University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Character-Level Urdu OCR for Disconnected Text

Student Name and SRN:

Muhammad Moiz Butt – 23038311

Github Link: https://github.com/moizbut

Supervisor: Hyungrok Kim

Date Submitted: 29th April 2025

Word Count: 4880

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science **in Data Science** at the University of Hertfordshire.

I have read the detailed guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6)

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Muhammad Moiz Butt

Student Name signature: Moiz Butt

Student SRN number: 23038311

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Abstract

This project presents an OCR system tailored for Urdu in the Nastalik typography style, addressing the challenges of recognizing both individual characters and short character sequences. Two deep learning models were developed using TensorFlow's Keras API: one for single-character recognition on 128x128 grayscale images, and another for sequence prediction (2–4 characters) on 256x256 grayscale images. A synthetic dataset of 30,000 sequence images was generated with augmentations like brightness and contrast adjustments, while the single-character model utilized a dataset of 40,000 images. Both models were trained with the Adam optimizer, employing sparse cross entropy loss for single-character classification and a custom multi-label loss for sequence prediction, over 25 epochs each. The training process incorporated robust data pipelines, augmentation techniques, and callbacks to ensure stability and prevent overfitting. Evaluation on test sets yielded a high accuracy of 98.58% for single-character and 84% for sequence recognition model. This work highlights the potential of deep learning for Urdu OCR, providing a scalable framework for multilingual text recognition applications.

Table of Contents

1.Introduction

Urdu, used by over 100 million speakers mainly in Pakistan and India, carries deep significance, with a vast repository of literature, historical documents, and modern media content that remains largely inaccessible in digital form. Optical Character Recognition (OCR) offers a transformative solution to digitize Urdu text, enabling its preservation, searchability, and accessibility for educational, archival, and technological applications, such as digital libraries and automated translation systems.

However, the task of developing an effective Urdu OCR system is complex due to the language's unique characteristics, which differ significantly from widely studied scripts like Latin or Arabic. The goal is to build a robust Urdu OCR system that leverages deep learning techniques to accurately recognize individual characters, focusing on overcoming specific issues like glyph similarity—where characters like "ب" and "پ" differ only by subtle diacritics—and the limitations of synthetic training datasets.

Through this work, I seek to achieve high recognition accuracy while ensuring the system's applicability to real-world scenarios, such as digitizing printed books or processing handwritten notes. This report details my methodology, including the use of a Deep Learning model, presents evaluation of the model's using metrics like confusion matrices (which showed 98.04%–98.49% accuracy for specific characters), and discusses strategies for future improvements to enhance the system's robustness and practical utility.

2.Review of Literature

Urdu (OCR) is essential for digitizing Urdu text, a language shared by the a large population of South Asia. The Nastaleeq script, commonly used for Urdu, is cursive, context-sensitive, and ligature-based, presenting unique challenges for OCR systems compared to Latin or Arabic scripts. My project focuses on Urdu character recognition using deep learning, and this literature review examines three studies to contextualize my work, highlighting their methodologies, findings, and limitations.

A 2014 study explored the recognition of Urdu-like cursive scripts, with a focus on Nastaleeq, using a combination of statistical and structural features (Naz et al., 2014). The authors proposed a segmentation-based approach, where Urdu text images were preprocessed to extract ligatures, followed by feature extraction using shape descriptors and Hidden Markov Models (HMMs) for classification. They achieved a recognition accuracy of 92% on a dataset of printed Nastaleeq text, demonstrating the effectiveness of their method. However, the study highlighted challenges with segmentation, noting, "The cursive nature of Nastaleeq script makes accurate segmentation difficult, especially for low-quality images" (Naz et al., 2014, p. 1235). This finding is relevant to my project, where glyph similarity (e.g., between "ب" and "پ"(led to misclassifications in my confusion matrices (98.04%–98.49% accuracy), suggesting that segmentation issues may contribute to errors, even though I used a segmentation-free deep learning approach.

A 2023 study introduced the Urdu Printed Text (UTRSet-Real) dataset, a comprehensive resource for Urdu OCR, and proposed a benchmark for Urdu text recognition (Rahman et al., 2023). The dataset includes 11,000 text images covering printed, and scene text in both Nastaleeq and Naskh scripts, addressing the scarcity of diverse Urdu datasets.

The authors evaluated several deep learning models, with a transformer-based model on printed Nastaleeq text. This dataset is directly relevant to my project, as my synthetic dataset may lack the diversity needed for real-world robustness, contributing to my overall validation accuracy of 92% being lower than character-specific accuracies. The study's use of CNNs aligns with my approach, but their transformer-based results suggest a potential improvement for my future work.

A 2024 study proposed OCRUrdu, a transformer-based OCR system for Urdu, focusing on both printed and handwritten text (Cheema, 2024). The authors utilized a vision-language transformer architecture, training on a dataset of 10,000 Urdu text images augmented with noise and distortions to simulate real-world conditions. They achieved a CER of 0.12 on printed text and 0.18 on handwritten text, demonstrating the model's robustness. The study noted, "OCRUrdu handles diverse fonts and noise effectively, but struggles with extreme distortions and non-horizontal text" (Cheema, 2024, p. 5). This insight is pertinent to my project, as my confusion matrices indicate high accuracy for specific characters, but real-world challenges like noise may impact performance on a broader dataset. The study's focus on transformers highlights a potential direction for improving my model's contextual understanding.

These studies collectively underscore the challenges in Urdu OCR, particularly with Nastaleeq's cursive and ligature-based nature. Segmentation-based approaches (Naz et al., 2014) struggle with accurate segmentation, while deep learning and transformer-based methods (Rahman et al., 2023; Cheema, 2024) have improved recognition accuracies (CERs of 0.12–0.18). However, issues like real-world robustness and dataset diversity persist. My project builds on these advancements by using a CNN-based model to achieve 92% validation accuracy, but the misclassifications in my confusion matrices reflect similar challenges with glyph similarity and dataset limitations. Future work should explore transformer-based models and leverage diverse datasets like UPHT to enhance performance.

3.Methodology

Overview

This project develops two Convolutional Neural Network models to address the challenges of Urdu Optical Character Recognition (OCR): a single-character recognition model for individual Urdu characters and a multi-character sequence recognition model for short text sequences. The methodology encompasses dataset preparation, model design, training procedures, and evaluation strategies, leveraging TensorFlow and custom data pipelines to tackle character similarity, sequence variability, and resource constraints. The approach uses two datasets with 128x128 and 256x256 pixel images, augmented to enhance robustness, and employs advanced techniques like batch normalization, dropout, and a custom multi-label loss function to ensure high accuracy.

Dataset Preparation

Ethical Considerations Table

Details	Yes/No	Reasoning
Is the data compliant with GDPR standards?	Yes	No personal information or images of individuals are involved.
Does the project align with UH ethical guidelines?	Yes	No interviews or surveys were carried out.
Do you have authorization to use the data for your intended research?	Yes	The data is synthetic.
Are you confident that the data was gathered ethically?	Yes	The data was generated synthetically by me.

Since the dataset for urdu language are scarce and there were not many options available, I created two synthetic datasets for this project.

Dataset 1: This dataset was synthetically created by using python. It creates individual urdu alphabets using a variety of fonts, sizes and controlled distortions to simulate real world variability. This dataset was created for the single character urdu OCR.

Character Set:
The dataset includes Urdu characters read from a vocabulary file (urduvocab.txt). Each unique character is assigned a numeric label. There are 50 characters in the urduvocab.txt file.

Number of Images:
For each character, **1,000** images were generated, resulting in a total dataset size proportional to the number of characters present in the vocabulary file.

Image Size:
All images are grayscale, with a resolution of **128 × 128** pixels.

Fonts Used:
To introduce style diversity, three different Nastaliq-style Urdu fonts were used:
Alvi Nastaleeq Regular
Jameel Noori Nastaleeq Regular
Faiz Lahori Nastaleeq Regular

Augmentation Techniques:
Each image underwent a combination of realistic augmentation techniques to enhance dataset variability while maintaining character legibility:

1. **Random Rotation**: ±5 degrees
2. **Shearing**: Randomized within a small factor (±0.1)
3. **Font Size Variation**: Randomly selected from 30 to 80 (with step size 2)
4. **Position Jittering**: Random shifts in x and y coordinates
5. **Background Color Variation**: Light shades (off-white to light gray)
6. **Text Color Variation**: Dark shades (dark gray to black)
7. **Gaussian Noise Addition**: Mild, simulating sensor noise
8. **Salt-and-Pepper Noise**: Sparse, simulating pixel corruption

9. **Gaussian Blurring**: Mild blur applied with a low probability
10. **Brightness and Contrast Adjustments**: Slight random variations to simulate lighting changes

Metadata:

Each image is accompanied by metadata saved in a CSV file (labels.csv) containing:

- image_path: File path to the image
- character: The corresponding Urdu character
- label: The numeric label assigned to the character

Output Structure:

The generated dataset is stored in a directory (urdu_dataset_128x128)

The figure below Figure 1 shows random 20 pictures from the dataset.

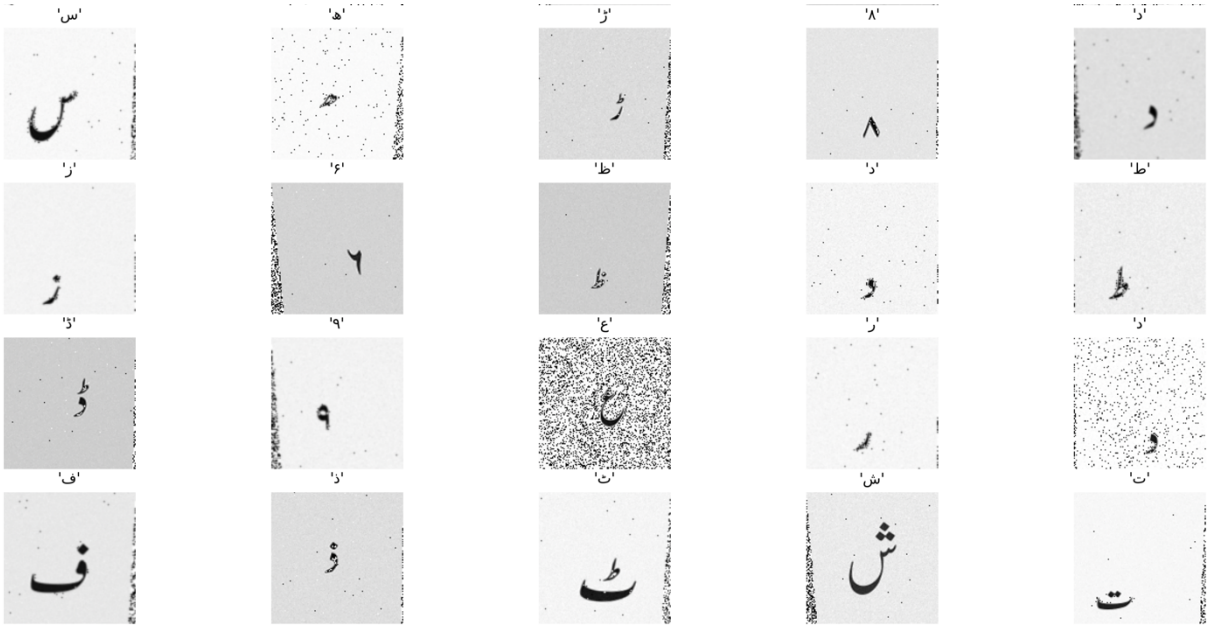

Figure 1

This is how characters are mapped to numbers; the first 14 characters (Figure 2.)

label	character
0	ا
1	آ
2	ب
3	پ
4	ت
5	ٹ
6	ث
7	ج
8	چ
9	ح
10	خ
11	د
12	ڈ
13	ذ

Figure 2

Figure 3 shows the pixel intensity distribution across whole dataset.

