UNIVERSITY OF LONDON

INTERNATIONAL PROGRAMMES

BSc Computer Science and Related Subjects

CM3070 PROJECT
PRELIMINARY PROJECT REPORT

<Optimising CNNs on uTHCD for Tamil Handwritten Character Recognition>

Author: C Sanjana Rajendran
Student Number : 190429454
Date of Submission: 9 September 2024
Supervisor : Mr Don Wee

Contents

1. INTRODUCTION

1.1 Project Concept and Motivation

Handwriting recognition has become increasingly vital in the digital age, particularly for streamlining business processes and enhancing accessibility. While deep learning has significantly advanced this field, languages with unique and complex scripts, such as Tamil, remain challenging due to limited high-quality datasets of handwritten characters. Tamil, a language with a rich 2000-year history, is spoken by a vast population and is an official language in several countries [4]. Its script, comprising 12 vowels, 18 consonants, a special character, and 216 compound characters (Fig.1) is morphologically rich and results in a large character set with more angles and modifiers [18] such as diacritical[1] marks [10]. These complex formations, including overlapping characters and diverse writing styles [4], pose a unique challenge for character recognition systems as opposed to Latin scripts that have achieved greater success in this area [10].

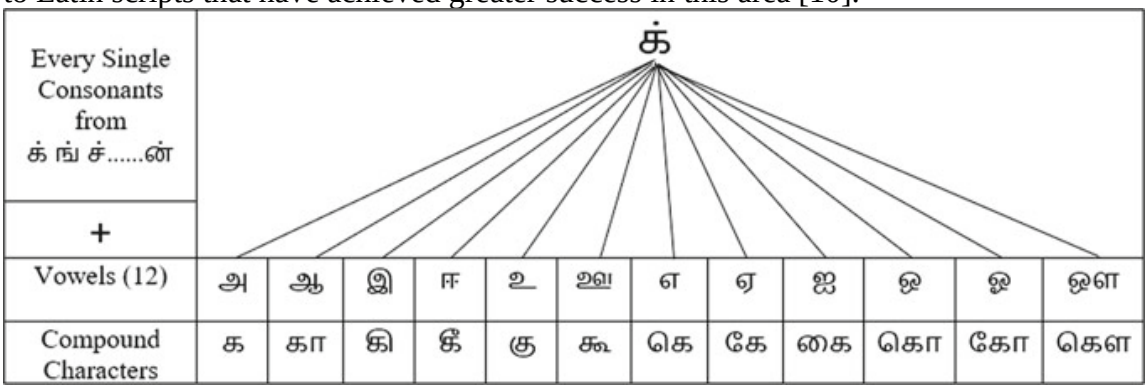


FIGURE 1. Combination of every consonant with 12 vowels yields 216 compound characters - Illustrated with a single consonant [18]

The development of robust **Tamil Handwritten Character Recognition (THCR)** systems has been hindered by the lack of diverse, standardized handwriting databases that may not adequately capture these complexities [4]. Until recently, only one standardized database ("HPL Isolated Handwritten Tamil Character Dataset," 2021) [19] was available for Tamil characters, limiting research potential.

1.2 Project Aims and Deliverable

This project aims to address some of these challenges and advance the field of Offline Tamil Handwritten Character by developing and optimizing cutting-edge deep learning models utilising the comprehensive **Unconstrained Tamil Handwritten Character Database (uTHCD)** [20]. This dataset was recently published and offers improved research potential compared to previously available resources.

1.2.1 Aims and Research Questions

We seek to answer the following research questions:

[1] Diacritic: a mark near or through an orthographic or phonetic character or combination of characters indicating a phonetic value different from that given the unmarked or otherwise marked element (Merriam-Webster, n.d.).

1. How can we improve upon the current benchmark performance for Tamil Handwritten Character Recognition using the uTHCD dataset?
2. What deep learning techniques and model architectures are most effective for recognizing Tamil handwritten characters?
3. How can current state-of-the-art deep learning models be optimized effectively for the uTHCD?
4. How does the performance of models trained on uTHCD compare to those trained on previously available datasets?

1.2.2 Objectives and Deliverables

To address these research questions, we have set the following objectives, each paired with a specific deliverable:

1) Objective: Reproduce the benchmark results published alongside the uTHCD dataset.

Deliverable: A report detailing the reproduction process, including code implementation and performance analysis.

- This will establish a validated baseline for comparison and ensure a thorough understanding of the benchmark methodology.

2) Objective: Implement and evaluate additional deep learning models and techniques based on research from other Tamil datasets and HCR studies.

Deliverable: A comprehensive comparative analysis report of various models and techniques, including their performance and evaluation metrics on the uTHCD dataset.

- This will identify promising approaches from related research that may improve THCR performance.

3) Objective: Develop an optimized model that improves upon the benchmark performance for Tamil character recognition.

Deliverable: A trained model with improved accuracy on the uTHCD test set, accompanied by a technical report detailing the model architecture, training process, and performance analysis.

- This will demonstrate the project's contribution to advancing THCR technology and provide a new state-of-the-art benchmark.

1.3 Project Scope and Template

This project focuses on Offline Tamil Handwritten Character Recognition, a subfield of Optical Character Recognition (OCR). Offline Handwritten Character Recognition (HCR) involves recognizing characters from static images of handwritten text, such as scanned paper documents [c].

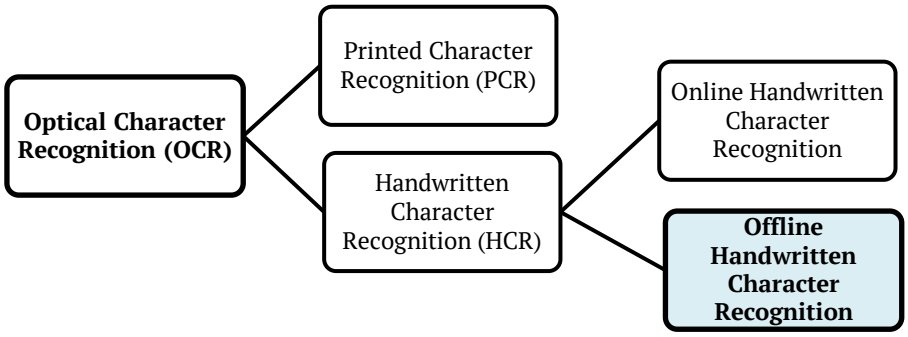


FIGURE 2. Classification of Optical Character Recognition Systems [11]

THCR can be effectively framed as a multi-class image classification task. This involves training a model to learn discriminative features from a large dataset of labelled character images, enabling it to categorize new, unseen character images with high precision. Therefore, we will be using the "Deep Learning on a public dataset" project template.

1.3.1 Template choice: Deep Learning for HCR

Prior to deep learning, HCR relied on traditional machine learning methods which involved manually extracting features like loops, inflection points, and aspect ratios from individual characters to be fed to classifiers like Support Vector Machines (SVMs) or Hidden Markov Models (HMMs) for training and prediction [c]. However, reliance on handcrafted features that require domain expertise and more time to develop, have limited model performance [1]. Training on these features designed for specific datasets could also constrain the model's learning capacity, hindering generalization to unseen variations [1], and result in poor scalability [21].

The remarkable success of deep learning in other OCR tasks has led to its widespread adoption for HCR, attributed to its their competence in learning and recognising complex patterns from large amounts of data [10]. A prominent technique within this realm is the Convolutional Neural Networks (CNNs) [6], which are a multi-layer feed forward neural network [4] particularly adept at extracting features from complex image data [6]. CNNs address challenges across various computer vision tasks, circumventing the high dimensionality problem and extracting robust features [4], due to two key advantages [18]:

1. Automatic learning of prominent features — eliminating the need for handcrafted feature engineering [1], and
2. The use of shared weights — reducing of the large number of parameters [14],

thus, reducing training time, improving generalization and resource efficiency [11]. Deep learning models, including CNNs in comparison with alternative learning algorithms have shown superior performance [6], further affirming their favourability for THCR. Various approaches have achieved high recognition accuracy for languages such as English, Chinese, Malayalam, Devanagari, Arabic, and Telugu using CNNs [14].

(906 words)

2. LITERATURE REVIEW

2.1 Datasets in THCR

2.1.1 Importance of standardised databases

Standardized databases play a crucial role in ensuring the robustness of a typical HCR system [4] by increasing the availability of comprehensive supervised data samples. By providing an unbiased platform for comparison on the same grid [2], they enable objective assessment of the true efficacy/ limitations of state-of-the-art (SOTA) algorithms [4], advancing research and development. This is attested by datasets such as CASIA-OLHWDB and CASIA-HWDB for Chinese and ETL9B for Japanese that have facilitated achieving the best results with SOTA CNNs [c].

For Tamil however, the **HPL-iso-tamil-char (HPL dataset**, [19]), developed by HP Labs India has been the only standardised database available for the Tamil OCR domain, until the publication of the **unconstrained Tamil Handwritten Database (uTHCD)** by (Shaffi and Hajamohideen, 2021, [4]) as shown in in Table 2:

TABLE 1. Standardised and ad-hoc Tamil Handwritten Character Databases for HCR [4][7][13][15]

Database/ Author	Publicly Available	Form	Classes	Samples
uTHCD [4]	Yes	Online, Offline	156	90,950
HPL-iso-tamil-char [19] [7][4]	Yes	Online*	156	77,617
Jose et al. [7][4]	No	Offline	100	10,000 (~100/class)
Shanti et al. [7][4]			106	35,441
Vinotheni et al. [7][4]			156	54,600 (~350/class)
Devendiran et al. (2024) [13]			Characters with diacritic symbols	Not mentioned
Gnanasivam et al. (2020) [15]			121	40,000 (~350/class)

*Offline samples artificially produced from online data

Considering that some methods have produced good performance only on self-collected data possessing several limitations often restricted in scope, the reliability of such algorithms being deployed for practical applications is in question [4]. Thus, emphasising the importance of standardised databases.

2.1.2 Comparative Analysis of Standardised Datasets

Even as a standardised database, the HPL dataset presents certain constraints. The following demonstrates how they are effectively mitigated by the uTHCD:

TABLE 2. Comparison of Standardised Tamil Handwritten Character Databases [4]

	HPL	*uTHCD*
Sample size and Distribution	~ 77,620 samples	90,950 samples
	270 to 571 samples/ class	600 samples/ class
Annotation Quality	Contains annotation errors	Meticulously verified for accurate annotations
Data collection	Constrained	Unconstrained
Form	Only digital (Online) samples collected*	Offline and Online samples collected
Stroke Characteristics	Single-pixel width strokes	Variable thickness of strokes
Sparsity: Foreground Pixel Density	~2%	~20%
Classifier Performance	Mixed (poor on RF/KNN, better on SVM)	Acceptable across classifiers

*Through the interpolation of online pen coordinates with a constant thickening factor

Benefits of the uTHCD over the HPL dataset and ad-hoc collections for THCR:

- Its larger, balanced sample size and higher quality annotations [4] ensure unbiased model training and reliable results.
- Including offline samples captures a wider range of handwriting characteristics, such as stroke thickness, discontinuities, and overwriting distortions that enhance real-world applicability [4].
- Reduced sparsity and variable stroke thickness [4] provide richer information for feature extraction.
- With better performance and consistency across different machine learning approaches, it is a more robust choice for OCR applications [4].

The HPL dataset having been a de-facto standard [4] with majority of the standard papers based on it [c][12][14] , or with most studies based on often incomplete/ incomprehensive self-collected data in limited scope (subsets of the script) [18][13][15], the uTHCD is positioned to become a new standard for THCR research, addressing key limitations. Thus, potentially advancing the field of Tamil OCR research [4]. Most standard research has been based on this dataset [4], State-of-the-Art approaches in THCR.

2.1.3 Kavitha and Srimathi [2022] – Benchmarking on HPL Dataset

"Benchmarking on offline Handwritten Tamil Character Recognition using convolutional neural networks" [c]

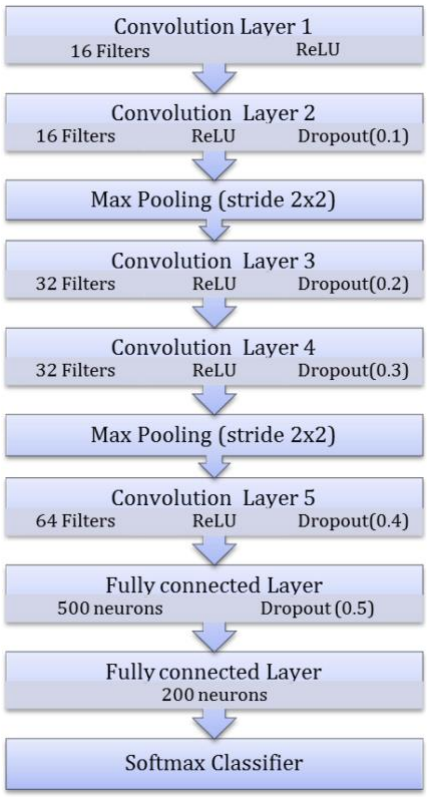


FIGURE 3. Architecture of the proposed model [3]

II. FINDINGS

The high test (Table.4) and validation accuracy (92.74%), demonstrates the model's ability to generalize well to unseen data. In general, the clear evolution of performance from traditional methods (clustering, SVM) to deep learning (CNN), reveals its effectiveness, especially when handling all 156 classes.

