Deep Learning Approach For Scene Text Recognition

Umanshiva Ladva
IIT HYDERABAD
ai22btech11016@iith.ac.in

Rajiv Chaudhary
IIT HYDERABAD
ai22btech11021@iith.ac.in

Siddhesh Gholap
IIT HYDERABAD
ai22btech11007@iith.ac.in

Sudarshan Shivashankar
IIT HYDERABAD
ai22btech11027@iith.ac.in

Ruvva Suraj Kumar
IIT HYDERABAD
ai22btech11022@iith.ac.in

1. Abstract

Scene text recognition is an essential task in computer vision with wide-ranging applications, from autonomous driving and augmented reality to digitizing documents and assisting visually impaired individuals.This project explores a deep learning-based solution for scene text recognition using a Convolutional Recurrent Neural Network (CRNN) architecture [1]. The hybrid model integrates CNNs for spatial feature extraction with Bidirectional Long Short-Term Memory (BiLSTM) networks for sequence modeling, and employs a lexicon-free transcription layer to handle diverse OCR tasks. Using synthetic datasets for training and benchmark datasets (IIIT5K) for testing, our model demonstrates robust text recognition capabilities across varied real-world scenarios. The report highlights dataset preprocessing techniques, model architecture, training procedures, and performance benchmarks,

2. Introduction

Scene text recognition is a vital task in computer vision, enabling machines to interpret textual information embedded in natural images. Unlike traditional optical character recognition (OCR), scene text recognition presents unique challenges such as varying text orientations, complex backgrounds, and distortions. To address these, our project employs a Convolutional Recurrent Neural Network (CRNN) architecture for an end-to-end trainable deep learning solution.

The CRNN framework is well-suited for this task because it combines convolutional layers to extract robust spatial features from images and recurrent layers to model sequential dependencies, making it ideal for handling the irregular and variable nature of scene text. By utilizing the Connectionist Temporal Classification (CTC) loss, the model bypasses the need for precise character-level annotations, further streamlining the training process.

We trained the model on a large synthetic dataset, which provides diverse and realistic text samples at scale. Synthetic datasets are advantageous as they enable extensive training without the cost of manually annotating real-world data. This approach ensures the model learns a wide range of text variations and generalizes effectively to unseen data.

Our project modifies the CRNN architecture to accept variable-length text inputs of up to 24 characters, enhancing its flexibility for practical applications. Testing on the IIIT5K dataset highlights the model's ability to accurately recognize text in complex scenes, demonstrating the robustness of the CRNN framework and the efficacy of using synthetic data for training.

3. Dataset Collection

3.1. Synthetic Dataset (Training Set)

This dataset was generated specifically for OCR tasks and contains around 12,000 images which is a subset of dataset used in [2]. Each image includes various fonts, sizes, and backgrounds to simulate real-world conditions, making the model robust to different text variations. Every image is associated with a ground truth label, which provides the correct transcription of the text in the image.

3.2. IIIT5K (Testing Set)

The IIIT5K dataset contains 5,000 images used for scene text recognition benchmarks. It includes a wide variety of text types such as street signs and book pages.

3.3. Data Preprocessing

To ensure that the datasets are compatible with the deep learning models, we applied several preprocessing techniques to enhance the quality of the images and prepare them for the recognition task:

- **Grayscale Conversion** Color images were converted to grayscale to reduce computational complexity and focus

on the essential features of text

- **Gaussian Blurring** A slight Gaussian blur was applied to the images to remove noise while preserving the text's sharpness. This helps the CNN focus on important structures such as character boundaries.
- **Resize** Every image was converted into 32x100 (height x width)
- **Normalization** The pixel values were normalized to a mean of 0.5 and a standard deviation of 0.5. This standardization improves convergence during model training by stabilizing gradients.

4. Breakdown of CRNN Architecture

The CRNN (Convolutional Recurrent Neural Network) architecture integrates CNNs and RNNs to effectively extract spatial features and process sequential data. CNNs excel at feature extraction for image classification but require RNNs for handling sequential data crucial for tasks like text recognition.

This hybrid approach enables:

- Feature extraction from images to provide structured inputs for RNNs.
- Sequential modeling of multi-character sequences using Bi-LSTMs.
- Robustness against noise and irregularities in real-world images.

Our updated CRNN design accommodates variable-length text inputs (up to 24 characters) and leverages CTC loss for efficient sequence prediction, allowing end-to-end training with word-level annotations.