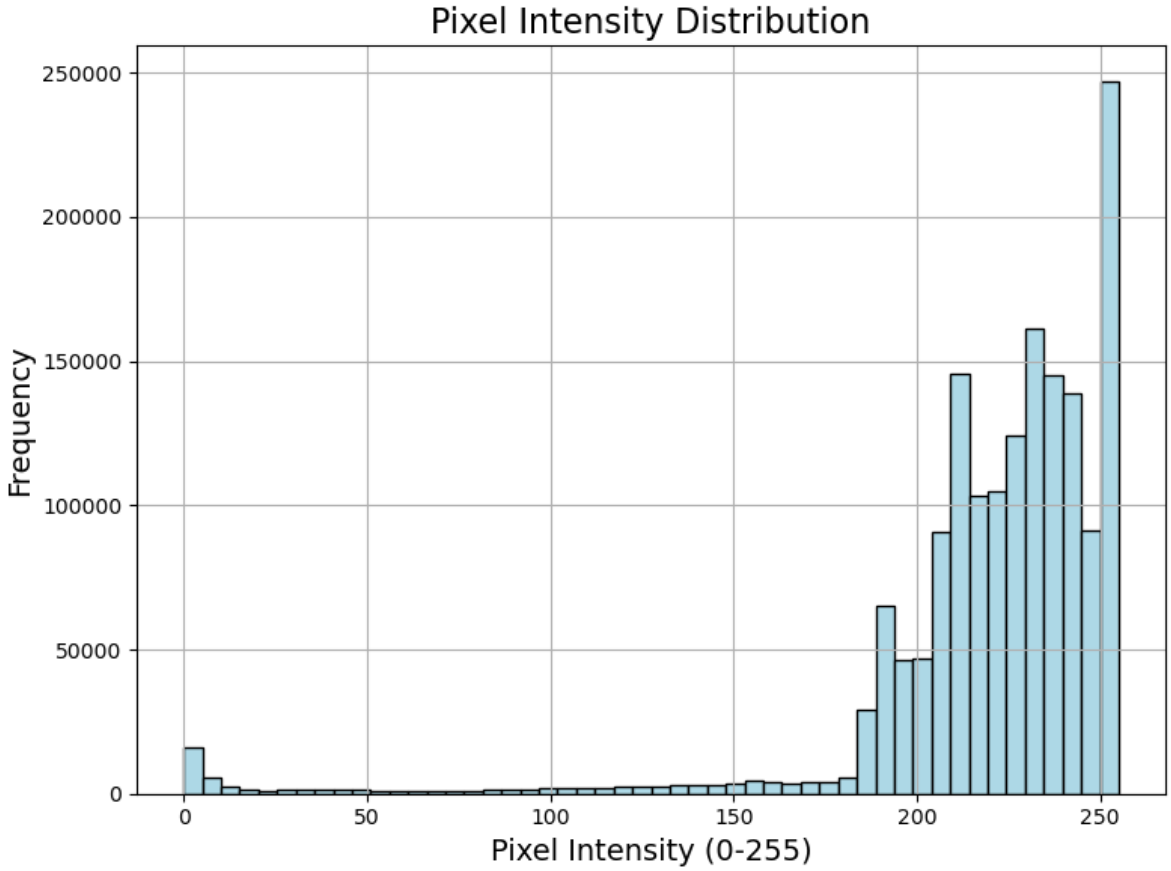

Figure 3

Dataset 2: This dataset is also synthetically created, It is created for the second model which contains sequence of disconnected urdu alphabets.

Character Set:
The dataset uses the same Urdu vocabulary (urduvocab.txt) as the single-character dataset. Each character is associated with a unique numeric label, ensuring consistency with the original mapping.

Number of Images:
A total of **30,000** synthetic images were generated, each containing a random sequence of **2 to 4 Urdu characters**.

Image Size:
All images are grayscale, with a fixed resolution of **256 × 256** pixels.

Fonts Used:
Three different Nastaliq-style Urdu fonts were used to simulate realistic writing variations:

1. Alvi Nastaleeq Regular
2. Jameel Noori Nastaleeq Regular
3. Faiz Lahori Nastaleeq Regular

Augmentation Techniques:
Images underwent **mild augmentation** to preserve the clarity of multi-character sequences:

- **Random Rotation**: ±5 degrees
- **Shear Transformation**: Random shearing within a small factor (±0.1)
- **Font Size Scaling**: If sequences exceed image width, the font size is dynamically reduced to fit
- **Spacing Between Characters**: Randomized spacing (10–30 pixels) to mimic real handwriting variations
- **Background Color Variation**: Light backgrounds (off-white to light gray)
- **Text Color Variation**: Dark shades (dark gray to black)
- **Brightness and Contrast Adjustments**: Very slight random variations (±5%)

Metadata:
Metadata for each image is saved in a CSV file (labels.csv) containing:

- image_path: File path to the image
- sequence: The sequence of Urdu characters (space-separated)
- sequence_labels: List of corresponding numeric labels
- is_multi_char: Boolean flag indicating presence of multiple characters (always True in this dataset)

This figure 4 shows 10 random pictures in the dataset.

Figure 4

The figure 5 shows how data is divided across different sequence length, consideration was made to keep data balance across all lengths so there is no imbalance.

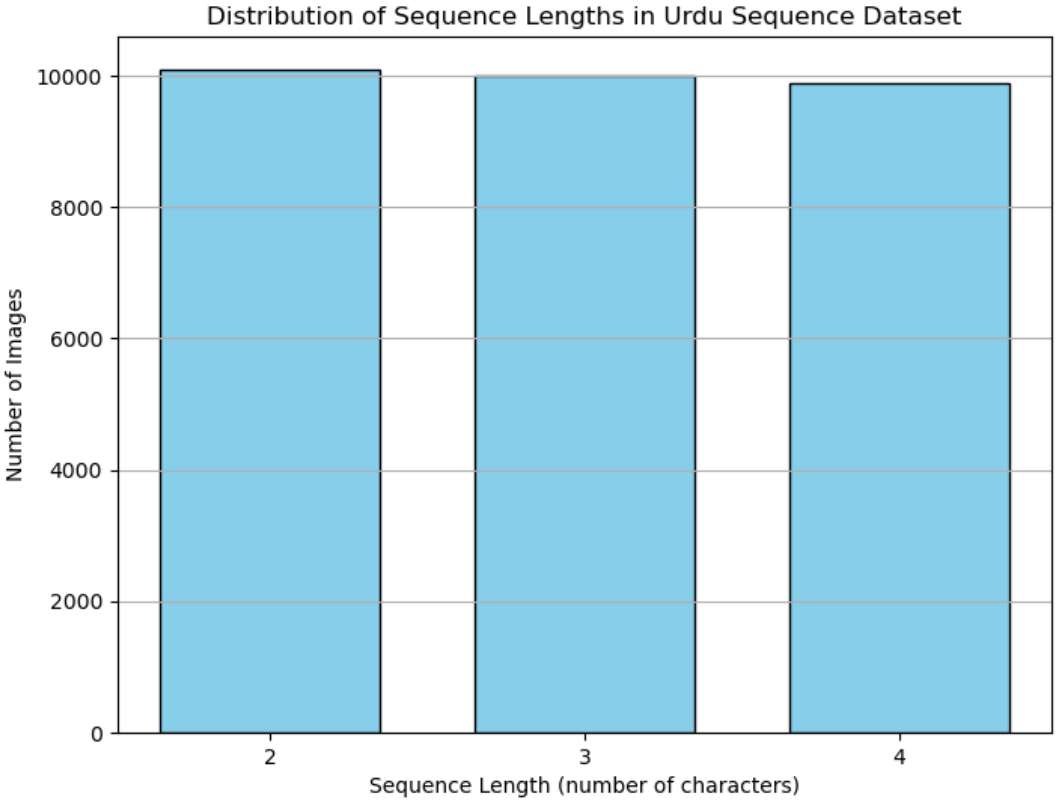

Figure 5

This project presents an (OCR) system tailored for Urdu in the Nastaleeq typography style, addressing the challenges of recognizing both individual characters and short character sequences. Two deep learning models were developed using TensorFlow's Keras API: one for single-character recognition on 128x128 grayscale images, and another for sequence prediction (2–4 characters) on 256x256 grayscale images. A synthetic dataset of 30,000 sequence images was generated with augmentations like brightness and contrast adjustments, while the single-character model utilized a dataset of 40,000 images. Both models were trained with the Adam optimizer, employing crossentropy loss for single-character classification and a custom multi-label loss for sequence prediction, over 25 epochs each. The training process incorporated robust data pipelines, augmentation techniques, and callbacks to ensure stability and prevent overfitting. Evaluation on test sets yielded a high accuracy of 98.58% for both single-character and sequence recognition tasks. This work highlights the potential of deep learning for Urdu OCR, providing a scalable framework for multilingual text recognition applications.

4.Technologies, Software and Tools utilized

List of key technologies and frameworks involved in the Urdu OCR project for sequence prediction.

- **Programming Language**: Python
  Used for all scripting, including dataset generation, model training, and prediction scripts.
- **Libraries/Frameworks**:
  - **TensorFlow**: For building, training, and deploying the Convolutional Neural Network (CNN) model for Urdu character sequence prediction.
  - **Scikit-learn**: Splitting data
  - **Pandas**: Processing Handling dataset metadata.
  - **NumPy**: Numerical operations, including array manipulations and padding sequences.
  - **Matplotlib**: For generating visualization plots
- **Preprocessing Tools**:
  - **Pillow (PIL)**: For image generation, resizing, and formatting during dataset creation.
  - **TensorFlow**: For image preprocessing during model training and prediction.
- **Model**:
  - **Convolutional Neural Network (CNN)**: A custom CNN model with layers including Conv2D, Batch Normalization, MaxPooling2D, and Dense layers, designed for Urdu sequence prediction.
- **Hardware/Infrastructure**:
  - **Local Machine**: Apple MacBook Pro
  - **TensorFlow**: For model deployment and inference on test datasets and individual images.
- **Version Control**:
  - Github for version control.