This work sought to establish a comprehensive benchmark for THCR utilising all 156 classes of the HPL dataset, thereby addressing the lack of an established benchmark for the standardised database.

I. TECHNIQUES AND METHODS

CNN-based approach: The proposed CNN architecture (Fig. 3) consisted of 9 layers: 5 convolutional layers, 2 max-pooling layers, and 2 fully connected layers, with the stride value of 1 and zero padding to preserve border information [c].

Optimization:
Early stopping was implemented to prevent overtraining. Incremental dropout probability starting at 0.1 and increasing by 0.1 in each layer was used to combat overfitting [c]. This reduced variance by 7.95% resulting in a trade-off between bias and variance that improved the model's generalization capabilities.

TABLE 3. Comparison of approach with previous work [c]

Authors	Dataset	No. of classes	Method	Training Accuracy (%)	Test Accuracy (%)
Bhattacharya et al. (2007)	HPLabs dataset	156	Clustering and groupwise classification	92.77	89.66
Shanthi and Duraiswamy (2010)	Own dataset	34	SVM	–	82.04
Vijayaraghavan and Sra (2014)	HPLabs dataset	35	CNN	99	94.4
Our work	HPLabs dataset	156	CNN	95.16	97.7

III. INSIGHTS AND CONTRIBUTION

In comparison with the other CNN methodology by (Vijayaraghavan and Sra, 2014 [17])

- The deeper structure of this model allowed for more complex feature extraction and classification, based on the high accuracy achieved while handling a much larger character set, and greater test accuracy than [17]. This suggests an effective optimization of learning capacity of the model, beneficial for comprehensive HCR tasks.

- The dropout strategy has been effective in improving generalization, and in reducing variance found in [17]. The strategy differed in introducing dropout in convolutional layers and at higher probabilities.

In addition,

- Further tuning of hyperparameters like stride, padding, and depth are suggested to enhance performance [c], providing direction for future optimization efforts.
- The identification of writing ambiguities as a primary source of misclassification errors [c] offers valuable insights for future improvements in THCR systems.

This study underscores the advantages of CNNs and the importance of standardized datasets for meaningful comparisons across studies. It provides potential for testing the model's performance on more diverse real-world datasets.

TABLE 4. HPL/uTHCD CNN Model Comparison [4, 5]

	HPL Benchmark [c]	uTHCD Benchmark [4]
Input Shape	64x64x1 (grayscale image)	
Output	156 classes (Softmax activation)	
Activation Function	ReLU (except output layer)	
Optimizer	Adam	
Learning Rate	0.001	
Weight Initialization	Xavier (Glorot)	
Early Stopping	✓	
Network Structure*	16C3-16C3-MP2-32C3-32C3-MP2-64C3-500N-200N	64C3-MP2-64C3-MP2-1024N-512N
Dropout	Incremental, starting at 0.1, increasing by 0.1 in each layer	Conv1: 0.1, Conv2: 0.05, Dense layers: 0.5
Batch Size	64	32
Max Epochs	100	200
Data Augmentation	—	✓ Rotation, zoom, shift

* Where nCi represents a convolution layer with n filters and i × i kernel, MPj represents a max pooling layer with j × j kernel and kN represents a fully connected layer with k neurons

2.1.4 Shaffi and Hajamohideen [2021] – Benchmarking on uTHCD

"uTHCD: A New Benchmarking for Tamil Handwritten OCR" [4]
Extending the creation of the uTHCD, this work presented use cases including CNNs and transfer learning to classify handwritten Tamil characters. Thus, setting a new benchmark in THCR, propelling the development of robust language technologies for the Tamil script.

I. TECHNIQUES AND METHODS

A. Basic CNN:

A basic CNN of 6 layers: 2 convolutional layers, 2 max-pooling layers, and 2 fully connected layers (Fig.4), was first constructed with similar parameters as in (Kavitha and Srimathi, 2022 [c]), differing mainly in architecture, dropout and with the addition of data augmentation as shown in Table. 5 above.

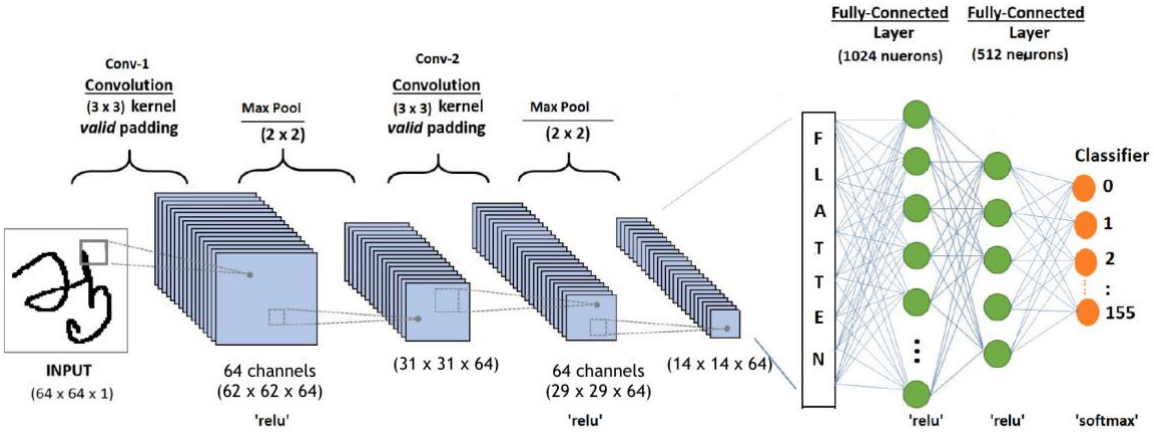


FIGURE 4. Basic CNN architecture used [4]

Dropout:

TABLE 5. Dropout in the convolutional layers [4]

Conv1 Layer	Conv2 Layer	Train	Test	Validation
0.05	0.05	96.09	89.26	92.81
0.05	0.10	95.55	90.68	94.14
0.10	0.05	**96.51**	**91.10**	**94.46**
0.10	0.10	94.38	90.27	93.62

Just as in [c], dropout was introduced in the convolutional layers as well but in lower probabilities, in view of better results (Table.6).

Data Augmentation

Data augmentation was implemented to supplement the training samples with artificially synthesized data from actual data by distorting the images as mentioned (Fig.5) [4]. Its regularization effect was substantiated by the improvement in validation accuracy [4] with an increase in augmented samples used for training.

FIGURE 5. Sample augmented images belonging to a character [Row-1: Rotation, Row-2: Horizontal Shift, Row-3: Vertical Shift, Row-4: Zoom, Row-5: Multiple effects] [4]

B. Transfer Learning

Transfer learning was employed to fine-tune the baseline model using Visual Geometry Group 16 (VGG16) [4], a 16-layer CNN pre-trained on ImageNet. This technique repurposes models trained on other data for new tasks, enhancing efficiency and performance with limited data. VGG16 was adapted by retaining ImageNet weights, modifying the top layers, and adding two dense layers for this task [4].

This model converged much faster than the basic CNN model, rapidly overfitting without dropout, indicating the simplicity of this task compared to what it was originally trained for [4].

II. FINDINGS

TABLE 6. Established Benchmark on uTHCD [4]

CNN Model Used	Test Accuracy	Train Accuracy	Val Accuracy	False Positive Rate (1-Specificity)	True Positive Rate (Sensitivity)	F1-Score	Correctly Classified Samples	Incorrectly Classified Samples
Basic Model	87.00	99.93	93.13	0.0008	0.8701	0.8703	24429	3651
Basic Model with Early Stopping	88.27	97.82	92.89	0.0008	0.8826	0.8829	24785	3295
Basic Model with Early Stopping and Dropout	91.10	96.51	94.46	0.0006	0.9109	0.9111	25582	2498
Basic Model with Early Stopping, Dropout and Augmentation	**93.16**	95.76	98.35	0.0004	0.9315	0.9314	26158	1922
Fine tuned VGG16 Model with Early Stopping	89.77	96.13	92.03	0.0007	0.8977	0.8978	25208	2872
Fine tuned VGG16 model With Early Stopping and Dropout	**92.32**	92.05	93.73	0.0004	0.9233	0.9232	25925	2155

Iterative development is demonstrated by the progression in model performance with the application of different techniques. Early Stopping improves generalization, with Dropout further enhancing model performance, and Data Augmentation boosting test accuracy.

- There is still some overfitting in the best model with higher train than test accuracy, suggesting that stronger values could be used for transformations while augmenting the training set [4].
- The success of Transfer learning in enhancing generalization with fewer training samples [4] is evidenced by the closer train and validation accuracies. It thus offers a good balance of performance and potentially faster training.

III. INSIGHTS AND CONTRIBUTION

- The Basic Model enhanced with a deeper architecture to improve learning capacity (following the success of the previous study [c]), regularized with data augmentation techniques introduced here, could better leverage the richness of the uTHCD.
- The suggestion of further optimization of the Transfer learning approach, such as fine-tuning the learning rate and making more layers trainable [4] shows promise in better performance and provides direction in capitalising on this technique.
- Additional metrics ensure a fair comparison and comprehensive evaluation of model performance as shown in Table 5.

2.1.5 Devendiran et al. [2024] – Focus on diacritic symbols

"Handwritten Text Recognition using VGG19 and HOG Feature Descriptors" [13]
This study contrasts the performance of Histogram of Oriented Gradients (HOG) based feature extraction against the convolutional neural network approach of VGG-19 for THCR with a focus on diacritic symbols conducted on self-collected data. Both models were based on neural networks.

I. TECHNIQUES AND METHODS

Feature Extraction Techniques
Characters before pre-processing and after going through HOG feature extraction are shown in Fig.6.

FIGURE 6. Before (left), after (right) - colours inverted for readability [13]

A. HOG: Excels in extracting fine details of handwritten Tamil characters, including curves, angles, and strokes. It analyses gradient and edge directions in localized sections, making it effective for distinguishing between characters despite variations in handwriting styles.
B. VGG-19: A deep learning approach that learns hierarchical feature representations through its 19 layers. It captures both low-level features (edges, textures) and high-level patterns (character shapes) of the Tamil script.

Model Development
A. Model A (HOG-based): Required pre-extraction of features, with data passed as an array rather than an image, which simplified processing demands.
B. Model B (VGG-19): A 19-layer deep neural network, pre-trained on over 1000 classes, made it suitable for multi-class classification tasks such as THCR.

II. FINDINGS

TABLE 7. Model analysis [13]

Comparison	Comparing the two models	
	Model	*Accuracy*
1	Model A	91.2%
2	Model B	98.34%

While Model B (VGG-19) significantly outperformed Model A (HOG) in terms of accuracy and efficiency, it was more computationally intensive to train and infer, requiring substantial memory due to its size which could pose as a drawback for deployment on devices with limited resources. Both models struggled with variations in handwriting due

to different writing instruments, paper qualities, and personal styles. HOG, relying on handcrafted features, was particularly limited in this aspect.

III. INSIGHTS AND CONTRIBUTION

- Essentially, VGG-19 demonstrates superior capabilities in handling the intricacies of handwritten Tamil script, demonstrating the potential of convolutional neural networks. However, the choice between deep learning approaches and traditional feature extraction methods involves trade-offs between resource requirements and accuracy
- In consideration of finding a suitable middle ground, it may be beneficial to find other techniques to optimise performance with resource efficiency, and to leverage pre-trained models like VGG-19 using transfer learning instead, where just the weights in convolutional layers are copied, rather than the entire network including fully-connected layers. This is very effective since many image datasets share low-level spatial characteristics that are better learned with big data [9].

2.1.6 Ravi [2024] – Tamil Alphabet Classification

"Handwritten alphabet classification in Tamil language using convolution neural network" [8]
This study focused on classifying Tamil handwritten alphabets into their respective categories using Convolutional Neural Networks (CNNs), with the creation of a benchmark dataset containing images of Tamil alphabets from three categories: vowels, consonants, and combinatory characters. The work is notable for its approach to categorizing Tamil characters rather than recognizing individual characters.

I. TECHNIQUES AND METHODS

Applied data augmentation techniques: horizontal flip, vertical flip, rotation, adding Gaussian noise, and shear.