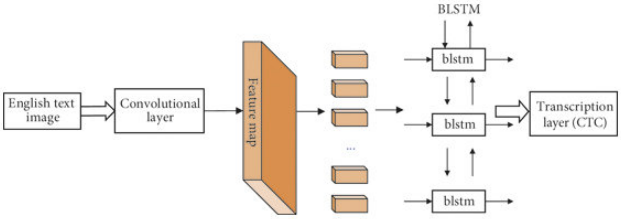


Figure 1. RCNN Architecture summary

4.1. CNN Module

The CNN[3] [4] module operates on the input image, producing feature maps that capture spatial characteristics such as edges, textures, and shapes. These feature maps are reshaped and fed into the RNN for sequential processing.

Key Features:

- Extracts spatial features from the image.
- Provides translation invariance and local feature encoding.
- Prepares sequential feature maps for RNN processing.

4.1.1 CNN Architecture

The CNN consists of 5 convolutional blocks, each designed to progressively extract features while preserving the spatial hierarchy. The architecture is as follows:

Block 1:

- 64 filters, with a 3×3 kernel.
- ReLU activation to introduce non-linearity.
- Max-pooling with a 2×2 kernel and stride 2×2 to reduce the spatial dimensions.

Block 2:

- 128 filters, with a 3×3 kernel.
- ReLU activation.
- Max-pooling with a 2×2 kernel and stride 2×2, further reducing spatial dimensions.

Block 3:

- 256 filters with a 3×3 kernel, followed by another 256-filter convolutional layer with a 3×3 kernel.
- ReLU activations after both convolutional layers.
- Max-pooling with a 2×1 kernel and stride 2×1, which reduces the height of the feature maps while retaining sequence width.

Block 4:

- 512 filters, with a 3×3 kernel, followed by Batch Normalization and ReLU activation.
- Another 512-filter convolutional layer with a 3×3 kernel, followed by Batch Normalization and ReLU activation.
- Max-pooling with a 2×1 kernel and stride 2×1 to further reduce spatial dimensions.

Block 5:

- A single 512-filter convolutional layer with a 2×2 kernel and stride 1.
- ReLU activation to introduce non-linearity.

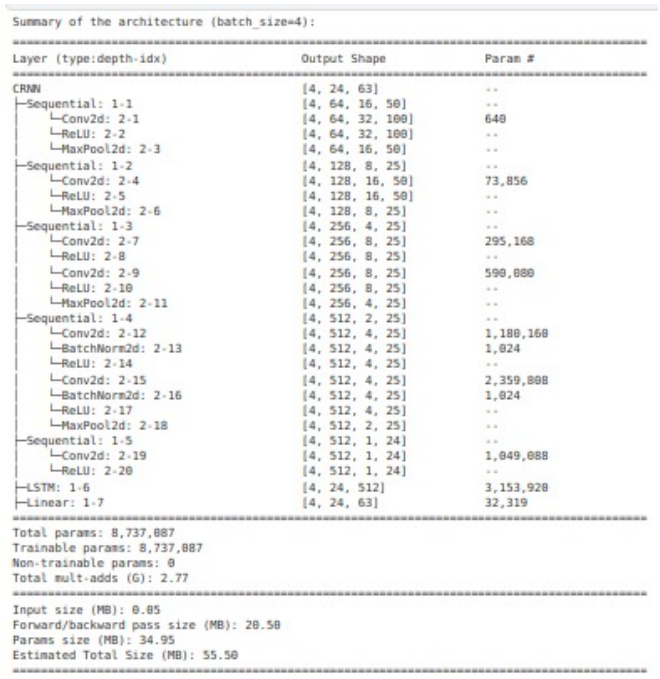
The CNN output is reshaped to a sequence of feature vectors for RNN input.

4.1.2 Configuration

- **Input Image Size:** 32×100 (height $\times$ width).
- **Convolution Parameters:** 3×3 filters, stride = 1, padding = 1.
- **Max-Pooling:** Vertical pooling (2×1) ensures sequence width is preserved.