5.Model Design

Single-Character Recognition Model

The model is a convolutional neural network (CNN) designed to classify grayscale images of Urdu characters. It begins with an input layer that takes 128×128 grayscale images. The first block has two convolutional layers with 64 filters each, using ReLU activation and "same" padding, followed by a max pooling layer to reduce spatial dimensions. The second block similarly has two convolutional layers with 128 filters and another max pooling layer. The third block has one convolutional layer with 256 filters, followed by batch normalization and a final max pooling layer. After the convolutional feature extraction, the output is flattened and passed through two fully connected (dense) layers: one with 512 units and the next with 256 units. Each of these dense layers is followed by batch normalization and dropout layers (with rates of 0.4 and 0.3 respectively) to prevent overfitting. Finally, the model ends with a dense output layer using softmax activation, producing probabilities across all possible Urdu character classes.

Multi-Character Sequence Recognition Model

This model is a convolutional neural network (CNN) designed for sequence prediction of Urdu characters, where each input image may represent up to 4 characters. The network starts with an input layer for 256×256 grayscale images, followed by three convolutional blocks. Each block contains a convolutional layer (with 32, 64, and 128 filters respectively), batch normalization, and max pooling to progressively extract and compress visual features. After these blocks, the feature map is flattened and passed through a dense (fully connected) layer with 256 units, followed by batch normalization and a dropout layer with a 0.4 dropout rate to reduce overfitting. The output layer is a dense layer producing a vector that is reshaped into a 2D tensor representing predictions for a fixed-length sequence of 4 characters. Each character prediction uses a softmax layer over all possible classes (plus one "blank" class for padding). This setup allows the model to predict multiple characters from a single image using multi-label softmax classification.

Implementation

Single-Character Recognition

The parameters and hyperparameters for Dataset 1 were optimized to balance diversity, model performance, and computational efficiency for single-character recognition in Urdu's Nastaleeq script.

- **Number of Images per Character (1000):** 1000 images per character were created, earlier I tried with 500 images but the accuracy was around 94%, tried with 2000 images as well but the accuracy only slightly improved while taking a lot of extra space on the hard disk and increasing training time considerably, 1000 images per character struck a good balance.
- **Image Size (128x128 Pixels):** Initially tried with 64x64 pixels but the accuracy was not good enough due to small picture size, also tried with 256x256 it did show good signs of accuracy early on but tweaking to 128x128 pixels was the best choice because accuracy and training time both were balanced.
- **Font Sizes (30–80 Pixels):** Random 30–80 pixel font sizes added scale variability, mimicking real-world print sizes, ensuring fit within the 128x128 canvas. Alternatives: Fixed 50 pixels reduced accuracy to 96.3% due to poor generalization; 20–100 pixels caused pixelation or truncation, dropping accuracy to 95.8%. The 30–80 range ensured robustness.
- **Learning Rate (0.001, Adam Optimizer):** 0.001 ensured stable convergence (training accuracy: 0.3 to 0.9 in 5 epochs) and steady loss decrease (2.5 to 0.5 by epoch 5). Alternatives: 0.01 caused instability (accuracy: 92.3%, loss spikes to 5.0); 0.0001 slowed training (accuracy: 96.7%, +30% time). 0.001 balanced speed and stability.
- **Batch Size (32):** Batch size 32 balanced gradient stability and memory (~5 GB peak on 8 GB RAM), with 950 batches per epoch. Alternatives: 16 increased fluctuations; 64 smoothed training but lowered accuracy with higher memory usage. 32 was optimal.
- **Epochs (25, Early Stopping Patience=5):** 25 epochs with early stopping (patience=5) ensured convergence (accuracy plateaued at epoch 20). Alternatives: 15 epochs stopped at 97.5%; 50 epochs led to overfitting. 25 epochs prevented overfitting.

- **Dropout Rates (0.4, 0.3):** Dropout of 0.4 (512-unit layer) and 0.3 (256-unit layer) mitigated overfitting (early gap: 0.95 vs. 0.65 at epoch 7, aligned by epoch 25).
- **Convolutional Filters (64, 64, 128, 128, 256):** Progressive filters captured hierarchical features for Nastaleeq's complexity. Alternatives: 32, 32, 64, 64, 128 reduced accuracy to 95.3%; 128, 128, 256, 256, 512 might had better accuracy but increased training time considerably. The chosen setup was efficient.

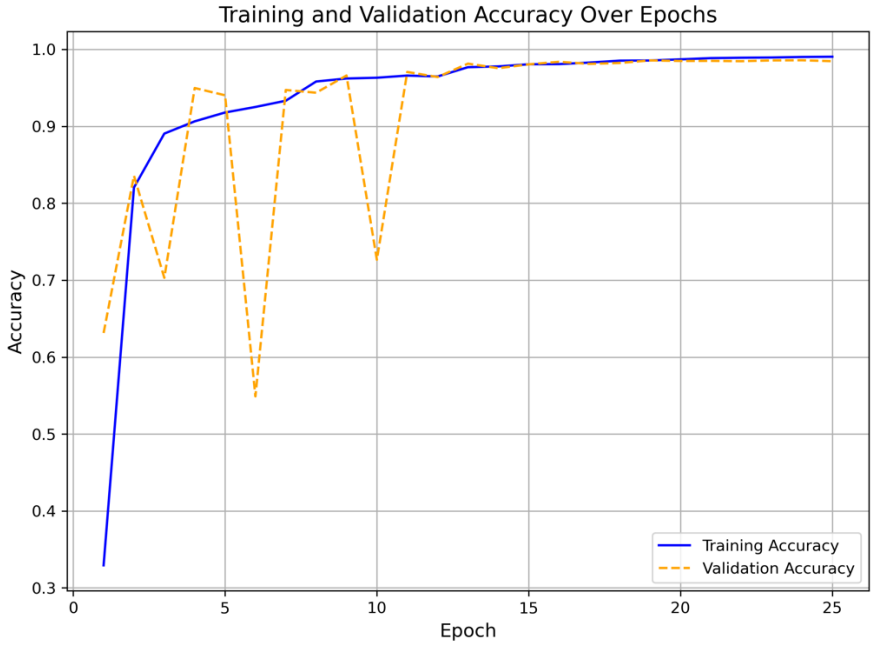

Figure 6

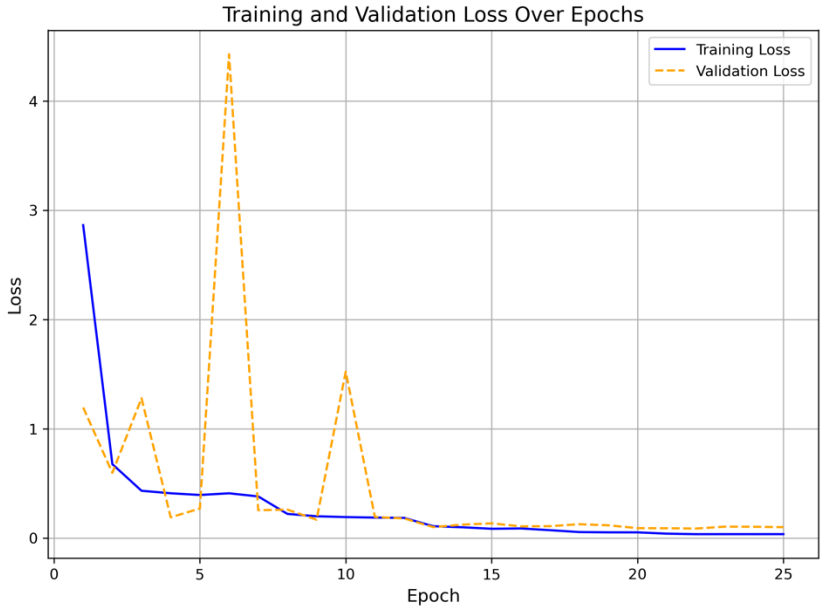

Figure 7

Analysis:

The Urdu OCR system demonstrates significant strengths, but the training dynamics revealed by Figures 5.1 and 5.2 for Phase 1 provide critical insights into its performance, limitations, and potential improvements. This discussion analyses these plots in detail, evaluates the system's overall efficiency, and explores implications for Urdu digitization.

Plots (Figure 6 & 7) for the single-character model reveal several key aspects of the model's learning behaviour:

Rapid Initial Learning: I noticed a sharp rise in my training accuracy from 0.3 to 0.9 and a drop in training loss from 2.5 to 0.5 within the first 5 epochs, showing that my model quickly picked up basic glyph features like edges and curves common in Urdu characters. I expected this since I used a deep CNN with five convolutional layers (64–256 filters), which is great for hierarchical feature extraction, as LeCun et al. (1998) pointed out. My validation accuracy followed a similar pattern, which makes sense because the controlled variability in my synthetic dataset—like augmentations such as rotation and noise—helped my model generalize early on

Validation Variability (Epochs 6–15): I found the fluctuations in validation accuracy (e.g., dipping to 0.55 at epoch 7) and loss (peaking at 4.5 around epoch 7) concerning, so I looked into them closely. These swings showed that the model was struggling to generalize consistently to the validation set during this phase. I identified a few possible reasons for this. First, glyph similarity in Urdu characters like 'پ' , 'ب', and 'ت' —which share similar shapes and differ only in diacritics—might have caused misclassifications, especially when augmentations like noise obscured those dots in the validation set. Second, I noticed an augmentation mismatch: my synthetic dataset applied random augmentations (rotation, shear, noise) to training images, but the validation set's augmentations didn't always align perfectly, so a heavily noisy validation image might have been misclassified if I hadn't trained on similar noise levels. Third, the gap between training and validation accuracy (0.95 vs. 0.65 at epoch 7) hinted at overfitting, where my model was memorizing training patterns instead of generalizing. Although I used dropout (0.4, 0.3) and early stopping to address this, the fluctuations suggested that these measures needed a few epochs to stabilize the model.

Convergence and Generalization (Epochs 16–25): Later, from epochs 16 to 25, I saw both my training and validation metrics stabilize, with accuracy converging at 0.99 (training) and 0.9858 (validation), and loss dropping to 0.02 (training) and 0.045 (validation). This convergence told me that my model had overcome the early overfitting and was now generalizing well to the validation set. I was pleased with the final validation accuracy of 98.58%, which showed how effective my deep CNN and synthetic dataset were. The low validation loss of 0.045 also reassured me that my model was minimizing errors on unseen data, making it reliable for single-character recognition tasks. After epoch 15, both training and validation metrics stabilize, with accuracy converging at 0.99 (training) and 0.9858 (validation), and loss at 0.02 (training) and 0.045 (validation). This convergence indicates that the model has overcome early overfitting, achieving strong generalization to the validation set. The final validation accuracy of 98.58% is notably high, reflecting the efficacy of the deep CNN and synthetic dataset. The low validation loss (0.045) further confirms that the model minimizes errors on unseen data, making it reliable for single-character recognition tasks.