CNN Models:
Proposed three CNN models for Tamil Handwritten Alphabet Classification (THAC):
1. THAC-CNN1: 2 convolutional layers (32 and 64 filters, 3x3 size)
2. THAC-CNN2: 3 convolutional layers (32, 64, and 128 filters, 3x3 size)
3. THAC-CNN3: 2 convolutional layers (96 and 256 filters, 11x11 and 5x5 sizes)

Model Optimization:
- Used ReLU activation for convolutional layers and Softmax for the output layer.
- Implemented Adam optimizer and Sparse Categorical Crossentropy as the loss function.

II. FINDINGS

- THAC-CNN1 and THAC-CNN2 performed similarly, achieving the best results.
- THAC-CNN3 suffered from overfitting.
- Best performing models (THAC-CNN1 and THAC-CNN2) achieved:

- 97% accuracy on the training data
- 92% accuracy on the test data

- Outperformed pre-trained models VGG-11 and VGG-16, which achieved approximately 72% accuracy on training data and 73% on test data.

III. INSIGHTS AND CONTRIBUTION

1. Novel approach to Tamil character recognition by classifying alphabets into categories rather than individual character recognition.
2. Demonstrated the effectiveness of custom CNN architectures over pre-trained models for this specific task.
3. Highlighted the importance of appropriate model complexity, as evidenced by the overfitting in THAC-CNN3.
4. Emphasized the value of data augmentation in improving model performance, especially for limited datasets.
5. Provided a benchmark for Tamil alphabet category classification, which could be a stepping stone for more complex THCR tasks.

This study contributes to the field of THCR by offering a new perspective on character classification and demonstrating the potential of custom CNN architectures for Tamil script. It also underscores the importance of balancing model complexity with dataset size and the effectiveness of data augmentation techniques in improving model performance.

2.1.7 Challenges and Future Directions

Overall, we can glean several important trends, challenges, and potential directions for future research and implementation:
Simple CNN architectures have shown promising results, with customisations having the potential to outperform pre-trained models. Trade-offs in performance and computational efficiency is highlighted in the choice of architectures.

Thus, pointing to transfer learning approaches that leverage on pre-trained models with additional optimization showing promise in balancing this trade-off, or experimenting with varied structural changes to networks. Some techniques that could be considered are:

- Regularisation:
  - As proven to improve model generalization and performance, expanding on these techniques could yield better results.
  - Further explore data augmentation techniques specific to Tamil characters to improve model robustness.
- Consider the trade-offs between accuracy and computational efficiency, aiming to develop models suitable for deployment on resource-constrained devices.

In conclusion, while significant progress has been made in THCR, there is still ample room for improvement, particularly in developing efficient, accurate, and robust models capable of handling the complexities of Tamil script. Our work could focus on addressing these challenges by leveraging the strengths of existing approaches, or new ideas not as experimented with, and considering the practical aspects of model deployment.

3. PROJECT DESIGN

3.1 Domain and Users

3.1.1 Domain:

This project is situated within the domain of Optical Character Recognition (OCR), specifically focusing on Offline Tamil Handwritten Character Recognition (THCR). This field intersects computational linguistics and document image analysis.

Offline character recognition is the process of converting handwritten characters on paper into a machine-readable format by optically or magnetically scanning the paper document, in contrast with online recognition systems that capture writing in real-time [6]. The offline method is considered a more demanding task than its online counterpart due to challenges such as diverse shapes of characters, variations in document quality and absence of stroke information, etc [6]. The non-exhaustive variability of human handwriting also makes HCR a more complex task than Printed Character Recognition.

THCR addresses the critical steps of character segmentation, feature extraction, and character recognition as part of the comprehensive workflow of OCR as outlined in Fig.7:

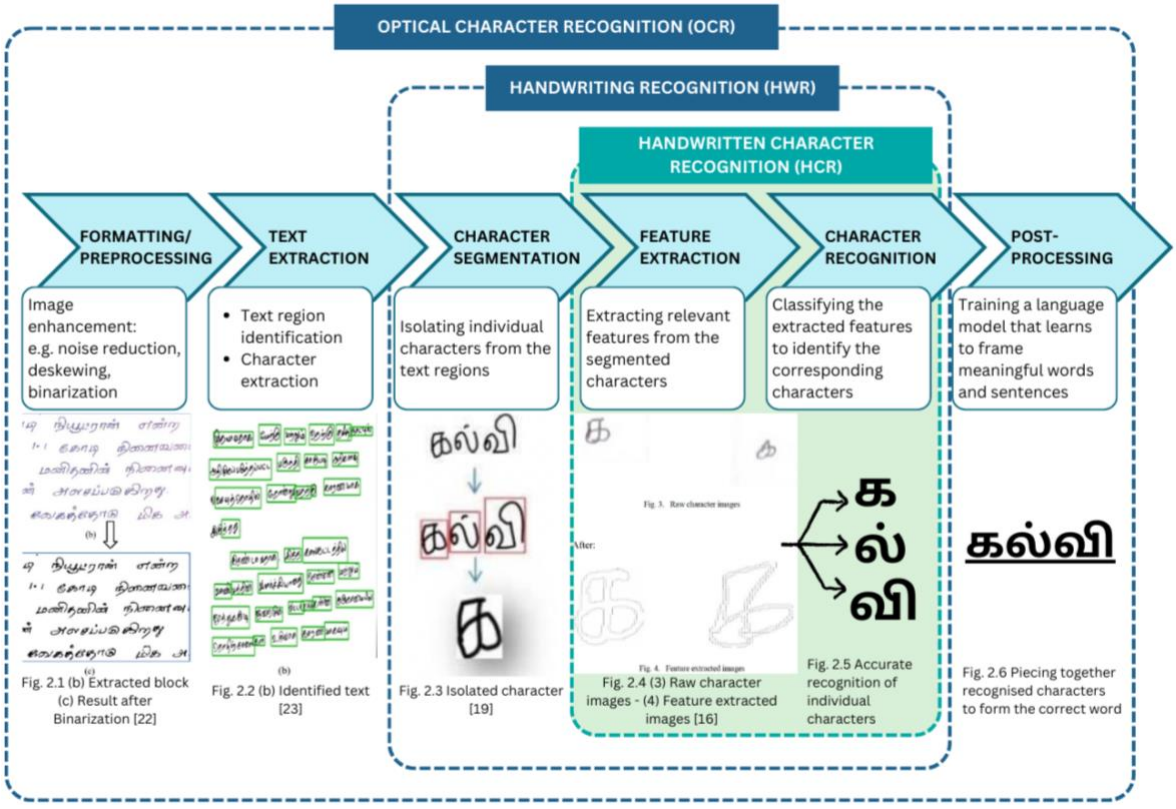


FIGURE 7. End-to-End OCR Process: Detailed Workflow for Handwritten Document Digitisation (Content [11], Illustration [own])

While OCR is broadly applicable to digitising any text or document, THCR is the component that addresses the specific challenges of the Tamil script. Accurate

recognition of these characters forms the building blocks for precise word and sentence recognition in the entire OCR pipeline [22].

The development of a THCR system involves training it with a diverse set of labelled Tamil character images, enabling it to learn and recognize patterns effectively [18], placing it in the field of machine/deep learning. The application of deep learning models enhances the capability to accurately interpret complex scripts like Tamil, characterized by its rich set of characters and diacritic marks.

3.1.2 Users and Stakeholders:

1. Development of a Robust THCR System

The development of a reliable and accurate Tamil Handwritten Character Recognition (THCR) system is crucial for improving OCR-based text recognition in Tamil, bringing substantial benefits across several domains:

- **Cultural Preservation:** The digitization of historical and ancient Tamil manuscripts is essential for preserving cultural heritage and ensuring long-term access to these texts [12][10].
- **Educational and Research Advancements:** Enhanced document processing enables easier information retrieval in education and research, facilitating efficient analysis of handwritten materials [16][4].
- **Facilitating Knowledge Dissemination:** Improved recognition systems can aid in the translation of Tamil documents, promoting the dissemination of knowledge across languages and regions [10].
- **Assistive Technologies:** For visually impaired individuals, THCR systems can power text-to-speech applications, allowing broader access to Tamil documents and literature [10].
- **Automation in Official Documentation:** Automatic processing of handwritten documents can greatly benefit administrative tasks in Tamil-speaking regions, reducing manual efforts and increasing efficiency [4].
- **Forensic Intelligence:** THCR systems could also support forensic applications by aiding in writer identification and authentication of handwritten documents in legal or investigative contexts [4].

2. Broader Impact

This project addresses the gap between handwritten Tamil documents and digital systems, aiming to improve the digitization, search, and analysis of Tamil text, which is currently underrepresented in the digital landscape. In doing so, it enhances digital accessibility and processing efficiency for Tamil text, supporting a range of practical applications.

By focusing on the unique and complex script of Tamil, this project also contributes to the broader goal of developing robust OCR systems for diverse languages and scripts. The methods and findings from this research have the potential to benefit similar efforts in other languages, promoting the development of multi-lingual OCR systems capable of handling various low-resource scripts.

3.2 Design choices

3.2.1 Research Utility

This study will offer important contributions to both the academic community and real-world applications through:

- **Benchmark Reproduction and Optimization:** By reproducing and enhancing the uTHCD benchmark, this study establishes a solid baseline for future research in Tamil OCR. The introduction of techniques like data augmentation, batch normalization, and learning rate scheduling not only improves model performance but also sets the stage for future experiments with larger datasets and transfer learning.
- **Model Efficiency and Scalability:** The focus on improving generalization without increasing model complexity ensures that the techniques developed here can be used in resource-constrained environments, making the model suitable for deployment in regions where computational power is limited.
- **Overfitting Mitigation and Generalization:** The study highlights the critical importance of addressing overfitting in OCR systems. By systematically fine-tuning the model, its ability to generalize to unseen data will be improved, which is essential for real-world applications.

Several key design choices have been identified to ensure the THCR system will meet user needs for reproducibility, accuracy, and resource efficiency:

- **Data Preparation & Augmentation:** Our work will involve designing a robust data processing and augmentation pipeline that preserves the integrity of Tamil characters while introducing variability.
- **Regularization Techniques:** These techniques will be crucial in stabilizing training and ensuring the model generalizes well to unseen data.
- **Adaptive Learning:** This will help optimize resource usage and prevent overtraining, ensuring efficient and effective model learning.
- **Evaluation Framework:** A reproducible evaluation framework will be set up, including standardized train/test splits and metrics such as accuracy, sensitivity, and F1-Score. This will enable clear comparative analysis of model improvements and generalization capabilities in future experimentation.

These design choices will form the foundation for future implementation, aimed at building a scalable and efficient THCR system that meets research and real-world needs

3.3 Overall project structure

- The overall structure of the project (this could be the architecture of a software project, the story of a VR project, the visual interface of a mobile app or the research question and methods of a data science project)

3.4 Identification of important technologies and methods

To achieve the project's aims of developing an efficient and accurate Tamil Handwritten Character Recognition (THCR) system, the following technologies and methods will be employed:

- **Keras and TensorFlow:** These will be used as the primary frameworks for model development, leveraging their flexibility and integration with various deep learning tools to implement and train convolutional neural networks (CNNs).
- **Convolutional Neural Networks (CNNs):** CNNs will be the core architecture for image recognition tasks due to their proven effectiveness in handling image data and extracting spatial hierarchies of features.
- **Data Augmentation:** Techniques like **ImageDataAugmenter** or **Albumentations** (alongside **OpenCV**) will be used to increase dataset variability, helping the model generalize better across different handwriting styles by introducing controlled distortions such as scaling, rotation, and brightness adjustments.
- **Hyperparameter Tuning with Keras Tuner [28]:** This will be employed to fine-tune training parameters, such as learning rate, batch size, and model architecture, optimizing the model’s performance efficiently.
- **Batch Normalization and Learning Rate Scheduling:** These regularization techniques will be incorporated to stabilize training and prevent overfitting by dynamically adjusting the learning rate and normalizing activations.
- **TensorBoard:** It will be used to track and visualize the model's performance over time, allowing for real-time monitoring of accuracy, loss, and learning behavior.
- **DataFrames for Logging Results:** Results from model training and evaluations will be logged using dataframes for systematic tracking and comparison of different model versions.
- **Google Colab with L4 GPU:** The project will leverage Google Colab's high-performance computing environment, specifically the L4 GPU, to handle the computational demands of training deep learning models on large datasets.

3.5 Plan of work

I. June 1 – June 14: Research and Setup

- Study uTHCD benchmark, gather resources, and set up the environment (Google Colab, TensorFlow, Keras).
- Prototype a feature extraction model.

Milestone: Initial setup and prototype completed.
Reporting: Document research and setup.

