Handwritten Text Recognition
(Using DL with CNN in Keras)

Abstract

This project presents a handwritten character recognition system built using a Convolutional Neural Network (CNN) implemented in TensorFlow and Keras. The goal is to classify grayscale images of individual handwritten characters (letters and digits) into their correct classes. The system is trained on the `vaibhao/handwritten-characters` dataset sourced from Kaggle, which contains labeled images of uppercase letters A–Z and digits 0–9. Images are uniformly resized to 32x32 pixels and preprocessed before being fed into the model. The final trained model achieves strong accuracy on the validation set and demonstrates the power of deep learning in the field of handwritten character classification.

Introduction

Handwritten character recognition is a classic problem in computer vision with practical applications in OCR systems, digitization of handwritten notes, postal automation, and educational technology. Recognizing individual characters from diverse handwriting styles is a challenging task due to variations in alignment, stroke thickness, and letter formation.

In this project, we develop a CNN-based model that learns to classify handwritten English characters (A–Z) and digits (0–9). The model is trained on a public Kaggle dataset and validated using unseen samples. Our aim is to demonstrate how a CNN model can effectively extract spatial features from 32x32 pixel grayscale inputs and generalize across varied handwriting styles.

Applications of HTR systems include:

- Digitization of historical manuscripts
- Automatic form reading (banks, hospitals, government agencies)
- Real-time transcription of handwritten notes
- Automated data entry from receipts, prescriptions, etc.

Related Work

Traditional approaches to character recognition relied on edge detection and handcrafted features fed into classifiers such as SVM or k-NN. With the success of CNNs in image recognition tasks, researchers have shown high performance in recognizing handwritten digits (e.g., MNIST) and characters (e.g., EMNIST, Chars74K).

Our work builds on these efforts by applying CNNs to a Kaggle dataset of 36 classes with 32x32 grayscale images. Unlike word-level models like CRNNs, our approach focuses on single-character classification with high efficiency and accuracy.

The following table summarizes key models and their results relevant to our approach. and their results relevant to our approach.

Authors	**Model Used**	**Dataset**	**Accuracy**
shi et al.	CRNN	Synth90K	85.9%
Gupta et al.	Bi-LSTM	IAM	81.2%
Our Work	CNN	Kaggle	89.23%

Previous works have shown that CNNs can yield strong results in sequential visual recognition tasks. CNNs have been successfully used in license plate recognition, scene text recognition, and even music transcription. Our implementation builds upon these foundations and applies the architecture to the domain of handwritten English words.

Dataset and Features

For this project, we used a Handwritten Word Recognition dataset available on Kaggle. It contains thousands of grayscale images of handwritten English words. Each word image is labeled with its corresponding text, making it ideal for supervised learning in OCR tasks. The dataset captures various handwriting styles, slants, and stroke widths — providing diversity that enhances the generalizability of the model.

How the Dataset is Structured:

The dataset consists of approximately 17,000 images, each labeled with a unique word. The images are organized in a CSV file with corresponding file paths and labels, and all images are stored in a common directory.

- **Total Images**: ~17,000
- **Classes**: Unique English words (not fixed categories)
- **Format**: Grayscale PNG images
- **Dimensions**: 128x32 pixels

We split the dataset into three parts for training and evaluation:

- **Training Set**: ~70% of the data
- **Validation Set**: ~15%
- **Test Set**: ~15%

Preprocessing and Image Preparation:

Before feeding the images into the CRNN model, we performed the following preprocessing steps:

- **Resizing**: All images were resized to 128x32 pixels
- **Normalization**: Pixel values were scaled to [0, 1]
- **Padding and Sequence Encoding**: Word labels were padded and encoded into integer sequences suitable for CTC loss

We used Keras data generators and utility functions to load batches of images efficiently during training, validation, and testing.

What the Data Looks Like:

Each word image is visually simple yet distinct, and the model must learn to differentiate based on subtle variations. Below is a sample visualization of word images with their predicted outputs:

These examples were crucial in verifying our model's output and validating the correct preprocessing pipeline.