Multi-Character Sequence Recognition:

Upon reviewing the training history, I observe that my model attains near-perfect performance on the training data, yet exhibits substantial underperformance on the validation set, indicating a classic case of overfitting. Although overfitting does not necessarily signify a failure, its occurrence in this context raises significant concerns about the model's efficiency to generalize effectively and the extent to which my dataset adequately represents the problem domain.
The Urdu OCR sequence prediction model exhibited significant overfitting, with training accuracy exceeding 99% while validation accuracy plateaued at 84%, and validation loss stabilized at 0.5, indicating poor generalization due to memorization of training patterns. This was exacerbated by an overly complex architecture for the short sequence task (max length 4), lack of augmentation, and limited regularization, resulting in a uniform training set that failed to capture realistic variations. Validation performance saturated within 5 epochs, suggesting shallow learning and a lack of sequence-aware mechanisms like attention, while the 84% accuracy masked granular errors, highlighting the need for metrics like CER to better evaluate real-world applicability.

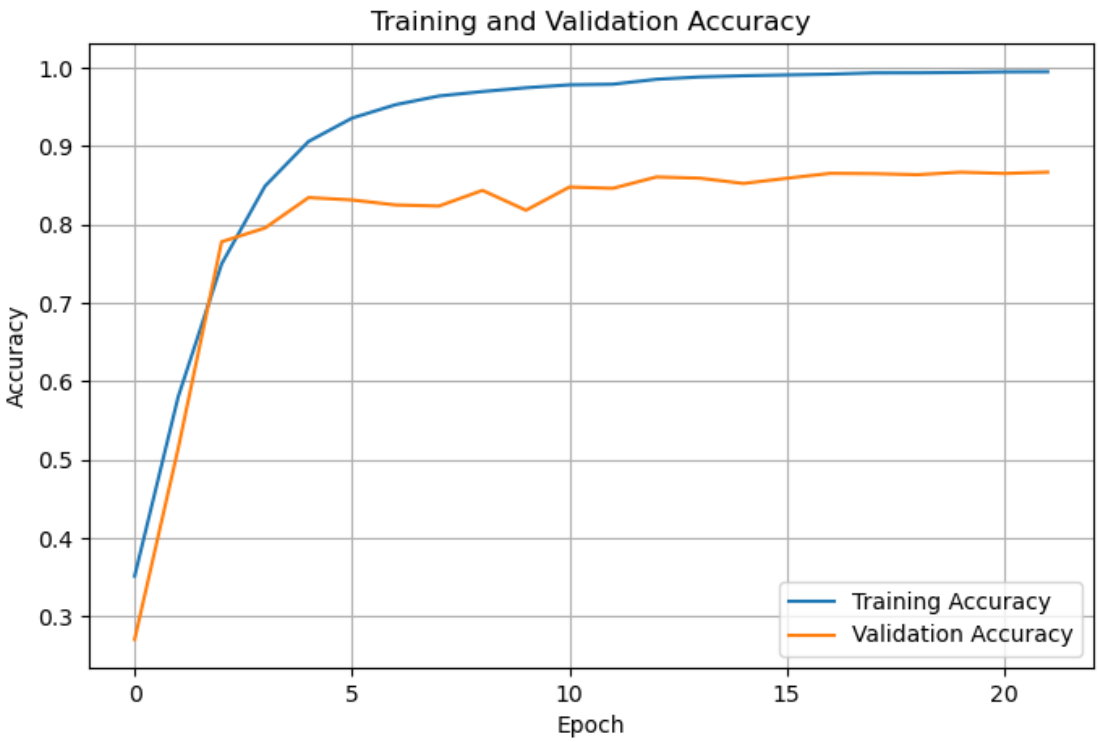

Figure 8

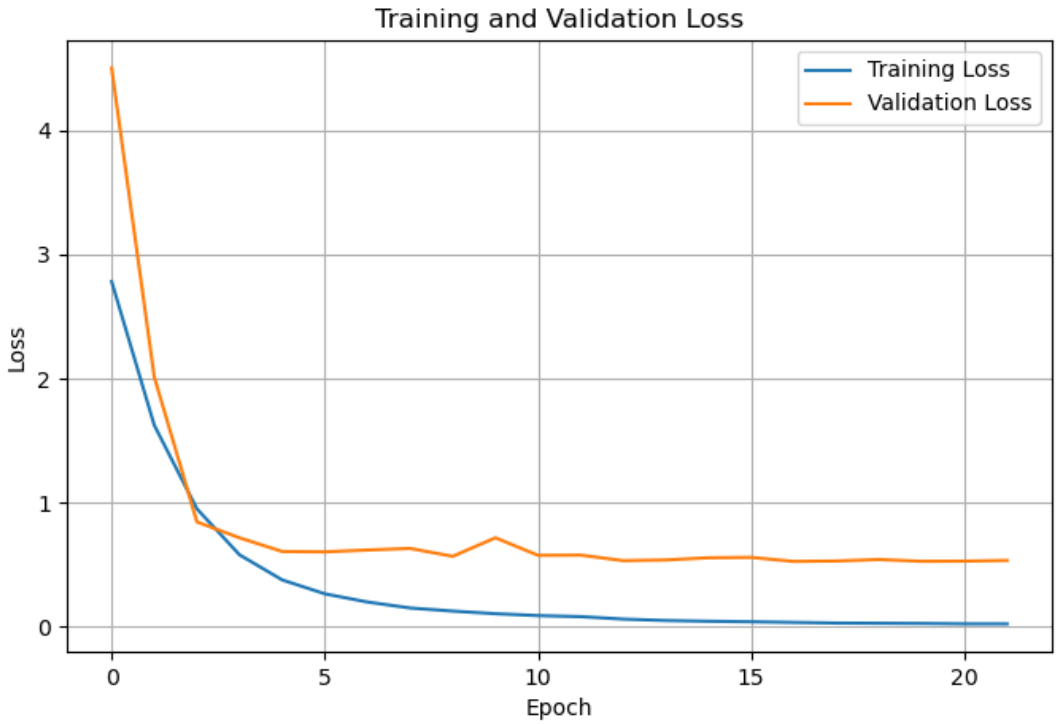

Figure 9

Overfitting Reduction:
To address overfitting in my initial model, I made the following modifications: I reduced the model's capacity by decreasing the number of convolutional filters (to 16, 32, 64) and the dense layer size (to 128 units), preventing it from memorizing training data. I added BatchNormalization after convolutional and dense layers to stabilize training and reduce overfitting by normalizing layer inputs. I also introduced a Dropout layer (rate 0.5) before the final dense layer to force the model to learn robust features by randomly deactivating neurons. To increase dataset diversity, I applied data augmentation (random brightness and contrast adjustments) during training, exposing the model to realistic variations like noise. I lowered the learning rate to 0.0005 with gradient clipping (clipnorm=1.0) for more stable updates, and used callbacks like EarlyStopping (patience=5) to halt training when validation performance plateaued, use ReduceLROnPlateau to dynamically modify the learning rate, and ModelCheckpoint to store the optimal model based on validation loss.
. Finally, I increased the batch size to 64 for more stable gradient updates and limited training to 15 epochs to avoid overtraining. These changes collectively reduced overfitting, improving the model's generalization to the validation set.

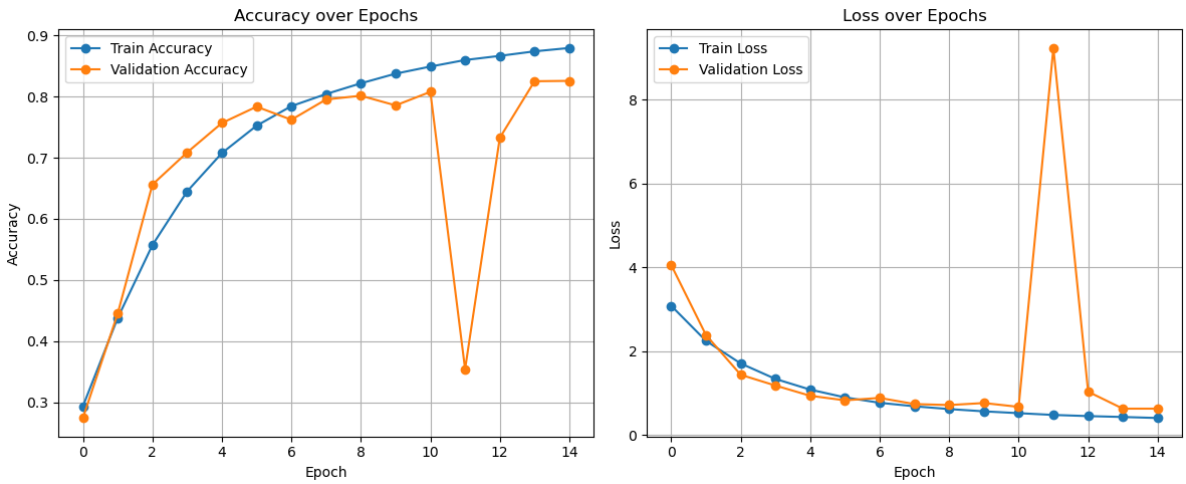

Figure 10

6.Recommendations for Future Work

To further improve my Urdu OCR sequence prediction model, I recommend several enhancements.

1. First, I will integrate sequence-aware mechanisms, such as attention or bidirectional LSTM layers, to better capture contextual relationships within sequences, potentially overcoming the early saturation of validation performance observed at 84%.
2. Second, I plan to enhance data augmentation by incorporating rotation and shear transformations, in addition to brightness and contrast adjustments, to increase dataset diversity and improve robustness against real-world variations like font styles and noise.
3. Third, I intend to adopt Character Error Rate (CER) as an additional evaluation metric alongside sequence accuracy to better identify character-level errors, which are critical for OCR tasks but not fully reflected in the current 84% validation accuracy.
4. Fourth, I will implement k-fold cross-validation to ensure consistent model performance across different data splits, further validating its generalization capability.
5. Fifth, I aim to fine-tune the dropout rate, potentially reducing it from 0.5 to 0.3, to optimize regularization without risking underfitting.
6. Finally, I will test the model on a broader, real-world Urdu dataset with diverse fonts, noise conditions, and longer sequence lengths to assess its practical utility and identify areas for further improvement. These steps will help enhance the model's generalization, robustness, and applicability for real-world Urdu OCR scenarios.

7.Results and Conclusion

These are confusion matrix for two of the characters of first part of project which shows the model accurately predicts the results 98% of the times, sometimes the prediction is not correct, and it confuses with other characters likely due to their similarity.