4.2. RNN Module (Sequence Modeling)

The RNN module employs a Bidirectional LSTM (Bi-LSTM) to process the sequential features extracted by the CNN. Bi-LSTMs capture temporal dependencies in both forward and backward directions, improving recognition accuracy by considering context from surrounding characters.



```
Summary of the architecture (batch_size=4):
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CRNN                                     [4, 24, 63]               --
├─Sequential: 1-1                        [4, 64, 16, 50]           --
│    └─Conv2d: 2-1                       [4, 64, 32, 100]          640
│    └─ReLU: 2-2                         [4, 64, 32, 100]          --
│    └─MaxPool2d: 2-3                    [4, 64, 16, 50]           --
├─Sequential: 1-2                        [4, 128, 8, 25]           --
│    └─Conv2d: 2-4                       [4, 128, 16, 50]          73,856
│    └─ReLU: 2-5                         [4, 128, 16, 50]          --
│    └─MaxPool2d: 2-6                    [4, 128, 8, 25]           --
├─Sequential: 1-3                        [4, 256, 4, 25]           --
│    └─Conv2d: 2-7                       [4, 256, 8, 25]           295,168
│    └─ReLU: 2-8                         [4, 256, 8, 25]           --
│    └─Conv2d: 2-9                       [4, 256, 8, 25]           590,080
│    └─ReLU: 2-10                        [4, 256, 8, 25]           --
│    └─MaxPool2d: 2-11                   [4, 256, 4, 25]           --
├─Sequential: 1-4                        [4, 512, 2, 25]           --
│    └─Conv2d: 2-12                      [4, 512, 4, 25]           1,180,160
│    └─BatchNorm2d: 2-13                 [4, 512, 4, 25]           1,024
│    └─ReLU: 2-14                        [4, 512, 4, 25]           --
│    └─Conv2d: 2-15                      [4, 512, 4, 25]           2,359,808
│    └─BatchNorm2d: 2-16                 [4, 512, 4, 25]           1,024
│    └─ReLU: 2-17                        [4, 512, 4, 25]           --
│    └─MaxPool2d: 2-18                   [4, 512, 2, 25]           --
├─Sequential: 1-5                        [4, 512, 1, 24]           --
│    └─Conv2d: 2-19                      [4, 512, 1, 24]           1,049,088
│    └─ReLU: 2-20                        [4, 512, 1, 24]           --
├─LSTM: 1-6                              [4, 24, 512]              3,153,920
├─Linear: 1-7                            [4, 24, 63]               32,319
==========================================================================================
Total params: 8,737,887
Trainable params: 8,737,887
Non-trainable params: 0
Total mult-adds (G): 2.77
==========================================================================================
Input size (MB): 0.05
Forward/backward pass size (MB): 20.50
Params size (MB): 34.95
Estimated Total Size (MB): 55.50
==========================================================================================
```

Figure 2. RCNN Architecture summary

RNN Architecture:

- Two Bi-LSTM layers with 256 hidden units per direction (512 total per layer).
- Outputs are passed to the classifier for sequence prediction.

4.3. Transcription Layer

The transcription layer converts RNN predictions into label sequences using Connectionist Temporal Classification (CTC).

4.4. Greedy Decode

To decode the predictions made by the CRNN model, greedy decoding was employed. This method involves selecting the character with the highest probability at each time step. For each frame in the sequence, the model outputs a probability distribution over the character set (including a blank token).

4.5. Loss Function: Connectionist Temporal Classification (CTC)

CTC loss [5] is ideal for sequence-to-sequence tasks where input and output lengths differ.

- Handles alignment between input frames and output labels.
- Computes the most likely label sequence, accommodating variable-length text.

Working :

- Introduces blank tokens to manage skipped positions.
- Calculates alignments to maximize the likelihood of the ground-truth sequence.

Key Features of CTC:

- Introduces a blank token to handle skipped timesteps.
- Aligns predicted and ground-truth sequences dynamically.
- Collapses repetitive and blank predictions into a compact sequence (e.g., "-hh-ee-ll-oo-" $\rightarrow$ "hello").

This approach enables recognition of variable-length sequences without requiring pre-segmented data.

5. Training and Results

5.1. Training Setup

The training process for the CRNN model involved the following configurations:

- **Optimizer:** Adadelta, an adaptive learning optimizer, was used with a **learning rate of 0.1** to ensure stable updates.(Adadelta adapts the learning rate for each parameter based on the gradient and update magnitudes, without requiring manual learning rate specification. It is known for handling large learning rates and avoiding local minima.)
- **Loss Function:** The Connectionist Temporal Classification (CTC) loss was employed to handle the alignment of variable-length text sequences.
- **Batch Sizes:** Experiments were conducted with varying batch sizes $(4, 8, 16, 32, 64)$ to analyze their effect on performance.
- **Hardware:** Training was carried out on an NVIDIA GPU for accelerated computation.

