in Colab using cv2_imshow
        matching_image_path = os.path.join(image_folder, similar_image)
        matching_image = io.imread(matching_image_path)
        cv2_imshow(matching_image)
    else:
        print("No matching identity found in the CSV file.")
else:
    print("Not Matched: No similar image found in the folder.")

```

html

```

<!-- templates/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>OCR Web App</title>
<link rel="stylesheet" href= " {{ url_for('static',filename='styles/styles.css') }}">
</head>
<body>

    <h1 class="heading" >Handwriting Identification</h1>

    <div class="container">

        <form method="post" action="/predict" enctype="multipart/form-data">
            <label for="file">Choose an image file:</label>
            <input type="file" id="file" name="image" accept=".png, .jpg, .jpeg"><br>
            <button class="predictBtn" type="submit">Predict</button>
        </form>
        <b>{{ match }}</b><br>
        <b>{{ var }}</b>
    </div>
</body>
</html>

```

App

```

from flask import Flask, render_template, request
from PIL import Image
import numpy as np
import os
import pandas as pd
from skimage import io
from skimage.metrics import structural_similarity as ssim
import cv2
# from google.colab.patches import cv2_imshow

app = Flask(__name__)
csv_filename = './content/written_name_test.csv'
df = pd.read_csv(csv_filename)
def find_most_similar_image(uploaded_image, image_folder):
    most_similar_image = None
    highest_ssim = -1

    for filename in os.listdir(image_folder):
        if filename.endswith((' .jpg', ' .jpeg', ' .png')):
            image_path = os.path.join(image_folder, filename)
            reference_image = io.imread(image_path)
            reference_image = cv2.resize(reference_image, (uploaded_image.shape[1],
uploaded_image.shape[0]))

            similarity = ssim(uploaded_image, reference_image, multichannel=True, channel_axis =
2)

```



```

        print(similarity)
        if similarity > highest_ssim:
            highest_ssim = similarity
            most_similar_image = filename

    return most_similar_image, highest_ssim

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    file = request.files['image']
    img_np = np.squeeze(file)
    file.save('./templates/handwriting name.jpg')
    uploaded_image = io.imread('./templates/handwriting name.jpg')
    image_folder = './content'
    similar_image, similarity_score = find_most_similar_image(uploaded_image, image_folder)
    if similar_image is not None:
        # Retrieve the identity corresponding to the matching image from the CSV file
        matching_row = df[df['FILENAME'] == similar_image]
        print(matching_row)
        if not matching_row.empty:
            matching_identity = matching_row.iloc[0]['IDENTITY']
            # f"Match: The uploaded image is Matched to '{similar_image}' with SSIM score
            {similarity_score:.3f}")
            return render_template('index.html', var = f"Identity: {matching_identity}", match =
f"Match: The uploaded image is Matched to '{similar_image}' with SSIM score
{similarity_score:.3f}")

            # Display the matching image in Colab using cv2_imshow
            matching_image_path = os.path.join(image_folder, similar_image)
            matching_image = io.imread(matching_image_path)
            # cv2_imshow(matching_image)
        else:
            return render_template('index.html', var = "No matching identity found in the CSV file.")
    else:
        return render_template('index.html', var = "Not Matched: No similar image found in the
folder.")

if __name__ == '__main__':
    app.run(debug=True)

```

style.css

```
body{
  font-family: system-ui, -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen,
  Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
  background-image: url(./caligraphy.jpg);
  background-repeat: no-repeat;
  background-size: 100%;
}
.container{
  margin:250px auto;
  width: 500px;
  text-align: center;
  background: rgba(0, 0, 0, 0.3);
  border-radius: 12px;
  padding: 25px;
}
form > label {
  font-size: 18px;
}
.container > b {
  font-size: 30px;
}
.predictBtn{
  border: none;
  background-color: black;
  padding: 8px 24px;
  border-radius: 12px;
  font-size: 16px;
  color: white;
  margin-top: 20px;
  cursor: pointer;
}
.heading{
  width: fit-content;
  margin: 30px auto;
  font-size: 50px;
}
```