II. June 15 – June 30: Data Preprocessing and Augmentation

- Implement data preprocessing (normalization, resizing) and reproduce the uTHCD augmentation pipeline.
- Test and validate the pipeline.

Milestone: Preprocessing and augmentation completed.
Reporting: Report on implementation.

III. July 1 – July 14: Augmentation Modification

- Modify augmentation pipeline to address specific issues (e.g., character modifiers).
- Test on data for accuracy.

Milestone: Modified pipeline ready.
Reporting: Report on modification and evaluation.

IV. July 15 – July 31: Model Development

- Develop baseline CNN model, integrate batch normalization, early stopping, learning rate scheduling, and train the model.

Milestone: Initial model trained.
Reporting: Report on training and results.

V. August 1 – August 22: Model Tuning

- Fine-tune parameters using Keras Tuner, analyze performance for overfitting, and track results via TensorBoard.
- Iteratively evaluate the model on un-augmented data, generate metrics (accuracy, F1-Score), log scores.

Milestone: Optimized model and Final evaluation completed.
Reporting: Document tuning and analysis, Report evaluation.

VI. August 23 – September 9: Final Report Writing and Submission

- Write and finalize the report.

3.6 Test and Evaluation plan

The following plan outlines how the THCR system will be tested and evaluated, ensuring it meets key performance objectives. It includes the metrics and techniques used to assess the model's performance and specific criteria for judging these aspects.

3.6.1 Evaluation Metrics

The primary evaluation metrics will include:

- **Training, Test, and Validation Accuracy:** These metrics will measure the percentage of correctly classified Tamil characters in the respective datasets. Performance will be assessed based on:
    - **Train-Validation Difference (Train-Val Diff):** The gap between training accuracy and validation accuracy will be used to detect overfitting. A large train-val difference may indicate that the model is memorizing the training data without generalizing well to unseen data. Ideally, this gap should be minimal.
    - **Test Accuracy:** After training, the model will be evaluated on the test set to assess how well it generalizes to completely unseen data.
- **F1-Score:** This metric will evaluate the balance between **precision** and **recall**, providing a comprehensive view of the model's classification performance, especially when dealing with imbalanced classes. A higher F1-score will indicate the model's effectiveness in recognizing true positive character classifications while minimizing false positives and false negatives.

3.6.2 Techniques for Performance Evaluation

- **Loss and Accuracy Curves:** These will be plotted across epochs to track model performance over time.
    - **Training Accuracy and Loss:** These curves will show how well the model is learning the training data. Rapid drops in training loss combined with high training accuracy may suggest overfitting if validation performance does not follow a similar trend.
    - **Validation Accuracy and Loss:** Validation performance will be used to judge the model's ability to generalize. If validation loss plateaus or increases while training loss continues to decrease, this is a sign of overfitting.
- **Train-Validation Difference:** A small difference between training and validation accuracy indicates that the model generalizes well, while a large difference

suggests overfitting. This difference will be monitored closely and minimized using regularization techniques like early stopping and learning rate scheduling.

- **Evaluation on Un-Augmented Data:** Models will be evaluated on un-augmented training data, even if trained with augmented data, to ensure they perform well on true representations of the dataset. This will demonstrate the model's ability to generalize to real-world handwritten Tamil characters without relying on artificially augmented variations.

3.6.3 Key Aims for Evaluation

The model will be evaluated against the following key objectives:

- **Generalization:** Judged by the consistency of training, validation, and test accuracy. A minimal **train-val difference** and high test accuracy will be indicators of good generalization. Ideally, the model should not perform well only on the training data but should generalize effectively to unseen validation and test sets.
- **Efficiency:** The computational performance of the model will be evaluated based on training time and resource usage. Efficient training will be judged by the number of epochs required to achieve optimal accuracy, particularly if the model utilizes techniques like **early stopping** and **learning rate scheduling** to converge faster without sacrificing accuracy.
- **Robustness to Variability:** The model's ability to correctly classify characters across different handwriting styles will be tested by applying augmentations during training, but performance will be judged on true, un-augmented data to simulate real-world handwritten text.

By following this plan, the project will ensure that the THCR model meets the criteria of **high accuracy**, **robust generalization**, and **efficient computation**, with well-defined metrics and techniques to track performance at every stage.

(1929 words)

4. IMPLEMENTATION

4.1 Data utilities designed for reproducing the uTHCD Benchmark

This file, `*data_utils.py*`, contains a collection of utility functions for data processing, augmentation, and visualization for this project. It is designed to support the workflow of pre-processing the uTHCD for training on CNNs using Keras and Tensorflow. It is structured to allow for flexible use in different parts of the project, with a focus on image data processing and augmentation techniques. This encapsulation allows accessible modification of various training parameters such as augmentation factor, streamlined visualisation of sample data and verification of data processing.

I. Import Libraries

```
import numpy as np
import random
import math
import h5py
import matplotlib.pyplot as plt
# Tensorflow
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
# ImageDataGenerator
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Albumentations
import albumentations as A
from tqdm import tqdm
# OpenCV
import cv2
# sklearn
from sklearn.utils import shuffle
```

II. Reproducibility:

The file sets random seeds for NumPy and TensorFlow to ensure reproducible results

```
# Set random seeds for reproducibility
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
```

III. Load and Extract Data

```
def load_data(file_path):

    # Open the uTHCD compressed HDF5 file in read mode
    with h5py.File(file_path, 'r') as hdf:

        # Extract training data and labels and convert to numpy arrays
        x_train = np.array(hdf['Train Data']['x_train'])
        y_train = np.array(hdf['Train Data']['y_train'])

        # Extract test data and labels and convert to numpy arrays
        x_test = np.array(hdf['Test Data']['x_test'])
        y_test = np.array(hdf['Test Data']['y_test'])

        # Extract validation set from the training set (last 7870 samples)
        x_val = x_train[-7870:]
        y_val = y_train[-7870:]

        # Update training set to exclude the validation set
        x_train = x_train[:-7870]
        y_train = y_train[:-7870]

        val_size = x_val.shape[0] # Size of validation set

    # Print shapes to verify the extracted data
    print(f"Data loaded ===============")
    print(f"Extracted data (X) and labels (Y) shapes:")
    print(f"Training data   | X: {x_train.shape}, Y: {y_train.shape}")
    print(f"Validation data | X: {x_val.shape},   Y: {y_val.shape}")
    print(f"Test data       | X: {x_test.shape},  Y: {y_test.shape}")

    return (x_train, y_train), (x_val, y_val), (x_test, y_test), val_size
```

- Loads data from an HDF5 file, splitting it into training, validation, and test sets.
- Converts data into NumPy arrays as required for model training.
- The data is already split into a 70:30 Train: Test set ratio in the downloaded file, with 62870 training samples split in a 7:8 ratio of train: validation samples, and 28080 test samples. Validation size is specified by the code provided in the paper, which this function follows. [Appendix. B]

IV. Data Pre-processing

- Ensure correct dimensionality of data:

```
def ensure_4d(data):
    if data.ndim != 4:
        # Reshape the data to add the channel dimension (for grayscale images)
        data = data.reshape((data.shape[0], data.shape[1], data.shape[2], 1))
    elif data.ndim == 4:
        return data
    else:
        raise ValueError(f"Unexpected number of dimensions: {data.ndim}.")
    return data
```

Ensures input data is 4-dimensional (for image data), adding a colour channel for grayscale images as expected by the Tensorflow environment.

- Normalize data:

```
def normalize_data(x_train, x_val, x_test):
    # Normalize data: divide by 255.0
    return x_train / 255.0, x_val / 255.0, x_test / 255.0
```

Normalizes image data to the range [0, 1], from the range of 0-255 of image pixels.

- Encode Labels:

```
def encode_labels(y_train, y_val, y_test):
    # Number of unique classes in the training labels
    num_classes = len(np.unique(y_train))

    # Verify the number of classes
    print(f"Number of unique classes: {num_classes}")

    # One-hot encode the labels for each dataset
    return [to_categorical(y, num_classes) for y in (y_train, y_val, y_test)]
```

One-hot encodes the labels, as necessary for machine learning tasks.

V. Data Augmentation:

To replicate the uTHCD Benchmark, Data augmentation was implemented as suggested in the study. The quantity of samples generated were controlled by the variable ‘augmentation factor’ as shown in Table. 8. These were defined by two strategies:

1. **Full augmentation**: Every image in the training set is augmented by implementing one or more of the variances mentioned, maintaining perfect balance in the training set with an equal number of samples in all classes, as implemented below:

- **Full data augmentation on all training data**

```
def full_augmentation(x_train, y_train, datagen, augmentation_factor):
    # Determine number of augmented samples required
    total_samples = math.ceil(len(x_train) * augmentation_factor)
    # Empty arrays to hold the augmented data
    x_augmented = np.zeros((total_samples,) + x_train.shape[1:])
    y_augmented = np.zeros((total_samples,))

    sample_count = 0 # Initialize counter
    # Generate batches of augmented data
    for x_batch, y_batch in datagen.flow(x_train, y_train, batch_size=32,
seed=RANDOM_SEED):
        for i in range(len(x_batch)):
            if sample_count < total_samples:
                # Add augmented samples to the arrays
                x_augmented[sample_count] = x_batch[i]
                y_augmented[sample_count] = y_batch[i]
                sample_count += 1
            else:
                break                   # Exit when total number of samples is reached
        if sample_count >= total_samples:
            break                       # Exit if total number of samples is reached
    return x_augmented, y_augmented
```

TABLE 8. Results for varying augmentation factors [c]

Factor	# of Samples	Train	Test	Validation	Diff
1.0	125740	96.07	92.32	96.70	3.75
2.0	188610	95.57	92.86	97.78	2.71
3.0	251480	95.76	93.32	98.35	2.44
4.0	314340	95.63	93.31	98.53	2.32

2. **Random augmentation**: A randomly chosen training image transforms as mentioned earlier, maintaining relative balance of the training set, as implemented below:

Therefore, yielding the numbers as shown in Table. 8 as such:

Factor	# Training Samples	# Fully augmented samples	# Randomly augmented samples	# Validation samples	# Total training samples	# of Samples ***(Table. 8)***
f	t	f x t	f x v	v	t + (f x t) + (f x v)	t + (f x t) + (f x v) + v
1	55000	55000	7870	7870	117870	125740
2	55000	110000	15740	7870	180740	188610
3	55000	165000	23610	7870	243610	251480
4	55000	220000	31480	7870	306480	314350

- **Random data augmentation on a subset of the training dataset.**

```
def random_augmentation(x_train, y_train, val_size, datagen, augmentation_factor):
    # Calculate the number of samples to augment
    num_to_augment = math.ceil(val_size * augmentation_factor)
    # Randomly select indices for augmentation, with replacement
    rng = np.random.default_rng(RANDOM_SEED)
    indices_to_augment = rng.choice(len(x_train), num_to_augment, replace=True)
    # Arrays to hold the augmented data
    x_augmented = np.zeros((num_to_augment,) + x_train.shape[1:])
    y_augmented = np.zeros(num_to_augment)

    sample_count = 0  # Initialize counter
    # Augment the selected samples
    for idx in indices_to_augment:
        # Extract a single sample from x_train and y_train
        x_sample = x_train[idx:idx+1]
        y_sample = y_train[idx:idx+1]
        # Generate an augmented version of the sample
        for x_aug, y_aug in datagen.flow(x_sample, y_sample, batch_size=1,
seed=RANDOM_SEED):
            x_augmented[sample_count] = x_aug[0]
            y_augmented[sample_count] = y_aug[0]
            sample_count += 1
            break  # Only take one augmented version of each sample

    return x_augmented, y_augmented
```

The randomly augmented samples add to about 7% *[7870/ 2(55000)]* of the training data at maximum and reduces in proportion as the augmentation factor increases, thus maintaining fairly relative balance in the data.

- **Combine full and random augmentation strategies, apply transformations:**

```
def augment_data(x_train, y_train, val_size, augmentation_factor):
    # Ensure x_train has a channel dimension for ImageDataGenerator
    x_train = ensure_4d(x_train)

    # Define the data augmentation parameters
    datagen = ImageDataGenerator(
        rotation_range=15,  # Rotation up to ±15 degrees
        zoom_range=0.2,     # Zoom with 0.2 factor (0.8 to 1.2)
        width_shift_range=20/64,  # Horizontal shift up to 20 pixels
        height_shift_range=20/64,  # Vertical shift up to 20 pixels
        fill_mode='constant',
        cval=255
    )
    # Perform full augmentation
    full_aug_x_train, full_aug_y_train = full_augmentation(x_train, y_train, datagen,
augmentation_factor)
    # Perform random augmentation
    random_aug_x_train, random_aug_y_train = random_augmentation(x_train, y_train,
val_size, datagen, augmentation_factor)

    # Combine fully augmented and randomly augmented data
    aug_x_train = np.concatenate((full_aug_x_train, random_aug_x_train), axis=0)
    aug_y_train = np.concatenate((full_aug_y_train, random_aug_y_train), axis=0)

    # Print details about the augmented data
    print(f"Total augmented samples (Factor {augmentation_factor}):
{aug_x_train.shape[0]} [{full_aug_x_train.shape[0]} Fully augmented +
{random_aug_x_train.shape[0]} Randomly augmented samples]")
    print(f"Total augmented data shapes | X: {aug_x_train.shape}, Y:
{aug_y_train.shape}")

    return aug_x_train, aug_y_train
```

The above-mentioned strategies are combined here with one more of the variances as specified, applied:

- rotation (up to ±15 degrees),
- zoom (with 0.2 factor), and
- horizontal/vertical shift variations (up to 20 pixels).