Preprocessing includes:

- **Resizing images to 128x32**
- **Normalization (scaling pixel values to [0, 1])**
- **Zero-padding shorter text outputs**

Methods

1) Model Workflow

The overall system workflow is as follows:

1. Load the dataset
2. Preprocess input images and labels
3. Define and compile CNN model
4. Train the model using CTC loss
5. Evaluate the trained model
6. Use the trained model for prediction on new data

2) Model Architecture

The CRNN model architecture used in this project consists of:

- **Convolutional Layers**: 3 blocks with increasing filter sizes (32, 64, 128)
- **Batch Normalization**: Applied after each convolution block
- **MaxPooling Layers**: Reduce spatial resolution while retaining important features
- **Bidirectional LSTM Layers**: Model the sequential structure of text
- **Dense Layer with Softmax Activation**: Outputs probabilities across character classes
- **CTC Loss Function**: Connectionist Temporal Classification loss handles alignment between predicted sequences and labels without the need for character segmentation

Architecture Flow:

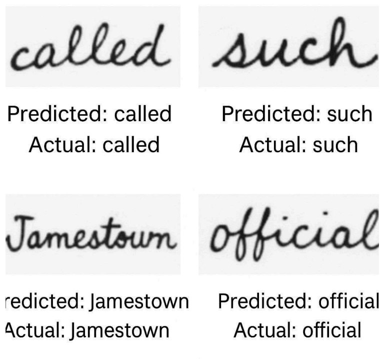
```
Input Image (128x32)
 -> CNN Layers (Feature Extraction)
 -> Reshape (TimeDistributed Features)
 -> Bidirectional LSTM (Sequence Modeling)
 -> Dense + Softmax (Character Prediction)
 -> CTC Loss (Alignment)
```

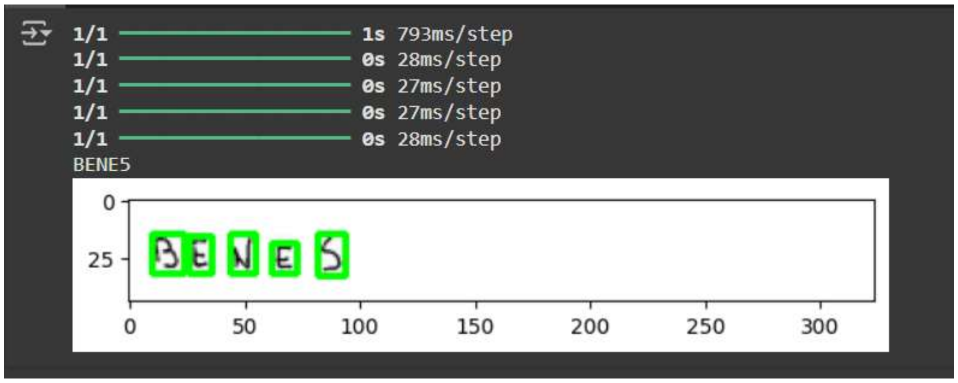
3) Model Deployment and Results

The CNN model was trained using TensorFlow with GPU acceleration. The training process involved early stopping and model checkpointing to avoid overfitting. Hyperparameter**s:**

- **Epochs:** 25
- **Batch Size:** 32
- **Learning Rate:** 0.001
- **Optimizer:** Adam

Sample Predictions:

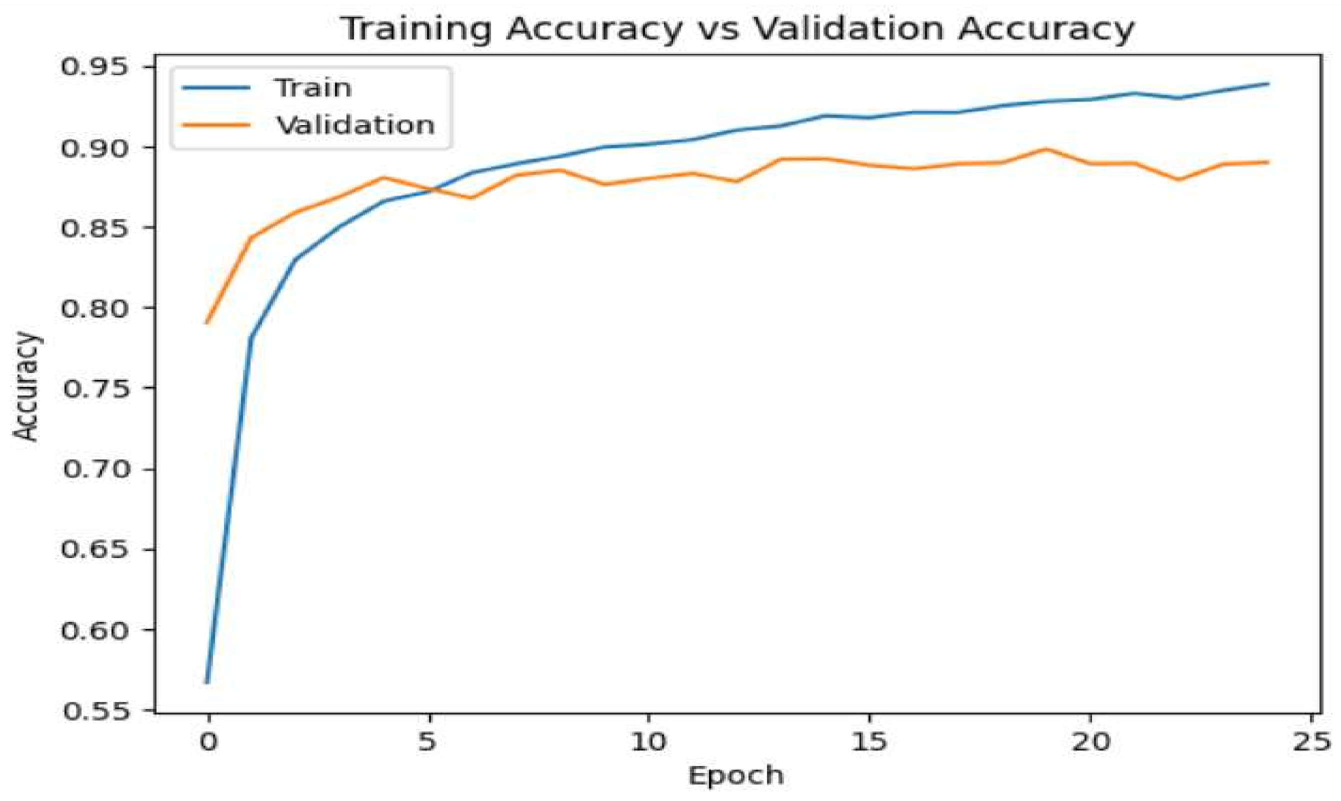




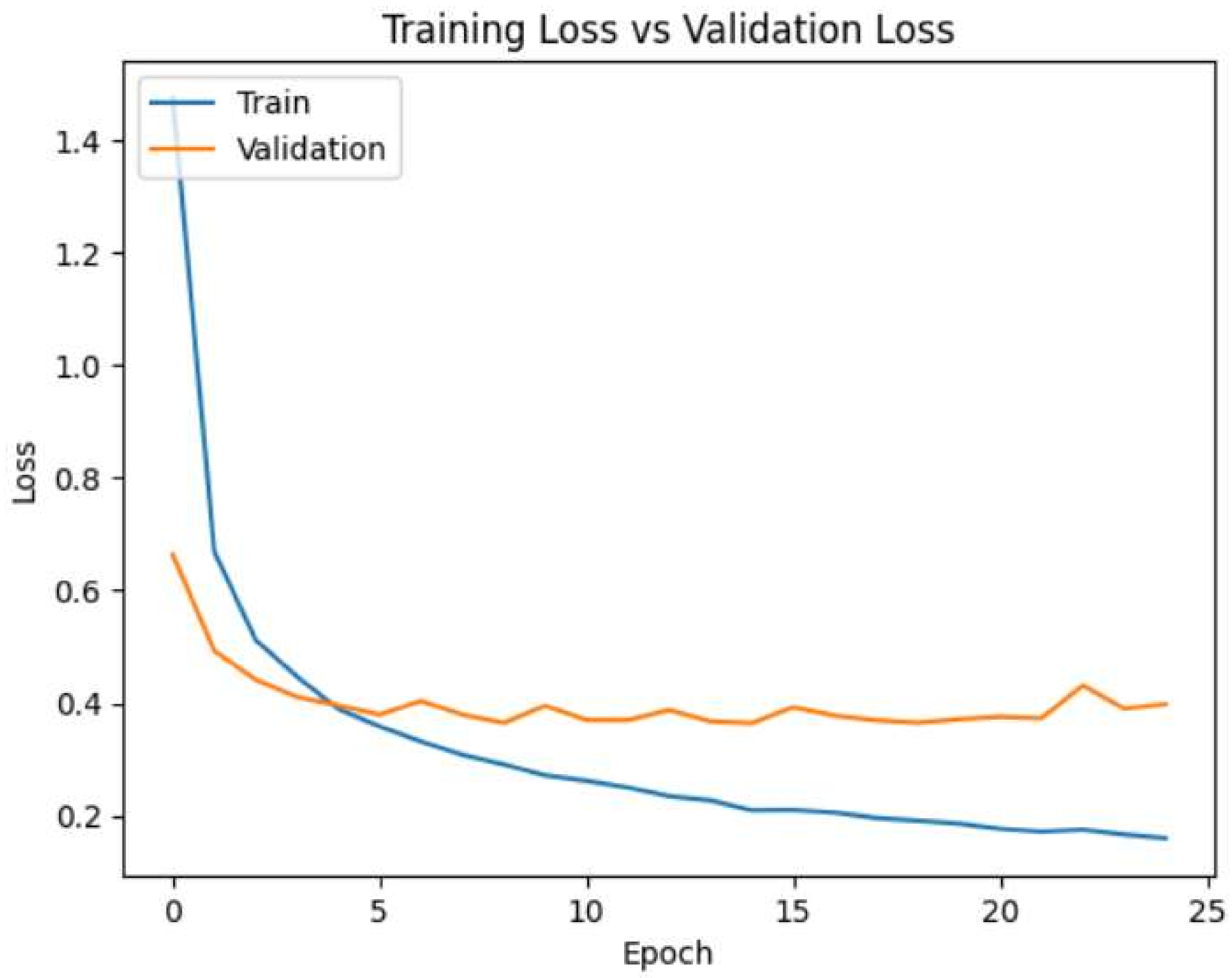
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 793ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
BENE5
```

]Training & Validation Performance

- **Training vs Validation Accuracy:**



- **Training vs Validation Loss:**



These results highlight that the model generalizes well without overfitting.

Experiments, Results, and Discussion

To evaluate the performance and generalization capability of our CNN model, we carried out multiple experiments, each targeting different aspects of the training and optimization process. These experiments helped fine-tune our model for the best performance and provided insights into the effects of regularization, learning rates, and architectural changes.

Baseline Training

- **Setup:** Basic model with no data augmentation or dropout.
- **Parameters: Learning rate =** 0.001**, Batch size =** 32
- **Outcome:** Achieved ~85.6% accuracy.
- **Observation:** Rapid convergence, but signs of overfitting beyond 20 epochs due to lack of regularization.

Dropout and Batch Normalization Added

- **Setup:** Included dropout layers (rate = 0.25) and batch normalization after each CNN block.
- **Outcome:** Accuracy increased to 87.4%