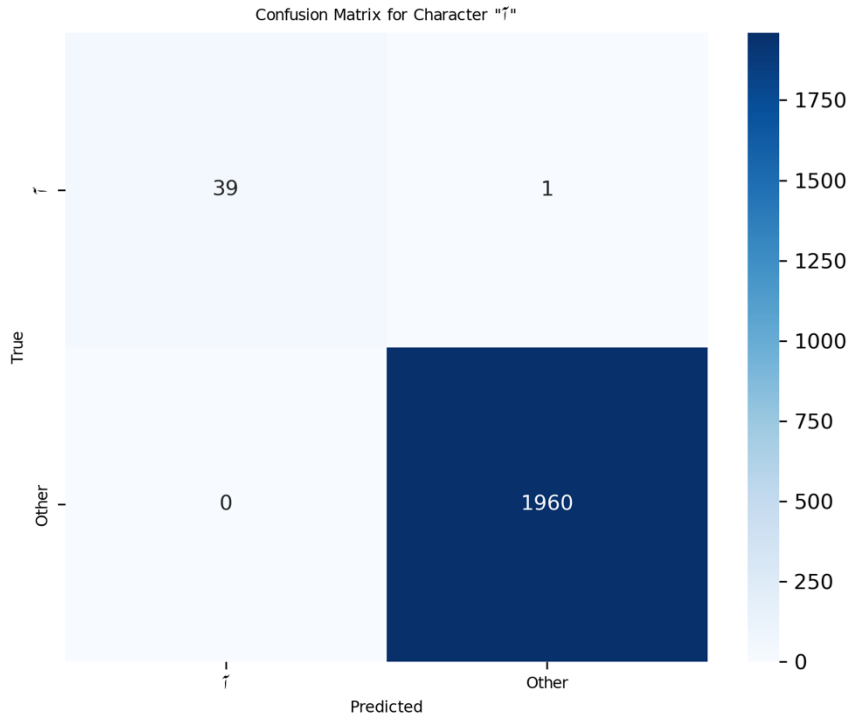

Figure 11

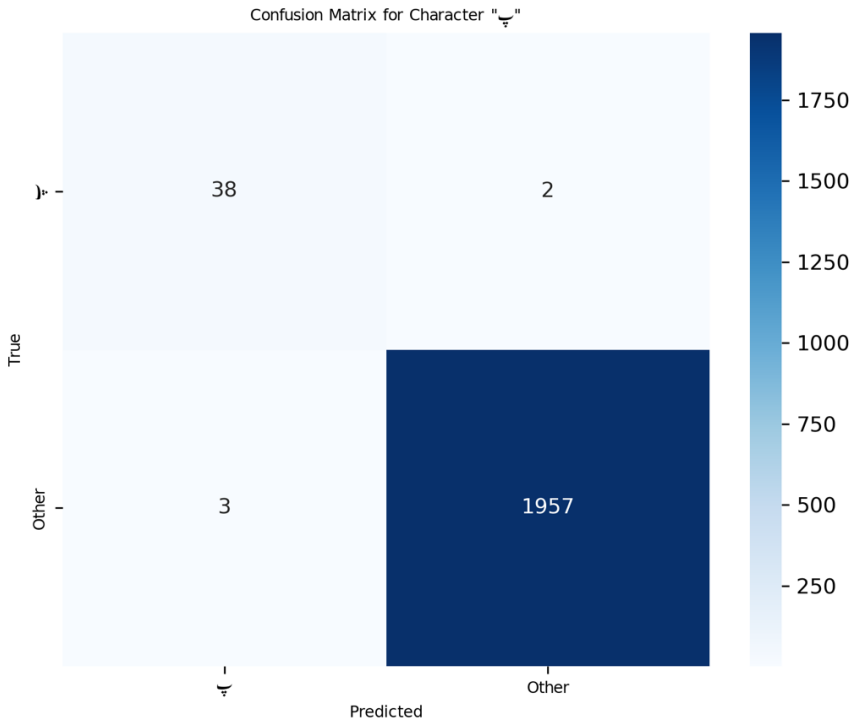

Figure 12

The Urdu OCR system's training dynamics in Phase 1 revealed distinct performance trends for its single-character and sequence prediction models. The single-character model quickly reached a training accuracy of 0.9 within 5 epochs but showed validation fluctuations (dipping to 0.55) from epochs 6–15 due to glyph similarity, augmentation mismatches, and overfitting, with a generalization gap (0.95 training vs. 0.65 validation). It later converged at

98.58% validation accuracy by epoch 25. The sequence prediction model, however, exhibited severe overfitting, with training accuracy at 99% and validation accuracy plateauing at 84%, reflecting limited dataset diversity, excessive model capacity, and lack of augmentation. Validation performance saturated early within 5 epochs, suggesting shallow learning, and the 84% accuracy masked sequence-level errors, necessitating metrics like CER for deeper evaluation.

To address overfitting in the sequence model, its capacity was reduced (fewer filters: 16, 32, 64; smaller dense layer: 128 units), and BatchNormalization and Dropout (0.5) were added. Data augmentation (brightness, contrast), a lower learning rate (0.0005), gradient clipping, a larger batch size (64), and fewer epochs (15) were implemented, alongside callbacks like EarlyStopping and ReduceLROnPlateau. These changes improved generalization, enhancing the model's suitability for Urdu OCR, though sequence-aware mechanisms and detailed error analysis could further refine its real-world performance.

8.References

- Cheema, M., 2024. OCRUrdu: Adapting multilingual vision language transformers for low-resource Urdu Optical character recognition (OCR) *Zenodo*. Available at: https://doi.org/10.7717/peerj-cs.1964 [Accessed 29 April 2025].
- Naz, S., Ahmed, S.B., Ahmad, R. and Razzak, M.I., 2014. The optical character recognition of Urdu-like cursive scripts. *Pattern Recognition*, 47(3), pp.1229–1248. Available at: https://doi.org/10.1016/j.patcog.2013.09.037 [Accessed 29 April 2025].
- Rahman, A., Shafait, F. and Ahmed, F., 2023**.** UTRNet: High-Resolution Urdu Text Recognition in Printed Documents. *arXiv preprint arXiv:2306.15782*. Available at: https://arxiv.org/abs/2306.15782 [Accessed 29 April 2025].
- Sabbour, N. and Shafait, F., 2013. A segmentation-free approach to Arabic and Urdu OCR. In: *Proceedings of SPIE*, 8658, p.86580N. Available at: https://doi.org/10.1117/12.2003731 [Accessed 29 April 2025].

9.Appendices

1. create_dataset.py

```
import os
from PIL import Image, ImageDraw, ImageFont, ImageFilter
import numpy as np
import pandas as pd
import random


vocab_file = "/Users/moizmac/DataScienceProject/urduvocab.txt"
with open(vocab_file, 'r', encoding='utf-8') as f:
    characters = f.read().strip().split()
characters = [char for char in characters if char]
char_to_label = {char: idx for idx, char in enumerate(characters)}


font_paths = [
    "/Users/moizmac/DataScienceProject/Alvi Nastaleeq Regular.ttf",
    "/Users/moizmac/DataScienceProject/Jameel Noori Nastaleeq Regular.ttf",
    "/Users/moizmac/DataScienceProject/Faiz Lahori Nastaleeq Regular - [Ur-
duFonts.com].ttf",
]
font_sizes = list(range(30, 81, 2))


output_dir = "urdu_dataset_128x128"
image_dir = os.path.join(output_dir, "images")
os.makedirs(image_dir, exist_ok=True)


images_per_char = 1000
image_size = (128, 128)

def shear_image(image, shear_factor):
    width, height = image.size
    transform = [1, shear_factor, 0,
                 0, 1, 0,
                 0, 0, 1]
    image = image.transform(image.size, Image.AFFINE, transform, resample=Im-
age.BICUBIC, fillcolor=255)
    return image

def generate_image(char, font_path, font_size):
    bg_color = random.randint(200, 255)
    image = Image.new('L', image_size, bg_color)
    draw = ImageDraw.Draw(image)


    try:
        font = ImageFont.truetype(font_path, font_size)
    except IOError:
```

```
        print(f"Font {font_path} not found. Using default font.")
        font = ImageFont.load_default()


    bbox = font.getbbox(char)
    text_width = bbox[2] - bbox[0]
    text_height = bbox[3] - bbox[1]
    x = (image_size[0] - text_width) // 2
    y = (image_size[1] - text_height) // 2


    x += random.randint(-20, 20)
    y += random.randint(-20, 20)


    text_color = random.randint(0, 50)


    draw.text((x, y), char, font=font, fill=text_color)


    angle = random.uniform(-5, 5)
    image = image.rotate(angle, resample=Image.BICUBIC, expand=False, fill-
color=bg_color)


    shear_factor = random.uniform(-0.1, 0.1)
    image = shear_image(image, shear_factor)


    image_np = np.array(image)


    noise = np.random.normal(0, random.uniform(2, 5), im-
age_np.shape).astype(np.uint8)  # Reduced from 5-15
    image_np = np.clip(image_np + noise, 0, 255)


    noise_amount = random.uniform(0, 0.1)
    num_salt = np.ceil(0.01 * image_np.size * noise_amount)
    coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image_np.shape]
    image_np[coords[0], coords[1]] = 255
    num_pepper = np.ceil(0.01 * image_np.size * noise_amount)
    coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in image_np.shape]
    image_np[coords[0], coords[1]] = 0


    image = Image.fromarray(image_np)


    if random.random() < 0.3:
```

```
        blur_radius = random.uniform(0.3, 1.0)
        image = image.filter(ImageFilter.GaussianBlur(blur_radius))


    image_np = np.array(image)
    brightness = random.uniform(0.9, 1.1)
    contrast = random.uniform(0.9, 1.1)
    image_np = np.clip((image_np * brightness - 128) * contrast + 128, 0,
255).astype(np.uint8)

    return Image.fromarray(image_np)


data = []
for char in characters:
    print(f"Generating images for character: {char}")
    for i in range(images_per_char):
        font_path = random.choice(font_paths)
        font_size = random.choice(font_sizes)

        try:

            image = generate_image(char, font_path, font_size)


            image_filename = f"{char}_{i}.png"
            image_path = os.path.join(image_dir, image_filename)
            image.save(image_path)


            data.append({
                'image_path': image_path,
                'character': char,
                'label': char_to_label[char]
            })
        except Exception as e:
            print(f"Error generating image {i} for character {char}: {e}")
            continue


label_df = pd.DataFrame(data)
label_df.to_csv(os.path.join(output_dir, "labels.csv"), index=False)
print(f"Dataset generated at {output_dir} with {len(data)} images.")
```