Any discontinuities of characters at the edges due to transformations are filled with white as the augmenter tends to extend the image with the nearest colour (`fill_mode = 'constant', cval=255`) which can introduce additional strokes that tamper with character integrity. This is crucial in a supervised learning task where the characters need to be identifiable as per their labels.

4.1.2 Data Preparation:

- Prepare data for training

```
def prepare_data_for_training(x_train, y_train, x_val, y_val, x_test, y_test,
aug_x_train=None, aug_y_train=None):
    # Ensure all data has the same shape (4D)
    x_train = ensure_4d(x_train)
    x_val = ensure_4d(x_val)
    x_test = ensure_4d(x_test)
    if aug_x_train is not None:
        aug_x_train = ensure_4d(aug_x_train)

    # Combine original and augmented training data if augmented data is provided
    if aug_x_train is not None and aug_y_train is not None:
        combined_x_train = np.concatenate([x_train, aug_x_train], axis=0)
        combined_y_train = np.concatenate([y_train, aug_y_train], axis=0)

        # Shuffle the combined training data
        combined_x_train, combined_y_train = shuffle(combined_x_train,
combined_y_train, random_state=RANDOM_SEED)
    else:
        combined_x_train = x_train
        combined_y_train = y_train

    # Normalize the data
    combined_x_train, x_val, x_test = normalize_data(combined_x_train, x_val, x_test)

    # Encode the labels
    combined_y_train, y_val, y_test = encode_labels(combined_y_train, y_val, y_test)

    # Verify normalization
    print(f"Range of combined_x_train, x_val, x_test values:
{np.min(combined_x_train)}-{np.max(combined_x_train)}, {np.min(x_val)}-
{np.max(x_val)}, {np.min(x_test)}-{np.max(x_test)}")

    # Verify shapes and one-hot encoding of labels (samples, num_classes)
    print(f"Final shapes:")
    print(f"Training data   | X: {combined_x_train.shape}, Y:
{combined_y_train.shape}")
    print(f"Validation data | X: {x_val.shape},   Y: {y_val.shape}")
    print(f"Test data       | X: {x_test.shape},  Y: {y_test.shape}")

    return combined_x_train, combined_y_train, x_val, y_val, x_test, y_test
```

- Encapsulates data pre-processing steps and prepares data for training:
    - Ensures data dimensions are compatible for training with 1 channel dimension for CNNs using Tensorflow and 3 for pre-trained models such as VGG.
    - Combines original training data, augmented data and shuffles them to reduce chances of overfitting.

- Normalizes data and encodes labels.
- Prints final shapes and range of data for verification of normalization.

- Defining augmented data as 'None' is included to prepare data for evaluation where augmented data is not included, to objectively assess the model's performance on data samples that are real-world representations.

4.1.3 Visualization:

- **Display sample images with their labels.**

```
def view_sample_images(images, labels, num_rows, title):
    # Decode one-hot encoded labels to integers if necessary
    if len(labels.shape) > 1 and labels.shape[1] > 1:
        labels = np.argmax(labels, axis=1)
    num_samples = num_rows * 8
    plt.figure(figsize=(16, num_rows*2 + num_rows))
    for i in range(num_samples):
        # Select a random index
        idx = np.random.randint(0, len(images))
        plt.subplot(num_rows, 8, i + 1)
        # Display the image and use squeeze to handle single channel images
        plt.imshow(images[idx].squeeze(), cmap='gray')
        plt.title(f'{title}\nLabel: {labels[idx]}')
        plt.axis('off')
    plt.tight_layout()
    plt.suptitle(f"Sample {title} Images", fontsize=16)
    plt.subplots_adjust(top=0.85)
    plt.show()
```

This function allows any set of images to be visualised in the quantity of samples desired (by defining num_rows). Thus, augmented samples or training samples can be visualised at each stage of data processing for verification to ensure they are as required (i.e. no unusual deformations).

- **Plots the training and validation accuracy/loss over epochs to evaluate training:**

```
def plot_training_history(history):
    plt.figure(figsize=(12, 5))
    plt.subplot(121)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.subplot(122)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.tight_layout()
    plt.show()
```

4.2 Reproducing the uTHCD: Overall workflow

The Basic CNN model as established by the Benchmark was reproduced as follows, with a comparison of results at each stage (Details provided in following steps are provided by the Benchmark unless otherwise stated):

4.2.1 Basic Model with Early Stopping and Dropout:

a) ***Data Pre-processing***: Images were normalised and labels one-hot encoded.
b) ***Model architecture***: Input layer accepting 64x64 sized images (with a 4-dimensional shape including a colour channel for grayscale images), followed by two convolutional layers with 64 filters and 3x3 kernels, each followed by 2x2 max pooling, and then two fully connected layers with 1024 and 512 neurons respectively.
c) ***Hyperparameters*** and ***training parameters*** are as mentioned in Table.4, based on optimal findings in the Benchmark.
d) ***Regularization*** was applied with dropout rates of 0.5 for dense layers, 0.1 for the first convolutional layer, and 0.05 for the second convolutional layer.
e) ***Early stopping*** based on validation loss is implemented. No further parameters were defined.
f) ***Model was trained*** for a maximum of 200 epochs, without any augmented data (only training set consisting of 55000 samples and validation set of 7870 samples).

4.2.2 Basic Model with Dropout, Early Stopping and Augmentation

a) ***Data Augmentation***: Data was augmented as mentioned in 4.1.1.V, with a factor of 3, producing a total of 243610 augmented samples for training, leaving 7870 un-augmented samples for validation and 28080 samples for testing.
b) Remaining steps were maintained as mentioned in 1a) – d).
c) ***Early stopping*** is implemented with an arbitrary patience of 20 as this value was not specified, to prevent prematurely stopping training before the model has a chance to learn the data sufficiently. Best weights are also restored from the epoch with the best validation loss.
d) ***Model is trained with augmented data***.

4.2.3 Evaluation

Conducted on un-augmented training data, validation and test data for Training, Validation and Test accuracy results to ensure they are measured on true representations of data.

4.3 Example output: Appendix A

4.4 Enhanced implementation

4.4.1 Modified Data Augmentation

```
def apply_morphological_ops(image, **kwargs):
    # Create an elliptical kernel for morphological operations
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    if random.randint(1, 5) == 1: # 20% chance of applying erosion
        # Erode the image 1-2 times
        image = cv2.erode(image, kernel, iterations=random.randint(1, 2))
    if random.randint(1, 6) == 1: # ~16.7% chance of applying dilation
        # Dilate the image once
        image = cv2.dilate(image, kernel, iterations=random.randint(1, 1))
    return image

def get_augmentation_pipeline():
    border_value = 255 # Define white color for borders (255 for grayscale images)
    return A.Compose([
        # Randomly scale the image up or down by up to 10%
        A.RandomScale(scale_limit=0.1, p=0.1),
        # Rotate the image by up to 15 degrees
        A.Rotate(limit=15, p=0.3, border_mode=cv2.BORDER_CONSTANT,
value=border_value),
        # Apply affine transformations (horizontal/vertical translation)
        A.Affine(translate_percent={'x': (-0.1575, 0.1575), 'y': (-0.1575, 0.1575)},
p=0.1,
                 mode=cv2.BORDER_CONSTANT, cval=border_value),
        # Combination of shift, scale, and rotate
        A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.1, rotate_limit=15,
p=0.3,
                           border_mode=cv2.BORDER_CONSTANT, value=border_value),
        # Apply custom morphological operations
        A.Lambda(image=apply_morphological_ops, p=0.5),
        # Add Gaussian noise
        A.GaussNoise(var_limit=(10.0, 50.0), p=0.3),
        # Adjust brightness and contrast
        A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.3),
        # Apply blur
        A.Blur(blur_limit=3, p=0.2),
        # Ensure final image size is 64x64
        A.Resize(64, 64, always_apply=True)
    ], p=1)  # Apply the entire pipeline with 100% probability
```

This image augmentation pipeline uses the Albumentations library [24] as it offers more flexibility in operations involved which consist of the following changes from the pipeline reproduced from the benchmark as stated in (4.1.1.V):

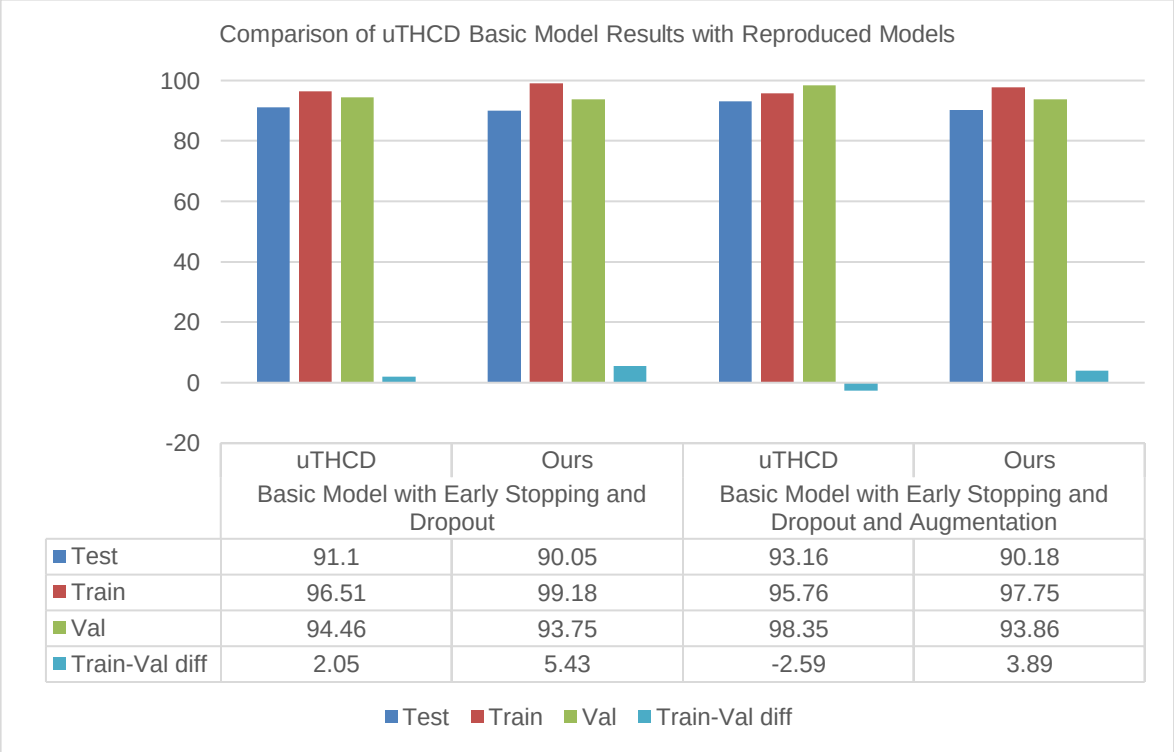
- **Rotate, shift and zoom** as stated in the benchmark are **scaled down** to: ±15 degrees, up to about 10 pixels, and a factor of 0.1 respectively (from ±20, 20, 0.2),

- **Brightness and Contrast** variations - reflect different lighting conditions of characters viewed on paper.
- **Shear** (Blur) - adds transparency.
- **Gaussian noise** - fills the empty spaces of the character images simulating real-world conditions.
- **Morphological** alterations such as erosion and dilation - change the thickness of strokes [23] using the OpenCV Library [25].

(1220 words)

5. EVALUATION

5.1 Reproduced uTHCD Benchmark:



	uTHCD (Basic Model with Early Stopping and Dropout)	Ours (Basic Model with Early Stopping and Dropout)	uTHCD (Basic Model with Early Stopping and Dropout and Augmentation)	Ours (Basic Model with Early Stopping and Dropout and Augmentation)
Test	91.1	90.05	93.16	90.18
Train	96.51	99.18	95.76	97.75
Val	94.46	93.75	98.35	93.86
Train-Val diff	2.05	5.43	-2.59	3.89

FIGURE 8. Comparison of Reproduced Basic Model with Early Stopping, Dropout, /and Augmentation results.