5.2. Experimentation with Batch Sizes

To study the impact of batch size on training, experiments were conducted with batch sizes of $4, 8, 16, 32,$ and 64. Metrics such as training and validation loss, edit distance, and match accuracy were tracked for each batch size.

Edit Distance Edit distance measures the dissimilarity between predicted and actual text by calculating the minimum number of character edits (insertions, deletions, or substitutions) required to match the strings. A lower edit distance indicates better model performance. Batch size 4 exhibited the lowest edit distance, reflecting the best performance among the configurations.

Performance Metrics Table 1 and 2 summarizes the performance for each batch size:

Table 1. Training Results for Different Batch Sizes

Batch Size	Train Loss	Avg. Training Edit Distance	Avg. Training Match Accuracy
4	0.2046	0.5223	0.7716
8	0.2776	0.7014	0.7090
16	0.1784	0.4678	0.7798
32	0.2889	0.7244	0.6860
64	0.3106	0.7617	0.6682

Table 2. Validation Results for Different Batch Sizes

Batch Size	Validation Loss	Avg. Validation Edit Distance	Avg. Validation Match Accuracy
4	0.8397	1.6775	0.5038
8	0.7930	1.5747	0.4970
16	0.9389	1.6759	0.4689
32	1.0829	2.0439	0.3915
64	1.0366	2.0638	0.3794

5.3. Results Visualization

Loss vs. Epochs The training and validation loss graphs for various batch sizes demonstrate that:

- Batch size 4 consistently achieved the lowest validation loss across epochs.
- Larger batch sizes exhibited slower convergence and higher validation loss, indicating less effective learning.

Accuracy vs. Epochs Accuracy trends reveal:

- Batch size 4 achieved the highest validation match accuracy, confirming its superior performance.
- Accuracy tended to decline with larger batch sizes due to reduced generalization.

Edit Distance vs. Epochs The graphs for edit distance (training and validation) against epochs highlight:

- Batch size 4 exhibited the smallest average edit distance, reflecting the model's ability to make fewer errors in predicting the correct text.
- Larger batch sizes showed increased edit distance, indicating more prediction errors.