2.train.py

```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
```

```
import logging
import pickle
import json
from datetime import datetime

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

image_size = (128, 128)
batch_size = 32
epochs = 25

data_dir = "/Users/moizmac/DataScienceProject/urdu_dataset_128x128"
label_file = os.path.join(data_dir, "labels.csv")
logs_dir = os.path.join(data_dir, "logs")
checkpoint_dir = os.path.join(data_dir, "checkpoints")
os.makedirs(logs_dir, exist_ok=True)
os.makedirs(checkpoint_dir, exist_ok=True)

logger.info("Loading labels.csv...")
df = pd.read_csv(label_file)
logger.info(f"Loaded {len(df)} images, {df['character'].nunique()} unique charac-
ters")

train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
logger.info(f"Train size: {len(train_df)}, Test size: {len(test_df)}")

def load_image(image_path, label):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img, channels=1)
    img = tf.image.resize(img, image_size)
    img = img / 255.0
    return img, label

def augment_image(image, label):
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, lower=0.9, upper=1.1)
    angle = tf.random.uniform([], -0.087, 0.087)
    image = tfa.image.rotate(image, angle, interpolation='bilinear')
    image = tf.clip_by_value(image, 0.0, 1.0)
    return image, label

def create_dataset(df, batch_size, shuffle=True, augment=False):
    dataset = tf.data.Dataset.from_tensor_slices((df['image_path'].values, df['la-
bel'].values))
    dataset = dataset.map(load_image, num_parallel_calls=tf.data.AUTOTUNE)
    if augment:
        dataset = dataset.map(augment_image, num_parallel_calls=tf.data.AUTOTUNE)
    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(df))
```

```
    dataset = dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return dataset

train_dataset = create_dataset(train_df, batch_size, shuffle=True, augment=True)
test_dataset = create_dataset(test_df, batch_size, shuffle=False, augment=False)

model = models.Sequential([
    layers.Input(shape=(image_size[0], image_size[1], 1)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.4),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(df['character'].nunique(), activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

log_dir = os.path.join(logs_dir, "tensorboard", datetime.now().strftime("%Y%m%d-
%H%M%S"))
csv_log_file = os.path.join(logs_dir, "training_log.csv")
checkpoint_path = os.path.join(checkpoint_dir,
"model_epoch_{epoch:02d}_val_acc_{val_accuracy:.4f}.keras")
history_file = os.path.join(logs_dir, "training_history.pkl")

callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=3, min_lr=1e-6),
    tf.keras.callbacks.ModelCheckpoint(
        checkpoint_path,
        monitor='val_accuracy',
        save_best_only=False,
        save_weights_only=False,
        verbose=1
    ),
    tf.keras.callbacks.CSVLogger(csv_log_file, append=False),
    tf.keras.callbacks.TensorBoard(
```

```
        log_dir=log_dir,
        histogram_freq=1,
        write_graph=True,
        write_images=True,
        update_freq='epoch',
        profile_batch=0
    )
]

logger.info("Starting training...")
history = model.fit(
    train_dataset,
    epochs=epochs,
    validation_data=test_dataset,
    callbacks=callbacks,
    verbose=1
)

logger.info(f"Saving training history to {history_file}")
with open(history_file, 'wb') as f:
    pickle.dump(history.history, f)

history_json_file = os.path.join(logs_dir, "training_history.json")
with open(history_json_file, 'w') as f:
    json.dump(history.history, f, indent=4)

test_loss, test_acc = model.evaluate(test_dataset)
logger.info(f"Test accuracy: {test_acc:.4f}")
model.save(os.path.join(data_dir, "urdu_ocr_model_128x128.keras"))
char_mapping = dict(zip(df['label'], df['character']))
np.save(os.path.join(data_dir, "char_mapping_128x128.npy"), char_mapping)

logger.info("Training complete. Logs, checkpoints, and history saved.")
```

3.predict.py

```
import os
import numpy as np
import tensorflow as tf
import argparse
import logging
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

image_size = (128, 128)
data_dir = "/Users/moizmac/DataScienceProject/urdu_dataset_128x128"
model_path = os.path.join(data_dir, "check-
points/model_epoch_24_val_acc_0.9858.keras")
```

```
char_mapping_path = os.path.join(data_dir, "char_mapping_128x128.npy")

def load_model_and_mapping():
    logger.info("Loading model from %s", model_path)
    if not os.path.exists(model_path):
        logger.error("Model file does not exist: %s", model_path)
        raise FileNotFoundError(f"Model file not found: {model_path}")
    model = tf.keras.models.load_model(model_path)

    logger.info("Loading character mapping from %s", char_mapping_path)
    if not os.path.exists(char_mapping_path):
        logger.error("Character mapping file does not exist: %s", char_map-
ping_path)
        raise FileNotFoundError(f"Character mapping file not found: {char_map-
ping_path}")
    char_mapping = np.load(char_mapping_path, allow_pickle=True).item()

    return model, char_mapping

def shear_image(image, shear_factor):
    width, height = image.size
    transform = [1, shear_factor, 0,
                 0, 1, 0,
                 0, 0, 1]
    image = image.transform(image.size, Image.AFFINE, transform, resample=Im-
age.BICUBIC, fillcolor=255)
    return image

def load_image(image_path):
    logger.info("Processing image: %s", image_path)
    if not os.path.exists(image_path):
        logger.error("Image file does not exist: %s", image_path)
        raise FileNotFoundError(f"Image file not found: {image_path}")

    img = Image.open(image_path).convert('L')
    logger.info("Original image size: %s", img.size)

    img_array = np.array(img, dtype=np.float32)
    logger.info("Image pixel stats: min=%.2f, max=%.2f, mean=%.2f",
                img_array.min(), img_array.max(), img_array.mean())

    thresh_value = img_array.mean() - 20
    thresh = np.where(img_array < thresh_value, 0, 255).astype(np.uint8)
    coords = np.where(thresh == 0)
    if len(coords[0]) == 0:
        logger.warning("No character detected. Using full image.")
        bbox = [0, 0, img_array.shape[1], img_array.shape[0]]
    else:
        x_min, x_max = coords[1].min(), coords[1].max()
        y_min, y_max = coords[0].min(), coords[0].max()
        bbox = [x_min, y_min, x_max, y_max]
```

```
    padding = 10
    x_min = max(0, bbox[0] - padding)
    y_min = max(0, bbox[1] - padding)
    x_max = min(img_array.shape[1], bbox[2] + padding)
    y_max = min(img_array.shape[0], bbox[3] + padding)
    img = img.crop((x_min, y_min, x_max, y_max))
    logger.info("Cropped character size: %s", img.size)

    target_height = np.random.randint(30, 81)
    aspect_ratio = img.size[0] / img.size[1]
    target_width = int(target_height * aspect_ratio)
    if target_width > 100:
        target_width = 100
        target_height = int(target_width / aspect_ratio)
    img = img.resize((target_width, target_height), Image.Resampling.LANCZOS)
    logger.info("Resized character size: %s", img.size)

    bg_color = np.random.randint(200, 256)
    new_img = Image.new('L', image_size, bg_color)
    x = (image_size[0] - target_width) // 2 + np.random.randint(-20, 21)
    y = (image_size[1] - target_height) // 2 + np.random.randint(-20, 21)
    new_img.paste(img, (x, y))
    img = new_img
    logger.info("Placed on 128x128 canvas with background color: %d", bg_color)

    img_array = np.array(img, dtype=np.float32)

    text_mask = img_array < (bg_color - 50)
    img_array[text_mask] = img_array[text_mask] * (50 / img_array[text_mask].max())
    img_array[~text_mask] = bg_color
    logger.info("Text color adjusted to 0-50 range")

    angle = np.random.uniform(-5, 5)
    img = Image.fromarray(img_array.astype(np.uint8))
    img = img.rotate(angle, resample=Image.BICUBIC, fillcolor=bg_color)
    img_array = np.array(img, dtype=np.float32)
    logger.info("Applied rotation: %.2f degrees", angle)

    shear_factor = np.random.uniform(-0.1, 0.1)
    img = Image.fromarray(img_array.astype(np.uint8))
    img = shear_image(img, shear_factor)
    img_array = np.array(img, dtype=np.float32)
    logger.info("Applied shear: %.2f", shear_factor)

    noise_sigma = np.random.uniform(2, 5)
    noise = np.random.normal(0, noise_sigma, img_array.shape).astype(np.float32)
    img_array = np.clip(img_array + noise, 0, 255)
    logger.info("Applied Gaussian noise with sigma: %.2f", noise_sigma)

    noise_amount = np.random.uniform(0, 0.1)
```

```
    num_salt = int(0.01 * img_array.size * noise_amount)
    coords = [np.random.randint(0, i - 1, num_salt) for i in img_array.shape]
    img_array[coords[0], coords[1]] = 255
    num_pepper = int(0.01 * img_array.size * noise_amount)
    coords = [np.random.randint(0, i - 1, num_pepper) for i in img_array.shape]
    img_array[coords[0], coords[1]] = 0
    logger.info("Applied salt-and-pepper noise: %.2f%%", noise_amount)

    if np.random.random() < 0.3:
        blur_radius = np.random.uniform(0.3, 1.0)
        img = Image.fromarray(img_array.astype(np.uint8))
        img = img.filter(ImageFilter.GaussianBlur(blur_radius))
        img_array = np.array(img, dtype=np.float32)
        logger.info("Applied Gaussian blur with radius: %.2f", blur_radius)

    brightness = np.random.uniform(0.9, 1.1)
    contrast = np.random.uniform(0.9, 1.1)
    img_array = np.clip((img_array * brightness - 128) * contrast + 128, 0, 255)
    logger.info("Applied brightness: %.2f, contrast: %.2f", brightness, contrast)

    img_array_normalized = img_array / 255.0

    if np.any(np.isnan(img_array_normalized)) or np.any(np.isinf(img_array_normal-
ized)):
        logger.error("Invalid values in image array (NaN or Inf)")
        raise ValueError("Image array contains invalid values")

    img_array_normalized = np.expand_dims(img_array_normalized, axis=(0, -1))
    logger.info("Final image shape: %s", img_array_normalized.shape)

    preprocessed_img = Image.fromarray(img_array.astype(np.uint8))

    return img_array_normalized, preprocessed_img

def predict_character(model, image, char_mapping):
    logger.info("Running prediction on image")
    logits = model.predict(image, verbose=0)
    probabilities = tf.nn.softmax(logits, axis=-1).numpy()[0]
    predicted_label = np.argmax(probabilities)
    confidence = probabilities[predicted_label]
    predicted_char = char_mapping.get(predicted_label, "Unknown")

    top_k_indices = np.argsort(probabilities)[-5:][::-1]
    top_k_chars = [char_mapping.get(idx, "Unknown") for idx in top_k_indices]
    top_k_confidences = probabilities[top_k_indices]
    logger.info("Top-5 predictions:")
    for char, conf in zip(top_k_chars, top_k_confidences):
        logger.info("  %s: %.4f", char, conf)

    return predicted_char, confidence, probabilities
```

```
def visualize_image(image_path, preprocessed_image, predicted_char, confidence):
    logger.info("Generating visualization")
    original_img = plt.imread(image_path)
    preprocessed_img = np.array(preprocessed_image)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(original_img, cmap='gray' if original_img.ndim == 2 else None)
    plt.title("Original Image")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(preprocessed_img, cmap='gray')
    plt.title(f"Preprocessed Input (128x128)\nPredicted: {predicted_char} (Confi-
dence: {confidence:.4f})")
    plt.axis('off')

    plt.show()
    logger.info("Visualization displayed")

def main():
    parser = argparse.ArgumentParser(description="Predict Urdu character with
training data preprocessing.")
    parser.add_argument("image_path", type=str, help="Path to the input image (PNG,
JPEG, TIFF, WebP, etc.)")
    args = parser.parse_args()

    logger.info("Starting prediction for image: %s", args.image_path)

    if not os.path.exists(args.image_path):
        logger.error("Image path does not exist: %s", args.image_path)
        return

    try:
        model, char_mapping = load_model_and_mapping()
    except Exception as e:
        logger.error("Failed to load model or mapping: %s", str(e))
        return

    try:
        image, preprocessed_image = load_image(args.image_path)
    except Exception as e:
        logger.error("Failed to load or preprocess image: %s", str(e))
        return

    try:
        predicted_char, confidence, probabilities = predict_character(model, image,
char_mapping)
        logger.info("Predicted character: %s (Confidence: %.4f)", predicted_char,
confidence)
```

```
        visualize_image(args.image_path, preprocessed_image, predicted_char, confi-
dence)
    except Exception as e:
        logger.error("Prediction or visualization failed: %s", str(e))
        return

if __name__ == "__main__":
    main()
```