- **Without Augmentation**: uTHCD slightly outperforms in Test and Validation accuracy, while our model has higher Train accuracy and thus greater Train-Validation difference, indicating more overfitting.
- **With Augmentation**: Test and Validation accuracy of uTHCD have improved with a significant decrease in Train accuracy, bridging variance in the model while our model has not improved much with Train accuracy still much higher than Test and Validation accuracy (Positive variance compared to negative variance in uTHCD (Fig.8)).

Therefore, uTHCD shows better generalization, demonstrating that augmentation improved uTHCD's performance more significantly than ours. This could be because

augmentation techniques have not been exactly reproduced in implementation (since not stated explicitly), and that an Early Stopping patience of 20 may be too high, causing overtraining and thus overfitting. Patience refers to the number of epochs with no improvement (in this case validation loss) after which training will be stopped. Even though the results of our model differ considerably, this was used as a baseline to assess the effectiveness of further techniques based on our implementation since direct comparison with the Benchmark is not possible without full details on execution.

5.2 Optimizing the Benchmark

5.2.1 Modified Data Augmentation

TABLE 9. Augmentation techniques

Original Samples	Benchmark Augmentation	Scaled down	Brightness & Shear added	Gaussian noise added	Morphological alterations added	Black & White Noise added
0	1	2	3	4	5	6

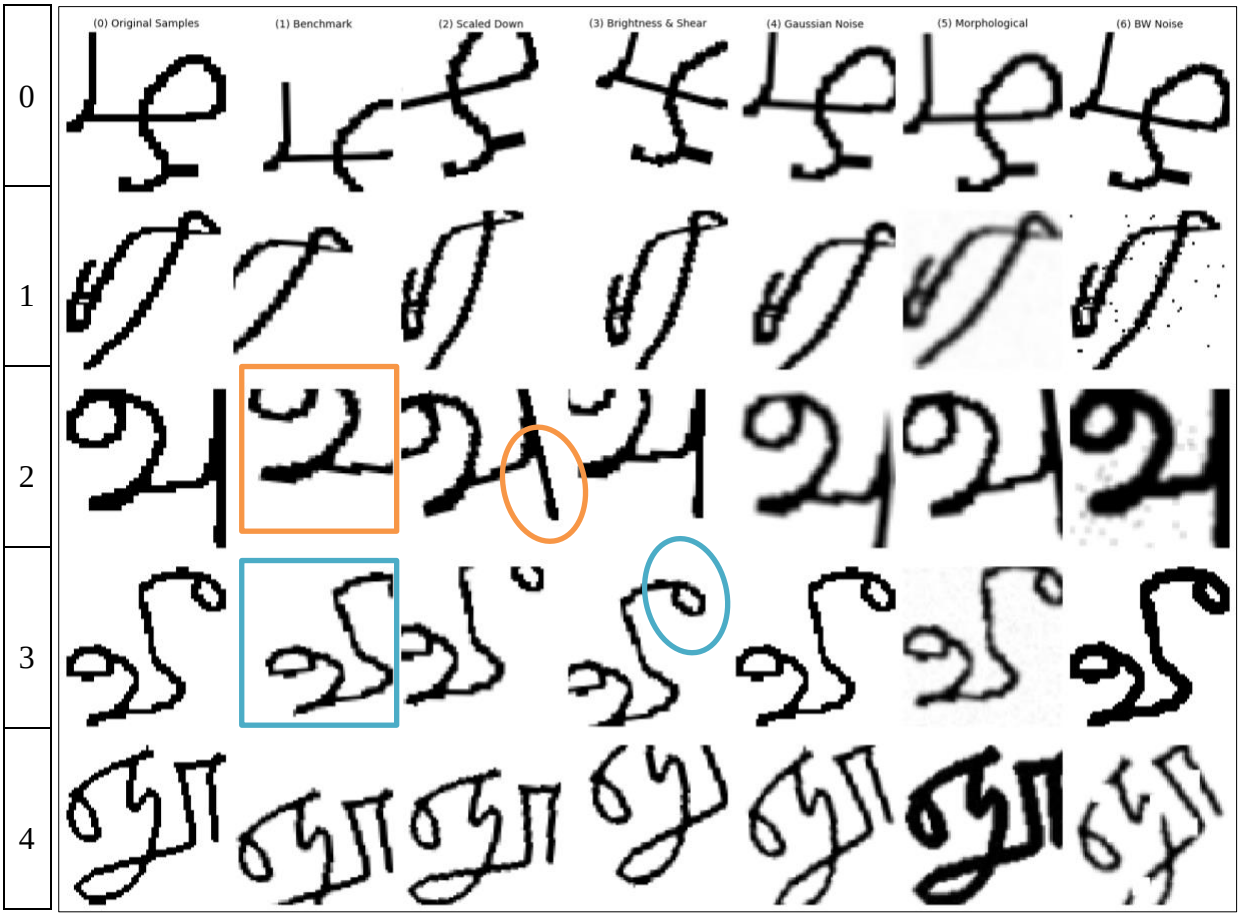


FIGURE 9. Sample images produced by varying augmentation pipelines.

In Image data augmentation, especially for text recognition, it is essential that augmented data preserve their labels [9]. This means that the augmented image is still identifiable by its label, therefore retaining its representativeness, which is especially crucial in the

recognition of Tamil characters that are distinguishable by minutely different modifiers. As seen in Fig.9, the original augmentation pipeline established by the benchmark (Column 1), due to extensive translation (up to 20 pixels), have removed some characters' modifiers from view. For example, as seen by characters in (1,2) and (1,3) in Fig.9 as compared to their original samples, ground truths are **வு** and **வீ**, that can be misinterpreted as **வ**/ **வி** respectively, due to the absence of their distinct modifiers (circled in Fig.9). This has been reiterated as a challenge in the uTHCD Benchmark as well which states "Misclassification happens between [characters] which differ in their overall shape with minor changes (e.g. elongation of a stroke, presence of tiny modifier, etc.)" [4]. Therefore, translations, zoom and rotation being augmenters that can obstruct these components, were scaled down in the modified pipeline.

5.2.2 Effect of varying Augmentation Techniques

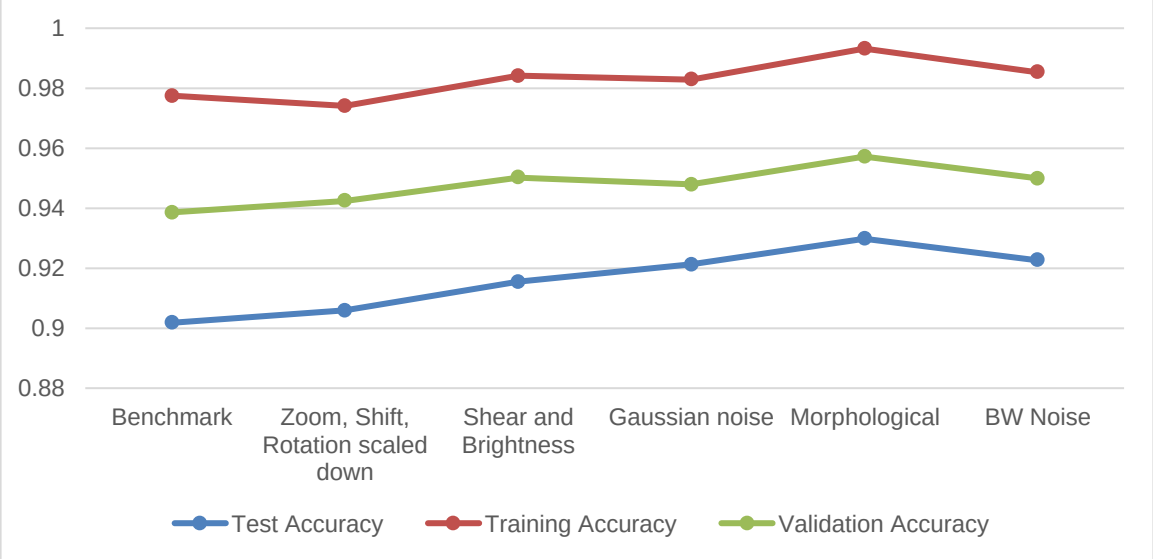


FIGURE 10. Results with addition of varying augmentation techniques

As seen by an increase in validation and test accuracy, the scaled down augmentation has been effective in generalization of the model. Modifying the augmentation pipeline further with incremental changes shown in Table.9 up till morphological augmentation have also steadily increased validation accuracy while slightly reducing overfitting with a smaller difference in training and validation accuracy. This shows effective regularization and generalization has been achieved by introducing greater variability in samples using these techniques, that have allowed the model to capture robust features and details. Introducing Black/White noise however has not been effective possibly as seen in Fig.8 (1,6), addition of black pixel noise could confuse the classifier in appearing like diacritical marks such as dots above characters (e.g. **க**/ **க்**), and white pixel noise as in (6,4) truncating strokes that result in an appearance of additional or discontinued strokes. Thus, it was not included in further experiments.

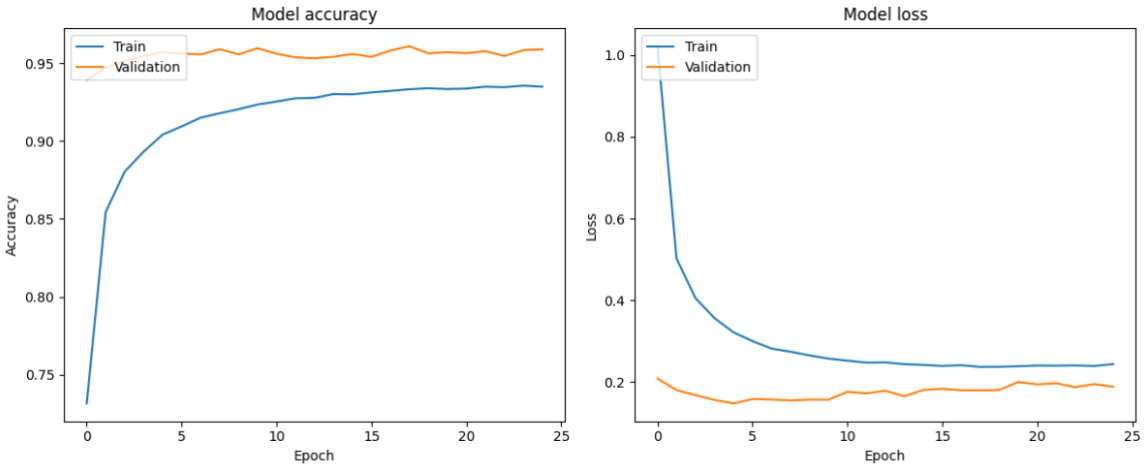


FIGURE 11. Model Accuracy and Loss Over Epochs for Modified Augmentation Pipeline

The model shows promising performance with accuracies of 92.98% (test), 99.09% (train), and 95.69% (validation). The notably high training accuracy likely results from evaluating on un-augmented training data while the model was trained on augmented data, rather than indicating overfitting. This suggests the model has learned robust features from augmented data that generalize well to original samples. The learning curves show validation accuracy consistently trailing training accuracy, implying room for the model to capture more nuanced patterns from the augmented training set.

5.2.3 Batch normalization

Next, we introduced batch normalization layers after every convolutional and dense layer, preceding activation and dropout layers.

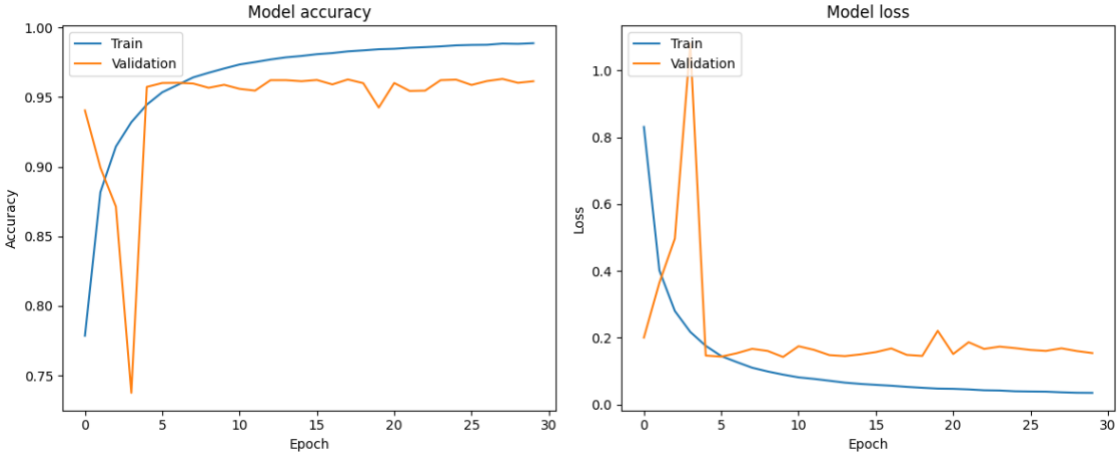


FIGURE 12. Model Accuracy and Loss Over Epochs after Batch Norm in every layer

This strategic enhancement aimed to stabilize training, enable higher learning rates, and improve gradient flow [26], ultimately leading to better utilization of augmented data and narrowing the train-validation gap.