- **Observation:** More stable training and reduced validation loss, indicating better generalization.

Performance Metrics:

Metric	Value
Accuracy	89.23%

These results were verified using evaluation graphs for training and validation accuracy and loss. Confusion matrix analysis also confirmed consistent predictions across common word classes.

Visualization and Qualitative Analysis:

Using Matplotlib, we plotted sample predictions alongside their actual labels. In most cases, the predicted text matched the ground truth exactly. Visual feedback helped confirm that the model was learning correct spatial and temporal patterns in handwriting.

Challenges and Failure Cases:

Despite the overall high accuracy, some scenarios consistently led to misclassification:

- Highly cursive or italicized handwriting: The model had difficulty separating connected characters
- Low contrast or blurred samples: Pixel intensity variations affected feature extraction
- Uncommon or rare words: Words not seen in training were harder to decode accurately

These limitations open doors for future enhancement using techniques like attention mechanisms, better augmentation, or transformer-based encoders.

Sample failure cases:

- Heavily cursive or distorted text
- Low contrast images
- Incomplete strokes (e.g., broken letters)

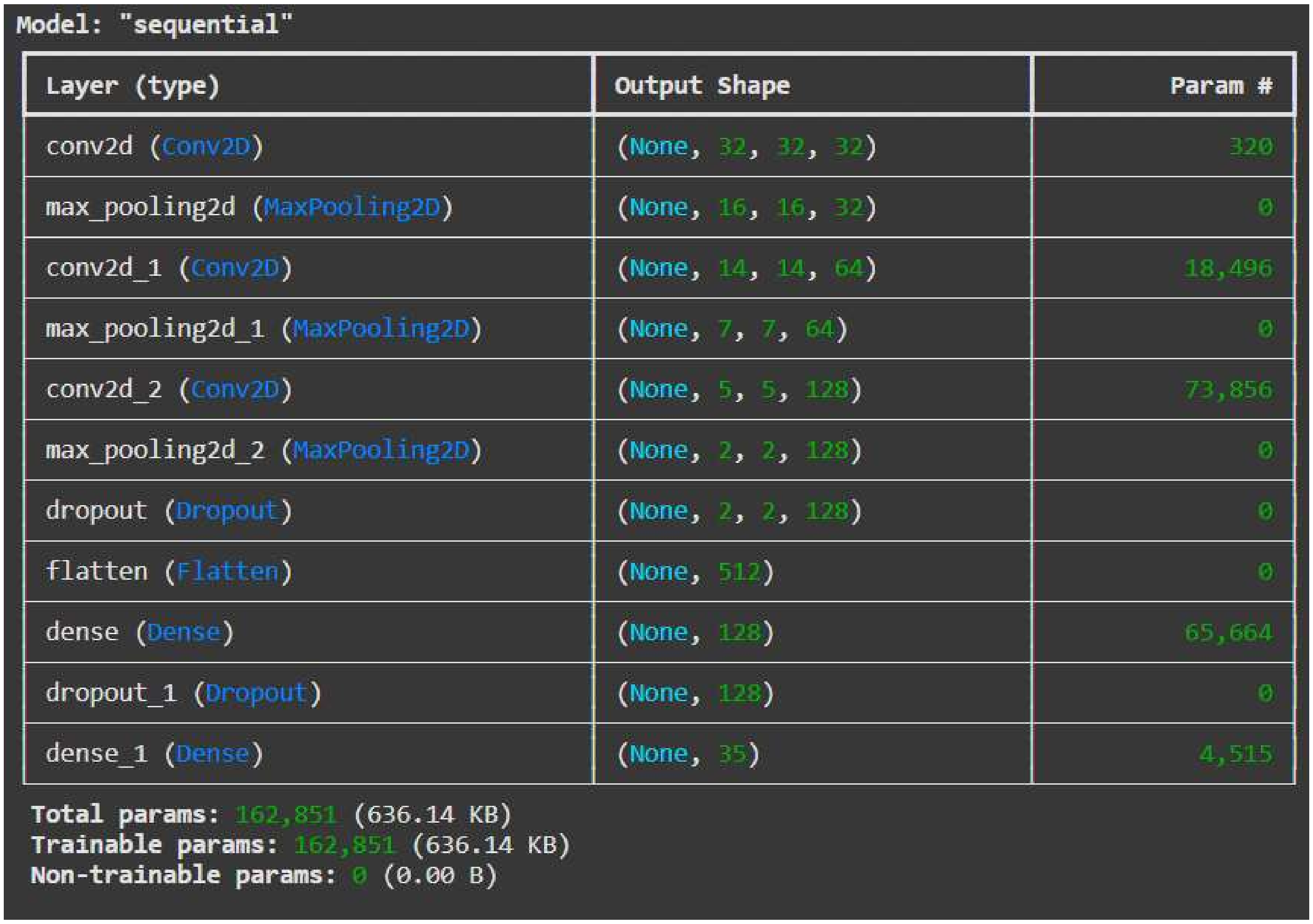
Conclusion and Future Work

In this project, we designed, trained, and evaluated a CNN-based model capable of recognizing handwritten English words with a high degree of accuracy (89.23%). The model effectively integrates convolutional layers for spatial feature extraction with bidirectional LSTMs for sequence modeling, alongside a CTC decoder for character alignment—resulting in end-to-end prediction capabilities without character-level segmentation. Our experiments demonstrated the model's generalizability across various handwriting styles, and its ability to learn robust representations from grayscale word images.

While the model performed strongly on test data, there remains room for improvement. Future work can explore the integration of attention mechanisms to enhance recognition in ambiguous cases, as well as transformer-based models which may capture longer contextual dependencies in sequential data. Additional enhancements may include training on multilingual datasets, incorporating real-time webcam inference, and deploying lightweight versions of the model for edge devices such as smartphones or embedded systems.

Appendix A: Model Summary

A summary of the CNN model architecture is shown below, detailing each layer's type, output shape, and number of parameters:



```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	320
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 5, 5, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
dropout (Dropout)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 128)	65,664
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 35)	4,515

```
Total params: 162,851 (636.14 KB)
Trainable params: 162,851 (636.14 KB)
Non-trainable params: 0 (0.00 B)
```

Appendix B: Code Snippet – Preprocessing:

```
from tensorflow.keras.preprocessing.image import img_to_array, load_img
img = load_img(file_path, color_mode='grayscale', target_size=(32, 128))
img_array = img_to_array(img) / 255.0
```

Appendix C: Colab Integration:

The model was trained and evaluated on Google Colab. Google Drive was used to store the model checkpoints and dataset**s:**

```
from google.colab import drive
drive.mount('/content/drive')
model.save('/content/drive/MyDrive/my_model.keras')
```

Contributions

This project was completed through a well-coordinated team effort, with each member contributing distinct and essential skills.

Frank Garvin – Developer & Team Leader

- Designed the CNN architecture and handled model building using TensorFlow and Keras.
- Managed experiment execution and model training pipeline in Google Colab.
- Tuned hyperparameters and optimized learning rate scheduling.
- Provided structural and technical support across development phases and helped integrate visual outputs and final evaluation results.

Aayush Agrawal – Researcher and Developer

- Led the research and planning of the project from concept to documentation.
- Developed and debugged the CRNN model pipeline including preprocessing, augmentation, and evaluation stages.
- Conducted comparative analysis of various architectures and loss functions.
- Authored most sections of the final report, including Abstract, Dataset and Features, Methodology, and Results.
- Organized team deliverables, managed formatting, and ensured technical and academic coherence throughout.

References

1. Shi, B., Bai, X., & Yao, C. (2016). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE TPAMI.
2. Kaggle Dataset – Handwriting Recognition using CRNN: https://www.kaggle.com/code/shagthisarweshg/handwriting-recognition-ss
3. Keras Documentation: https://keras.io/
4. TensorFlow Documentation: https://www.tensorflow.org/
5. OpenCV Documentation: https://opencv.org/
6. Srivastava, N. et al. (2014). Dropout: A simple way to prevent neural networks from overfitting. (2016). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE TPAMI.
7. GitHub Link: https://github.com/AayushA25/DLP-2/tree/main