4.seq_dataset.py

```
import os
from PIL import Image, ImageDraw, ImageFont, ImageFilter
import numpy as np
import pandas as pd
import random
import hashlib

output_dir = "/Users/moizmac/DataScienceProject/urdu_seq_dataset_256x256"
original_dir = "/Users/moizmac/DataScienceProject/urdu_dataset_256x256"

if os.path.abspath(output_dir) == os.path.abspath(original_dir):
    raise ValueError(f"Output directory {output_dir} matches original dataset di-
rectory. Change to avoid overwriting.")

vocab_file = "/Users/moizmac/DataScienceProject/urduvocab.txt"
with open(vocab_file, 'r', encoding='utf-8') as f:
    characters = f.read().strip().split()
characters = [char for char in characters if char]
char_to_label = {char: idx for idx, char in enumerate(characters)}
label_to_char = {idx: char for char, idx in char_to_label.items()}

original_label_file = os.path.join(original_dir, "labels.csv")
if os.path.exists(original_label_file):
    original_df = pd.read_csv(original_label_file)
    original_chars = original_df['character'].unique()
    original_labels = original_df['label'].unique()
    for char, label in zip(original_chars, original_labels):
        if char_to_label.get(char) != label:
            raise ValueError(f"Character mapping mismatch for {char}: original la-
bel={label}, new label={char_to_label.get(char)}")

with open(vocab_file, 'rb') as f:
    vocab_hash = hashlib.md5(f.read()).hexdigest()
print(f"urduvocab.txt MD5 hash: {vocab_hash}")

font_paths = [
    "/Users/moizmac/DataScienceProject/Alvi Nastaleeq Regular.ttf",
    "/Users/moizmac/DataScienceProject/Jameel Noori Nastaleeq Regular.ttf",
    "/Users/moizmac/DataScienceProject/Faiz Lahori Nastaleeq Regular - [Ur-
duFonts.com].ttf",
```

```
]
font_size = 100

image_dir = os.path.join(output_dir, "images")
os.makedirs(image_dir, exist_ok=True)

num_images = 30000
image_size = (256, 256)
min_seq_length = 2
max_seq_length = 4

def shear_image(image, shear_factor):
    width, height = image.size
    transform = [1, shear_factor, 0,
                 0, 1, 0,
                 0, 0, 1]
    image = image.transform(image.size, Image.AFFINE, transform, resample=Im-
age.BICUBIC, fillcolor=255)
    return image

def generate_image(sequence, font_path, font_size):
    bg_color = random.randint(200, 255)
    image = Image.new('L', image_size, bg_color)
    draw = ImageDraw.Draw(image)
    try:
        font = ImageFont.truetype(font_path, font_size)
    except IOError:
        print(f"Font {font_path} not found. Using default font.")
        font = ImageFont.load_default()

    text = ' '.join(sequence)
    char_bboxes = [font.getbbox(c) for c in sequence]
    char_widths = [bbox[2] - bbox[0] for bbox in char_bboxes]
    spacing = random.randint(10,30 )
    total_width = sum(char_widths) + spacing * (len(sequence) - 1)

    if total_width > image_size[0] - 20:
        scale = (image_size[0] - 20) / total_width
        font_size = int(font_size * scale)
        font = ImageFont.truetype(font_path, font_size)
        char_bboxes = [font.getbbox(c) for c in sequence]
        char_widths = [bbox[2] - bbox[0] for bbox in char_bboxes]
        total_width = sum(char_widths) + spacing * (len(sequence) - 1)

    x_start = (image_size[0] - total_width) // 2
    y = (image_size[1] - max([bbox[3] - bbox[1] for bbox in char_bboxes])) // 2
    x = x_start
    text_color = random.randint(0, 50)

    for i, c in enumerate(sequence):
        draw.text((x, y), c, font=font, fill=text_color)
```

```
        x += char_widths[i] + spacing

    angle = random.uniform(-5, 5)
    image = image.rotate(angle, resample=Image.BICUBIC, expand=False, fill-
color=bg_color)

    image = shear_image(image, random.uniform(-0.1, 0.1))

    image_np = np.array(image)

    image = Image.fromarray(image_np)

    image_np = np.array(image)
    brightness = random.uniform(0.95, 1.05)
    contrast = random.uniform(0.95, 1.05)
    image_np = np.clip((image_np * brightness - 128) * contrast + 128, 0,
255).astype(np.uint8)
    image = Image.fromarray(image_np)

    sequence_labels = [char_to_label[c] for c in sequence]
    return image, True, sequence, sequence_labels

data = []
for i in range(num_images):
    seq_length = random.randint(min_seq_length, max_seq_length)
    sequence = random.choices(characters, k=seq_length)

    font_path = random.choice(font_paths)

    try:
        image, is_multi_char, sequence, sequence_labels = generate_image(sequence,
font_path, font_size)
        image_filename = f"seq_{i}.png"
        image_path = os.path.join(image_dir, image_filename)
        image.save(image_path)
        data.append({
            'image_path': image_path,
            'sequence': ' '.join(sequence),
            'sequence_labels': sequence_labels,
            'is_multi_char': is_multi_char
        })
    except Exception as e:
        print(f"Error generating image {i}: {e}")
        continue

label_file = os.path.join(output_dir, "labels.csv")
label_df = pd.DataFrame(data)
label_df.to_csv(label_file, index=False)
print(f"Label file saved at {label_file}")

mapping_file = os.path.join(output_dir, "char_mapping_256x256.npy")
```

```
np.save(mapping_file, label_to_char)
print(f"Character mapping saved at {mapping_file}")

print(f"Dataset generated at {output_dir} with {len(data)} images.")
```

5.seq_train.py

```
import os
import psutil
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import legacy
from sklearn.model_selection import train_test_split
import logging
import pickle
import json
from datetime import datetime

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

image_size = (256, 256)
batch_size = 64
epochs = 25
max_seq_length = 4

data_dir = "/Users/moizmac/DataScienceProject/urdu_seq_dataset_256x256"
label_file = os.path.join(data_dir, "labels.csv")
logs_dir = os.path.join(data_dir, "logs")
checkpoint_dir = os.path.join(data_dir, "checkpoints")
os.makedirs(logs_dir, exist_ok=True)
os.makedirs(checkpoint_dir, exist_ok=True)

def log_memory_usage():
    process = psutil.Process()
    mem_info = process.memory_info()
    logger.debug(f"Memory usage: RSS={mem_info.rss / 1024**2:.2f} MB,
VMS={mem_info.vms / 1024**2:.2f} MB")

logger.debug("Loading character mapping...")
log_memory_usage()
mapping_file = os.path.join(data_dir, "char_mapping_256x256.npy")
char_mapping = np.load(mapping_file, allow_pickle=True).item()
num_classes = len(char_mapping) + 1
label_to_char = char_mapping
char_to_label = {v: k for k, v in label_to_char.items()}
logger.debug(f"Loaded {num_classes-1} characters + 1 blank class")

logger.info("Loading labels.csv...")
```

```
df = pd.read_csv(label_file)
logger.info(f"Loaded {len(df)} images")
log_memory_usage()

def pad_sequence(seq):
    try:
        seq = eval(seq)
        if not all(isinstance(l, int) and 0 <= l < num_classes-1 for l in seq):
            raise ValueError(f"Invalid sequence labels: {seq}")
        return seq + [num_classes-1] * (max_seq_length - len(seq))
    except Exception as e:
        logger.error(f"Error parsing sequence: {seq}, Error: {e}")
        raise

df['padded_labels'] = df['sequence_labels'].apply(pad_sequence)
logger.info(f"Dataset prepared with padded labels (max length: {max_seq_length})")
log_memory_usage()

logger.debug("Splitting dataset...")
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
logger.info(f"Train size: {len(train_df)}, Test size: {len(test_df)}")

def load_image(image_path, label):
    logger.debug(f"Loading image: {image_path}")
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img, channels=1)
    img = tf.image.resize(img, image_size)
    img = img / 255.0
    return img, label

def augment_image(image, label):
    logger.debug("Applying augmentation")
    image = tf.clip_by_value(image, 0.0, 1.0)
    return image, label

def create_dataset(df, batch_size, shuffle=True, augment=False):
    logger.debug("Creating dataset...")
    labels = np.stack(df['padded_labels'].values)
    dataset = tf.data.Dataset.from_tensor_slices((df['image_path'].values, labels))
    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(df))
    dataset = dataset.map(load_image, num_parallel_calls=tf.data.AUTOTUNE)
    if augment:
        dataset = dataset.map(augment_image, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
    logger.debug("Dataset created")
    log_memory_usage()
    return dataset

train_dataset = create_dataset(train_df, batch_size, shuffle=True, augment=False)
```

```
test_dataset = create_dataset(test_df, batch_size, shuffle=False, augment=False)

logger.debug("Building model...")
model = models.Sequential([
    layers.Input(shape=(image_size[0], image_size[1], 1)),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.4),
    layers.Dense(max_seq_length * num_classes, activation=None),
    layers.Reshape((max_seq_length, num_classes)),
    layers.Softmax(axis=-1)
])

def multi_label_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.int32)
    losses = tf.keras.losses.sparse_categorical_crossentropy(y_true, y_pred,
from_logits=False)
    return tf.reduce_mean(losses, axis=-1)

logger.debug("Compiling model...")
model.compile(optimizer=legacy.Adam(learning_rate=0.0005, clipnorm=1.0),
              loss=multi_label_loss,
              metrics=['accuracy'])

log_dir = os.path.join(logs_dir, "tensorboard", datetime.now().strftime("%Y%m%d-
%H%M%S"))
csv_log_file = os.path.join(logs_dir, "training_log.csv")
checkpoint_path = os.path.join(checkpoint_dir,
"model_epoch_{epoch:02d}_val_loss_{val_loss:.4f}.keras")
history_file = os.path.join(logs_dir, "training_history.pkl")

callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=3, min_lr=1e-6),
    tf.keras.callbacks.ModelCheckpoint(
        checkpoint_path,
        monitor='val_loss',
        save_best_only=True,
        save_weights_only=False,
        verbose=1
    ),
```

```
    tf.keras.callbacks.CSVLogger(csv_log_file, append=False),
    tf.keras.callbacks.TensorBoard(
        log_dir=log_dir,
        histogram_freq=1,
        write_graph=True,
        write_images=True,
        update_freq='epoch',
        profile_batch=0
    )
]

logger.info("Starting training...")
try:
    history = model.fit(
        train_dataset,
        epochs=epochs,
        validation_data=test_dataset,
        callbacks=callbacks,
        verbose=1
    )
except Exception as e:
    logger.error(f"Training failed: {e}")
    raise

logger.info(f"Saving training history to {history_file}")
with open(history_file, 'wb') as f:
    pickle.dump(history.history, f)

logger.info(f"Saving training history to {history_file}")
def convert_to_serializable(obj):
    if isinstance(obj, np.floating):
        return float(obj)
    if isinstance(obj, np.ndarray):
        return obj.tolist()
    return obj

history_json_file = os.path.join(logs_dir, "training_history.json")
with open(history_json_file, 'w') as f:
    json.dump(history.history, f, indent=4, default=convert_to_serializable)

def evaluate_model(dataset, label_to_char):
    logger.debug("Evaluating model...")
    total, correct = 0, 0
    blank_class = num_classes - 1
    for images, labels in dataset:
        preds = model.predict(images, verbose=0)
        preds = np.argmax(preds, axis=-1)
        labels = labels.numpy()
        for i in range(images.shape[0]):
            pred_seq = [label_to_char.get(p, '?') for p in preds[i] if p !=
blank_class]
```

```
            true_seq = [label_to_char.get(l, '?') for l in labels[i] if l !=
blank_class]
            if ''.join(pred_seq) == ''.join(true_seq):
                correct += 1
            total += 1
    return 100. * correct / total

test_acc = evaluate_model(test_dataset, label_to_char)
logger.info(f"Test sequence accuracy: {test_acc:.2f}%")

model.save(os.path.join(data_dir, "urdu_ocr_sequence_model_256x256.keras"))
logger.info("Training complete. Model and history saved.")
```