The modification yielded improved results with test, train, and validation accuracies of 93.53%, 99.88%, and 95.9% respectively. The learning curves (Fig.12) demonstrate a rapid increase in training accuracy, reaching and maintaining a high level throughout the

epochs. Validation accuracy stabilizes earlier and shows more consistency, while both training and validation loss curves exhibit a faster initial decrease and a more stable learning process.

Initial volatility observed in validation is a common phenomenon when introducing batch normalization, stemming from factors such as varying mini-batch statistics, higher learning rates, adaptive layer dynamics, increased layer interdependence, and sensitivity to initialization. These factors typically lead to faster convergence and better generalization as training stabilizes.

While these improvements affirm the positive impact of batch normalization, the substantial gap between training and validation accuracies (99.88% vs 95.9%) suggests potential overfitting. To address this, we employed a more incremental approach by applying batch normalization only to the first convolutional layer, reducing the early stopping patience, and implementing a learning rate scheduler. This adjustment aims to balance input distribution normalization with the model's capacity to learn complex features, while mitigating overfitting and capturing more subtle patterns in the data.

5.2.4 Fine-tuning with Batch Normalization, Early Stopping and Learning Rate Schedule

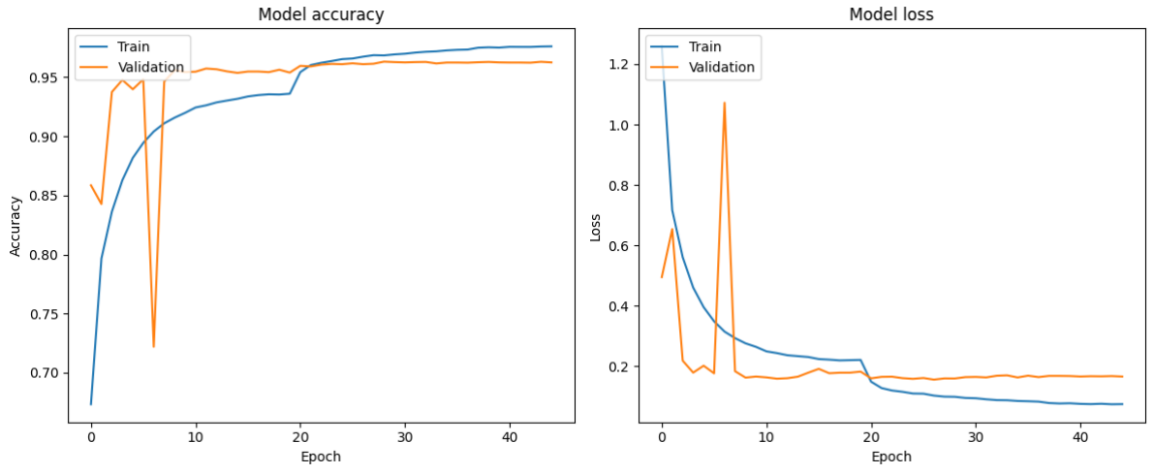


FIGURE 13. Model Accuracy and Loss Over Epochs with Batch Norm in first convolutional layer only and Learning Rate Schedule

Along with applying batch normalization only to the first convolutional layer, Early Stopping patience was reduced to 15 (from 20), and a learning rate scheduler (ReduceLROnPlateau [27] with factor 0.1 and patience 8) was employed. These adjustments yielded improved results with test, train, and validation accuracies of 93.92%, 99.94%, and 96.1% respectively.

The learning curves reveal several key insights:

1. Training accuracy shows a smooth, steady increase, reaching and maintaining a very high level (99.94%), indicating effective learning on the training set.

2. Validation accuracy, while improved overall, displays noticeable fluctuations throughout training. These fluctuations suggest that the model is sensitive to the specific batches used in each validation step, possibly due to the diversity introduced by data augmentation.
3. The gap between training and validation accuracies has narrowed compared to previous iterations, indicating improved generalization.
4. Both training and validation loss curves show a consistent downward trend, with the validation loss stabilizing in later epochs. This suggests that the learning rate scheduler is effectively adjusting the learning rate to fine-tune the model's performance.
5. The model trains for a longer period (over 40 epochs) compared to earlier versions, likely due to the learning rate scheduler allowing for more gradual improvements.

The fluctuating validation accuracy, while initially concerning, may be a positive sign. It suggests that the model is continually adapting to different aspects of the augmented data, preventing it from overfitting to specific patterns. The overall upward trend in validation accuracy, despite these fluctuations, indicates that the model is gradually improving its generalization capabilities.

The combination of targeted batch normalization, tuned early stopping, and adaptive learning rate has clearly contributed to a more robust model. The improved test accuracy (93.92%) demonstrates enhanced generalization to unseen data, while the high training accuracy (99.94%) shows the model's capacity to capture complex features from the augmented dataset.

These results suggest that our iterative approach to fine-tuning the model architecture and training process is yielding positive outcomes, striking a balance between model stability and the ability to learn nuanced features from the augmented Tamil character dataset.

5.2.5 Final Results and Evaluation

Model	Test Accuracy (%)	Train Accuracy (%)	Val Accuracy (%)	True Positive Rate (Sensitivity)	F1-Score
uTHCD Final Benchmark	*93.16*	*95.76*	*98.35*	*0.9315*	*0.931*
Enhanced Model Implementation (Ours)					
With Modified Augmentation	92.98	99.09	95.69	0.9298	0.9296

With Batch Normalization and Learning Rate Schedule	**93.92**	99.94	96.1	**0.9392**	**0.9392**

Overall, although reproducing the benchmark did not yield on-par results, the process revealed important insights on the how the data performs and key elements of data augmentation that affect model performance. It also provides an avenue to further discover how the strategies and variations were implemented to produce those results.

The **batch normalization and learning rate scheduling** yielded the best test accuracy, indicating improved generalization and better performance on unseen data, surpassing the uTHCD benchmark.

The model with **batch normalization and learning rate scheduling** again performs best with an F1-Score of **0.9392**, indicating a better overall balance between precision and recall, and confirming that it has achieved a better trade-off between false positives and false negatives compared to the other models.

(1490 words)

6. CONCLUSION

The focus on reproducing the uTHCD benchmark and improving it by enhancing learning capacity and generalization has been instrumental in optimizing the performance of our model. By incorporating techniques such as modified data augmentation, batch normalization, early stopping, and learning rate scheduling, incrementally we have demonstrated how these strategies can reduce overfitting and enhance generalization without adding complexity to the model. The resulting improvements in test, train, and validation accuracies, as well as true positive rates, highlight the robustness of the chosen approach. This iterative process, which optimizes resource efficiency, sets a strong foundation for future experimentation. The methods explored here could be applied to larger models or transfer learning tasks, enabling the study of more advanced architectures while maintaining efficiency and scalability. This work paves the way for further exploration of these techniques in handling more complex datasets and tasks in Tamil handwritten character recognition.

The clear dearth of standardised datasets in Tamil has short-changed the number of studies and experimentation in this field that could reveal a lot more insights on various techniques that apply specifically to Tamil characters. Therefore, there is an imminent need in utilizing the uTHCD as it captures more complexity of this data which has been evident in experimenting with various techniques so far. For example, in experimenting with model architectures, it was clear that convolutional layers with a filter size below 64 were not able to capture details as robustly, impeding model performance, illustrating the richness of the data compared to the HPL dataset, and the nuanced approach to improving learning capacity required for this data.

This points to more advanced techniques such as transfer learning, residual networks or long-short-term excitation networks, etc. or using larger state-of-the-art models that will

invariably require greater resources. Thus, it is worth considering optimisation techniques that maintain resource efficiency to maintain deploy ability and applicability. This aspect was challenging considering the difficulty in finding relevant studies that demonstrate these techniques on comprehensive data similar to the uTHCD. There is a lot more scope in furthering experimentation in aspects such as data augmentation using General Adversarial Networks (GANs), or strategies such as Global Average Pooling, or structural modifications for hierarchical learning, kernel size variations to reduce parameters and parallel networks to maintain receptivity etc, that have shown good progress in other languages and datasets. I hope that this study provides a good foundation to further delve into these techniques.

7. APPENDICES

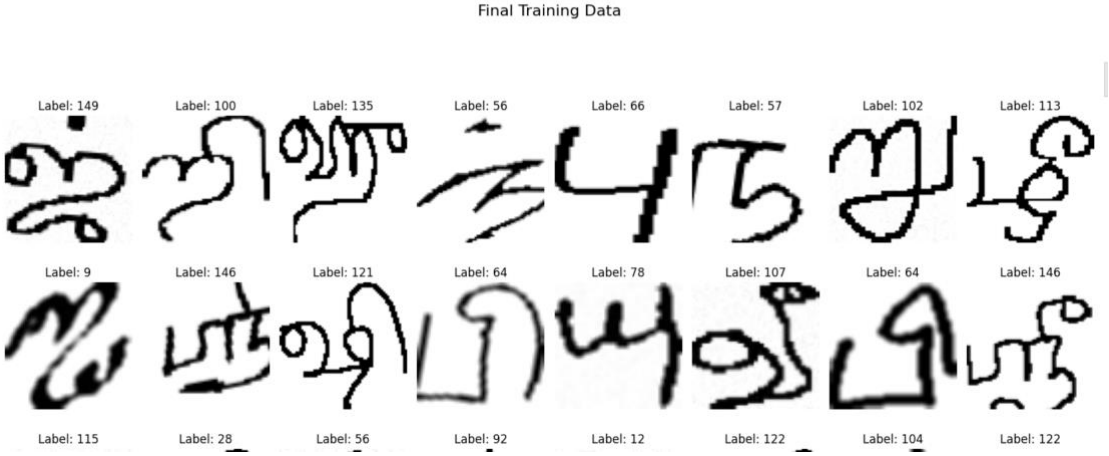
A.

Data Preparation

```
[ ] # Define augmentation factor
    aug_factor = 3

[ ] # Load data
    (x_train, y_train), (x_val, y_val), (x_test, y_test), val_size = du.load_data('hdf5_uTHCD_compressed.h5')

Data loaded ================================
Extracted data (X) and labels (Y) shapes:
Train Set| X: (62870, 64, 64), Y: (62870,)
Test Set | X: (28080, 64, 64),  Y: (28080,)
Train-Validation Split ====================
Training data    | X: (55000, 64, 64), Y: (55000,)
Validation data  | X: (7870, 64, 64),   Y: (7870,)

[ ] # Augment data (optional)
    if (aug_factor > 0):
      aug_x_train, aug_y_train = du.augment_data(x_train, y_train, val_size, augmentation_factor=aug_factor)

Full Augmentation: 100%|| 165000/165000 [02:20<00:00, 1177.89it/s]
Random Augmentation: 100%|| 23610/23610 [00:20<00:00, 1175.57it/s]
==> Total augmented samples (Factor 3): 188610
Total augmented data shapes | X: (188610, 64, 64), Y: (188610,)

[ ] # Prepare data for training
    X_train, Y_train, X_val, Y_val, X_test, Y_test = du.prepare_data_for_training(x_train, y_train, x_val, y_val, x_test, y_test, aug_x_train, aug_y_train)

Number of unique classes: 156
Range of combined_x_train, x_val, x_test values: 0.0-1.0, 0.0-1.0, 0.0-1.0
Final shapes:
Training data   | X: (243610, 64, 64, 1), Y: (243610, 156)
Validation data | X: (7870, 64, 64, 1),   Y: (7870, 156)
Test data       | X: (28080, 64, 64, 1),  Y: (28080, 156)
```

Visualise data

```
[ ] # View sample images
    du.view_sample_images(X_train, Y_train, num_rows=3, title='Final Training Data')
```