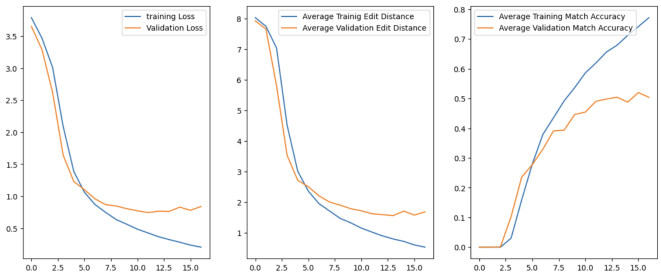


Figure 3. Batch Size 4

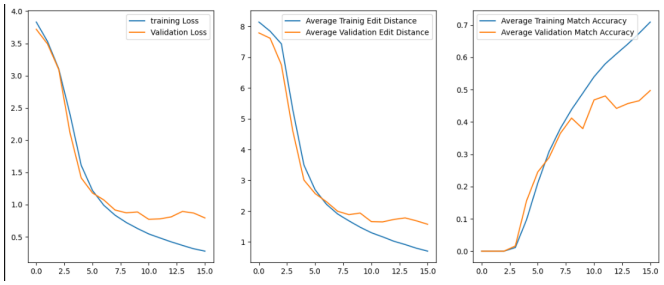


Figure 4. Batch Size 8

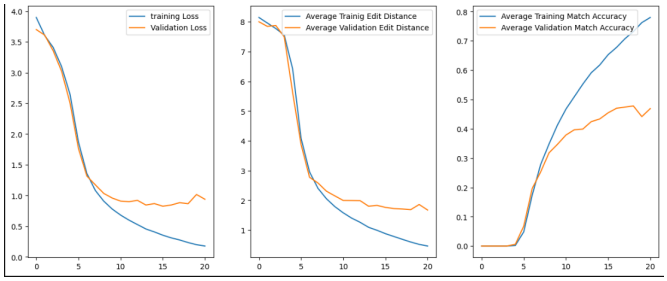


Figure 5. Batch Size 16

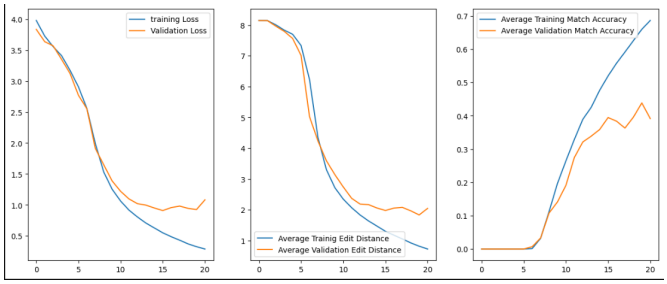


Figure 6. Batch Size 32

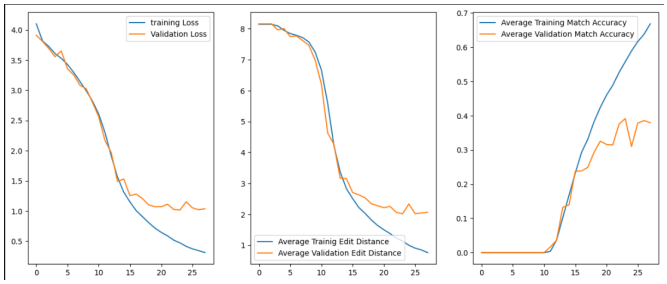


Figure 7. Batch Size 64

5.4. Testing and Evaluation

Testing was conducted using the **IIIT5K dataset**, a benchmark dataset containing 5,000 images with scene text and using batch size=4 which was the best performing one in the validation phase. The following steps outline the testing process:

1. **Model Prediction:** Images from the test set were passed through the trained CRNN model to generate predictions. Predictions were decoded using a greedy decoding strategy.
2. **Performance Metrics:** Edit Distance and Match Accu-

racy were computed to evaluate the model's ability to transcribe text accurately.

3. **Sample Evaluation:** Actual and predicted text strings were compared qualitatively to assess the model's robustness against noise and variations.

Sample Output For an input image, the model generated the following results:

- **Actual String**
- **Predicted String**

The model effectively captured most characters, with minor errors attributed to noise in the input image.

Figure 8. Result 1

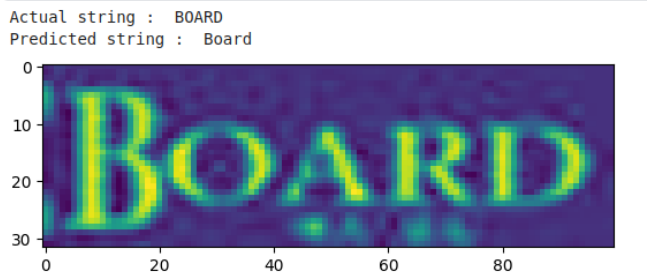


Figure 9. Result 2

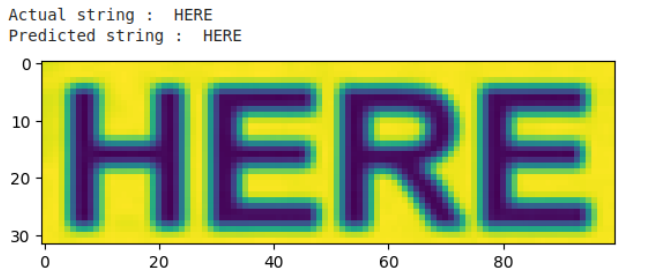


Figure 10. Result 3

6. Conclusion

In this project we successfully implemented a robust scene text recognition system using a CRNN architecture. By integrating CNNs for spatial feature extraction and Bi-LSTMs for sequence modeling, the model demonstrated significant efficacy in handling diverse and complex text recognition tasks. Leveraging synthetic datasets for training provided a scalable and effective means to enhance generalization, while benchmarking on IIIT5K validated the model's performance in real-world scenarios. Notably, the experiments revealed that smaller batch sizes yielded optimal results in terms of accuracy and edit distance, underlining the importance of thoughtful hyperparameter tuning in achieving superior model performance. This study highlights the potential of deep learning techniques to overcome the challenges inherent in scene text recognition, paving the way for further advancements in the field. Future work could explore enhancements such as attention mechanisms or transformer-based architectures to further improve accuracy and adaptability.

References

[1] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 11, pp. 2298–2304, 2016. 1

[2] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *arXiv preprint arXiv:1406.2227*, 2014. 1

[3] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," *International journal of computer vision*, vol. 116, pp. 1–20, 2016. 2

[4] Z. Liu, W. Zhou, and H. Li, "Scene text detection with fully convolutional neural networks," *Multimedia Tools and Applications*, vol. 78, pp. 18205–18227, 2019. 2

[5] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," pp. 369–376, 2006. 3