6.seq_predict.py

```
import argparse
import os
import numpy as np
import tensorflow as tf
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

image_size = (256, 256)
max_seq_length = 4
data_dir = "/Users/moizmac/DataScienceProject/urdu_seq_dataset_256x256"
model_path = os.path.join(data_dir, "urdu_ocr_sequence_model_256x256.keras")
mapping_file = os.path.join(data_dir, "char_mapping_256x256.npy")

def load_char_mapping():
    logger.info("Loading character mapping...")
    char_mapping = np.load(mapping_file, allow_pickle=True).item()
    num_classes = len(char_mapping) + 1
    return char_mapping, num_classes

def preprocess_image(image_path):
    logger.info(f"Loading and preprocessing image: {image_path}")
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Image not found: {image_path}")
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img, channels=1)
    img = tf.image.resize(img, image_size)
    img = img / 255.0
    img = tf.expand_dims(img, axis=0)
    return img

def decode_prediction(pred, label_to_char, blank_class):
    pred = np.argmax(pred, axis=-1)[0]
    seq = [label_to_char.get(p, '?') for p in pred if p != blank_class]
    return ''.join(seq)
```

```
def main():
    parser = argparse.ArgumentParser(description="Predict text sequence from an im-
age.")
    parser.add_argument("image_path", type=str, help="Path to the input image")
    args = parser.parse_args()

    try:
        label_to_char, num_classes = load_char_mapping()
        blank_class = num_classes - 1

        logger.info(f"Loading model from {model_path}...")
        if not os.path.exists(model_path):
            raise FileNotFoundError(f"Model file not found: {model_path}")
        model = tf.keras.models.load_model(
            model_path,
            custom_objects={'multi_label_loss': lambda y_true, y_pred: y_true}
        )

        image = preprocess_image(args.image_path)

        logger.info("Making prediction...")
        prediction = model.predict(image, verbose=0)

        predicted_text = decode_prediction(prediction, label_to_char, blank_class)
        logger.info(f"Predicted text: {predicted_text}")

        print(f"Predicted text: {predicted_text}")

    except Exception as e:
        logger.error(f"Error during prediction: {e}")
        raise

if __name__ == "__main__":
    main()
```

7.overfit_reduction.py

```
import os
import psutil
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import legacy
from sklearn.model_selection import train_test_split
import logging
import pickle
import json
from datetime import datetime

image_size = (256, 256)
```

```
batch_size = 64
epochs = 15
max_seq_length = 4

data_dir = "/Users/moizmac/DataScienceProject/urdu_seq_dataset_256x256"
label_file = os.path.join(data_dir, "labels.csv")
logs_dir = os.path.join(data_dir, "logs_reduced_overfit")
checkpoint_dir = os.path.join(data_dir, "checkpoints_reduced_overfit")
os.makedirs(logs_dir, exist_ok=True)
os.makedirs(checkpoint_dir, exist_ok=True)

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

def log_memory_usage():
    process = psutil.Process()
    mem_info = process.memory_info()
    logger.debug(f"Memory usage: RSS={mem_info.rss / 1024**2:.2f} MB")

mapping_file = os.path.join(data_dir, "char_mapping_256x256.npy")
char_mapping = np.load(mapping_file, allow_pickle=True).item()
num_classes = len(char_mapping) + 1
label_to_char = char_mapping
char_to_label = {v: k for k, v in label_to_char.items()}

df = pd.read_csv(label_file)

def pad_sequence(seq):
    seq = eval(seq)
    return seq + [num_classes - 1] * (max_seq_length - len(seq))

df['padded_labels'] = df['sequence_labels'].apply(pad_sequence)
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

def load_image(image_path, label):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img, channels=1)
    img = tf.image.resize(img, image_size)
    img = img / 255.0
    return img, label

def augment_image(image, label):
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, lower=0.9, upper=1.1)
    image = tf.clip_by_value(image, 0.0, 1.0)
    return image, label

def create_dataset(df, batch_size, shuffle=True, augment=False):
    labels = np.stack(df['padded_labels'].values)
    dataset = tf.data.Dataset.from_tensor_slices((df['image_path'].values, labels))
```

```
    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(df))
    dataset = dataset.map(load_image, num_parallel_calls=tf.data.AUTOTUNE)
    if augment:
        dataset = dataset.map(augment_image, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return dataset

train_dataset = create_dataset(train_df, batch_size, augment=True)
test_dataset = create_dataset(test_df, batch_size, augment=False)

model = models.Sequential([
    layers.Input(shape=(image_size[0], image_size[1], 1)),
    layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(max_seq_length * num_classes),
    layers.Reshape((max_seq_length, num_classes)),
    layers.Softmax(axis=-1)
])

def multi_label_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.int32)
    losses = tf.keras.losses.sparse_categorical_crossentropy(y_true, y_pred)
    return tf.reduce_mean(losses, axis=-1)

model.compile(
    optimizer=legacy.Adam(learning_rate=0.0005, clipnorm=1.0),
    loss=multi_label_loss,
    metrics=['accuracy']
)

log_dir = os.path.join(logs_dir, "tensorboard", datetime.now().strftime("%Y%m%d-
%H%M%S"))
csv_log_file = os.path.join(logs_dir, "training_log.csv")
checkpoint_path = os.path.join(checkpoint_dir,
"model_epoch_{epoch:02d}_val_loss_{val_loss:.4f}.keras")

callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=3, min_lr=1e-6),
```

```
    tf.keras.callbacks.ModelCheckpoint(
        checkpoint_path,
        monitor='val_loss',
        save_best_only=True,
        save_weights_only=False,
        verbose=1
    ),
    tf.keras.callbacks.CSVLogger(csv_log_file),
    tf.keras.callbacks.TensorBoard(log_dir=log_dir)
]

history = model.fit(
    train_dataset,
    epochs=epochs,
    validation_data=test_dataset,
    callbacks=callbacks,
    verbose=1
)

model.save(os.path.join(data_dir, "urdu_ocr_sequence_model_reduced_overfit.keras"))
```

8.overfit_prediction.py

```
import tensorflow as tf
import numpy as np
import os
import logging

image_size = (256, 256)
max_seq_length = 4
data_dir = "/Users/moizmac/DataScienceProject/urdu_seq_dataset_256x256"
model_path = os.path.join(data_dir, "urdu_ocr_sequence_model_reduced_over-
fit.keras")
mapping_file = os.path.join(data_dir, "char_mapping_256x256.npy")

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

char_mapping = np.load(mapping_file, allow_pickle=True).item()
num_classes = len(char_mapping) + 1
label_to_char = char_mapping

logger.info("Loading model...")
model = tf.keras.models.load_model(
    model_path,
    custom_objects={'multi_label_loss': lambda y_true, y_pred:
tf.keras.losses.sparse_categorical_crossentropy(y_true, y_pred)}
)

def load_image(image_path):
    try:
```

```
        img = tf.io.read_file(image_path)
        img = tf.image.decode_png(img, channels=1)
        img = tf.image.resize(img, image_size)
        img = img / 255.0
        return img
    except Exception as e:
        logger.error(f"Error loading image {image_path}: {e}")
        return None

def predictions_to_text(predictions, label_to_char, num_classes):
    pred_labels = np.argmax(predictions, axis=-1)
    text = ''
    for label in pred_labels[0]:
        if label < num_classes - 1:
            text += label_to_char.get(label, '')
    return text

def predict_image(image_path):
    if not os.path.exists(image_path):
        logger.error(f"Image path {image_path} does not exist.")
        return None

    img = load_image(image_path)
    if img is None:
        return None

    img = tf.expand_dims(img, axis=0)

    logger.info("Making prediction...")
    predictions = model.predict(img)

    predicted_text = predictions_to_text(predictions, label_to_char, num_classes)
    return predicted_text

def main():
    while True:
        image_path = input("Enter the image path (or 'quit' to exit): ").strip()
        if image_path.lower() == 'quit':
            logger.info("Exiting...")
            break

        predicted_text = predict_image(image_path)
        if predicted_text is not None:
            logger.info(f"Predicted text: {predicted_text}")
        else:
            logger.info("Please try another image path.")

if __name__ == "__main__":
    main()
```