Evaluate

```
# Prepare data for evaluation (without augmented data)
X_train_final, Y_train_final, X_val_final, Y_val_final, X_test_final, Y_test_final = du.prepare_data_for_training(x_train, y_train, x_val, y_val, x_test, y_test,
                                                                                                                   aug_x_train=None, aug_y_train=None)
```

```
Number of unique classes: 156
Range of combined_x_train, x_val, x_test values: 0.0-1.0, 0.0-1.0, 0.0-1.0
Final shapes:
Training data   | X: (55000, 64, 64, 1), Y: (55000, 156)
Validation data | X: (7870, 64, 64, 1),   Y: (7870, 156)
Test data       | X: (28080, 64, 64, 1),  Y: (28080, 156)
```

```
# Evaluate
train_loss, train_accuracy = model.evaluate(X_train_final, Y_train_final)
val_loss, val_accuracy = model.evaluate(X_val_final, Y_val_final)
test_loss, test_accuracy = model.evaluate(X_test_final, Y_test_final)

print("\nEVALUATION RESULTS")
print(f"Train accuracy: {train_accuracy:.4f}")
print(f"Validation accuracy: {val_accuracy:.4f}")
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9994 - loss: 0.0054
246/246 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9616 - loss: 0.1468
878/878 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9389 - loss: 0.2376

EVALUATION RESULTS
Train accuracy: 0.9994
Validation accuracy: 0.9610
Test accuracy: 0.9392
```

```
# Make predictions
y_pred = model.predict(X_test_final)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(Y_test_final, axis=1)  # Assuming y_test is one-hot encoded

# Calculate precision, recall, and F1-score
precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred_classes, average='weighted')

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

# Generate a detailed classification report
# report = classification_report(y_true, y_pred_classes)
# print("\nClassification Report:")
# print(report)
```

```
878/878 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
Precision: 0.9402
Recall: 0.9392
F1-score: 0.9392
```

Update results

```
# Log final results
df.loc[['Precision'], model_name] = np.round(float(precision), decimals=4)
df.loc[['Recall'], model_name] = np.round(float(recall), decimals=4)
df.loc[['F1-score'], model_name] = np.round(float(f1), decimals=4)
df.loc[['Training Accuracy'], model_name] = np.round(float(train_accuracy), decimals=4)
df.loc[['Validation Accuracy'], model_name] = np.round(float(val_accuracy), decimals=4)
df.loc[['Test Accuracy'], model_name] = np.round(float(test_accuracy), decimals=4)


df.to_csv(tuning_results_file_path)
print(f"Original file '{tuning_results_file_path}' has been overwritten with the updated data.")
```

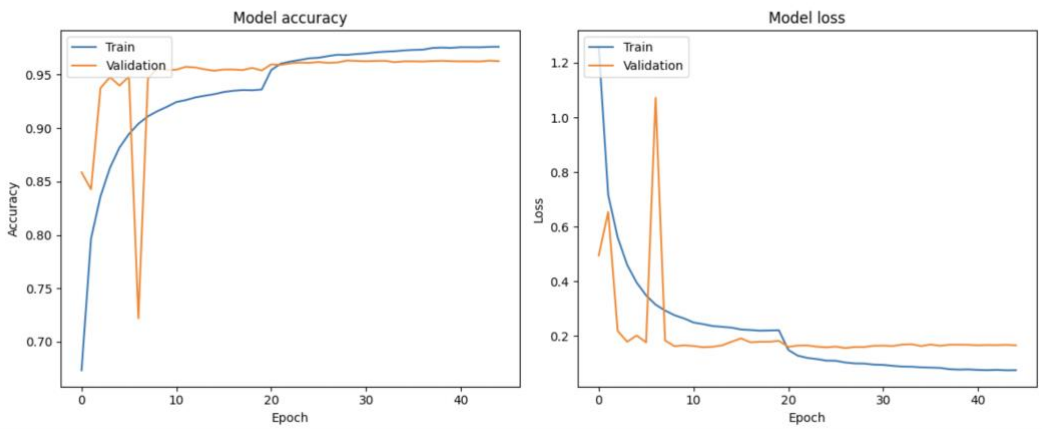
```
Original file '/content/drive/MyDrive/THCR/tuning_results.csv' has been overwritten with the updated data.
```

```
# Save the entire model (architecture, weights, and optimizer state)
model.save(model_save_path)
print(f"Model saved to {model_save_path}")
```

```
Model saved to /content/drive/MyDrive/THCR/saved_models/Model0.keras
```

Plot Train and Validation curves

```
# Call the function to plot
du.plot_training_history(history)
```



B.

```
import numpy as np
import h5py
with h5py.File('../uTHCD_compressed.h5','r') as hdf:
 base_items = list(hdf.items())
 print('Items in the Base Directory:', base_items)
 G1 = hdf.get('Train Data')
 G1_items = list(G1.items())
 print('\n Items in Group 1:', G1_items)
 G2 = hdf.get('Test Data')
  G2_items = list(G2.items())
 print('Items in Group 2:', G2_items)
x_train = np.array(G1.get('x_train'))
y_train = np.array(G1.get('y_train'))
x_test = np.array(G2.get('x_test'))
y_test = np.array(G2.get('y_test'))
x_val = x_train[-7870:, :, :]
y_val = y_train[-7870:]
x_train = x_train[:-7870, :, :]
y_train = y_train[:-7870]
```

8. REFERENCES

I. Journal Articles

18. Laith Alzubaidi, Jinshuai Bai, Aiman Al-Sabaawi, Jose Santamaría, A. S. Albahri, Bashar Sami Nayyef Al-dabbagh, Mohammed A. Fadhel, Mohamed Manoufali, Jinglan Zhang, Ali H. Al-Timemy, Ye Duan, Amjed Abdullah, Laith Farhan, Yi Lu, Ashish Gupta, Felix Albu, Amin Abbosh, and Yuantong Gu. 2023. A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications. *J Big Data* 10, 1 (April 2023), 46. https://doi.org/10.1186/s40537-023-00727-2
19. Raashid Hussain, Ahsen Raza, Imran Siddiqi, Khurram Khurshid, and Chawki Djeddi. 2015. A comprehensive survey of handwritten document benchmarks: structure, usage and evaluation. *J Image Video Proc.* 2015, 1 (December 2015), 46. https://doi.org/10.1186/s13640-015-0102-5
20. B.R. Kavitha and C. Srimathi. 2022. Benchmarking on offline Handwritten Tamil Character Recognition using convolutional neural networks. *Journal of King Saud University - Computer and Information Sciences* 34, 4 (April 2022), 1183–1190. https://doi.org/10.1016/j.jksuci.2019.06.004
21. Noushath Shaffi and Faizal Hajamohideen. 2021. uTHCD: A New Benchmarking for Tamil Handwritten OCR. *IEEE Access* 9, (2021), 101469–101493. https://doi.org/10.1109/ACCESS.2021.3096823
22. Vinotheni C. and S. Lakshmana Pandian. 2024. Fast Recurrent Neural Network with Bi-LSTM for Handwritten Tamil Text Segmentation in NLP. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* 23, 5 (May 2024), 1–20. https://doi.org/10.1145/3643808
23. Husam Ahmad Alhamad, Mohammad Shehab, Mohd Khaled Y. Shambour, Muhannad A. Abu-Hashem, Ala Abuthawabeh, Hussain Al-Aqrabi, Mohammad Sh. Daoud, and Fatima B. Shannaq. 2024. Handwritten Recognition Techniques: A Comprehensive Review. *Symmetry* 16, 6 (June 2024), 681. https://doi.org/10.3390/sym16060681
24. C. Vinotheni and S. Lakshmana Pandian. 2023. End-To-End Deep-Learning-Based Tamil Handwritten Document Recognition and Classification Model. *IEEE Access* 11, (2023), 43195–43204. https://doi.org/10.1109/ACCESS.2023.3270895
25. Jayasree Ravi. 2024. Handwritten alphabet classification in Tamil language using convolution neural network. *International Journal of Cognitive Computing in Engineering* 5, (2024), 132–139. https://doi.org/10.1016/j.ijcce.2024.03.001
26. Connor Shorten and Taghi M. Khoshgoftaar. 2019. A survey on Image Data Augmentation for Deep Learning. *J Big Data* 6, 1 (December 2019), 60. https://doi.org/10.1186/s40537-019-0197-0

II. Conference Papers

22. C. S. Arjun, N. Shobha Rani, and Akshatha Prabhu. 2024. Handwritten Character Recognition for South Indian Languages Using Deep Learning. In *Proceedings of Fifth International Conference on Computer and Communication Technologies*, B. Rama Devi, Kishore Kumar, M. Raju, K. Srujan Raju and Mathini Sellathurai (eds.).

Springer Nature Singapore, Singapore, 49–63. https://doi.org/10.1007/978-981-99-9704-6_5

23. B M Vinjit, Mohit Kumar Bhojak, Sujit Kumar, and Gitanjali Chalak. 2020. A Review on Handwritten Character Recognition Methods and Techniques. In *2020 International Conference on Communication and Signal Processing (ICCSP)*, July 2020. IEEE, Chennai, India, 1224–1228. https://doi.org/10.1109/ICCSP48568.2020.9182129
24. L Sherly Puspha Annabel, Karthik Raja Rajan, and Lakshman Siva. 2023. Machine Learning-Based Tamil Handwritten Word Recognition. In *2023 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI)*, December 21, 2023. IEEE, Chennai, India, 1–7. https://doi.org/10.1109/ICDSAAI59313.2023.10452511
25. Saikrishna Devendiran, Jyothiratnam, Prema Nedungadi, and Raghu Raman. 2024. Handwritten Text Recognition using VGG19 and HOG Feature Descriptors. In *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, March 14, 2024. IEEE, Gwalior, India, 1–4. https://doi.org/10.1109/IATMSI60426.2024.10502872
26. R Babitha Lincy, J Jency Rubia, C Sherin Shibi, and N Kanimozhi. 2023. Handwritten Character Recognition System using Deep Learning Models for Tamil Language. In *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, June 14, 2023. IEEE, Coimbatore, India, 396–400. https://doi.org/10.1109/ICSCSS57650.2023.10169854
27. Gnanasivam P., Bharath G., Karthikeyan V., and Dhivya V. 2021. Handwritten Tamil Character Recognition using Convolutional Neural Network. In *2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, March 25, 2021. IEEE, Chennai, India, 84–88. https://doi.org/10.1109/WiSPNET51692.2021.9419451
28. S. K. Jayanthi and D. Rajalakshmi. 2011. Writer identification for offline Tamil handwriting based on gray-level co-occurrence matrices. In *2011 Third International Conference on Advanced Computing*, December 2011. 187–192. https://doi.org/10.1109/ICoAC.2011.6165173
29. Prashanth Vijayaraghavan and Misha Sra. 2014. Handwritten Tamil recognition using a convolutional neural network. In *2018 international conference on information, communication, engineering and technology (ICICET)*, 2014. 1–4.

III. Book Section

18. Vinotheni, S. Lakshmana Pandian, and G. Lakshmi. 2021. Modified Convolutional Neural Network of Tamil Character Recognition. In *Advances in Distributed Computing and Machine Learning*, Asis Kumar Tripathy, Mahasweta Sarkar, Jyoti Prakash Sahoo, Kuan-Ching Li and Suchismita Chinara (eds.). Springer Singapore, Singapore, 469–480. https://doi.org/10.1007/978-981-15-4218-3_46

IV. Online Resources

Datasets:

19. HPL Isolated Handwritten Tamil Character Dataset. Retrieved September 2, 2024 from https://lipitk.sourceforge.net/datasets/tamilchardata.htm
20. uTHCD-Unconstrained Tamil Handwritten Database. Retrieved September 2, 2024 from https://www.kaggle.com/datasets/faizalhajamohideen/uthcdtamil-handwritten-database

Online articles/ documentation:

21. Anil Chandra Naidu Matcha. 2022. Handwriting Recognition with ML (An In-Depth Guide). Nanonets AI and Intelligent Document Processing Blog. (June 20, 2022). Retrieved June 17, 2024 from https://nanonets.com/blog/handwritten-character-recognition/
22. Ganeshmm. 2022. TamilNet: Handwritten Tamil Character Recognition system using a convolutional neural network. GitHub repository. Retrieved June 17, 2024 from https://github.com/ganeshmm/TamilNet
23. Toon Beerten. 2023. Effective Data Augmentation for OCR. *Medium*. Retrieved September 5, 2024 from https://towardsdatascience.com/effective-data-augmentation-for-ocr-8013080aa9fa
24. Albumentations: Efficient Image Augmentation Library for Machine Learning. Retrieved September 6, 2024 from https://albumentations.ai
25. OpenCV: Morphological Transformations. Retrieved September 6, 2024 from https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
26. Jason Brownlee. 2019. A gentle introduction to batch normalization for deep neural networks. *MachineLearningMastery.com*. Retrieved from https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/
27. Keras Team. Keras documentation: ReduceLROnPlateau. Retrieved from https://keras.io/api/callbacks/reduce_lr_on_plateau/
28. Keras Team. Keras documentation: KerasTuner. Retrieved from https://keras.io/keras_tuner/