Typo Correction for RGB Character Image Sequence

Jaewon Lee
Seoul National Univ.
2019-12172
jaewon16@snu.ac.kr

Abstract

This paper presents the development and evaluation of sequence-to-sequence (Seq2Seq) models for the prediction of text sequences from image inputs. The project is divided into several key stages: data preparation, model development, training, and evaluation. Three types of datasets—training, validation, and challenge—are utilized, with inputs comprising sequences of images of varying lengths. The models leverage both Long Short-Term Memory (LSTM) networks and Transformer architectures, each enhanced with custom Convolutional Neural Networks (CNNs) for feature extraction. The study highlights the effectiveness of these models in handling different sequence lengths and complexities, employing techniques such as teacher forcing, hyperparameter tuning, and data augmentation to optimize performance. The experimental results demonstrate the superior performance of Transformer-based models in managing complex sequences, achieving a validation accuracy of 90.745% and a challenge test accuracy of 87.171%. These findings underscore the potential of advanced Seq2Seq models for practical applications in sequence prediction tasks.

1. Introduction

The final project aims to develop and evaluate a model capable of predicting sequences from given inputs, specifically focusing on text sequences. The overall process involves several key steps, including data preparation, model development, training and evaluation. This section provides an overview of each step, highlighting the critical aspects and methodologies employed.

1.1. Dataset

There are three types of datasets: training, validation, and challenge. Each dataset has input as sequences of images. Each image in the sequence has a shape of (28, 28, 3). The input shape is (t, 28, 28, 3), where t is the length of the sequence. t varies by the type of dataset. For the training and validation datasets, $4 \leq t \leq 8$. For the challenge dataset, $9 \leq t \leq 10$.

As the training dataset and challenge dataset have totally different shapes, when training for the challenge, data augmentation was used for training the model.

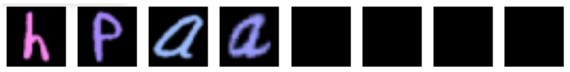

Figure 1. One example of training data. The sequence length is 4. Padding is added at the end because every sequence length in a batch must be the same. The target is "**hapa**".

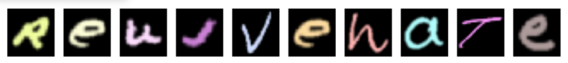

Figure 2. One example of challenge validation data. This data cannot be used for training. The sequence length is 10. The target is "**rejuvenate**".

1.2. Model Development

The core of the project is the development of the sequence-to-sequence (Seq2Seq) model. This model comprises an encoder-decoder architecture, which can be customized with various components such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers [4]. The encoder processes the input sequence into a fixed representation, which the decoder then transforms into the output sequence. The flexibility in choosing model components and hyperparameters allows for experimentation and optimization to achieve the best performance.

1.3. Training and Evaluation

Various strategies such as teacher forcing and hyperparameter tuning are employed to enhance the training process. The training is monitored using metrics like validation loss, ensuring that the model is learning effectively and improving over time.

The evaluation phase assesses the model's performance on a separate validation set. The primary metric used is accuracy, defined as the ratio of correctly predicted sequences to the total number of sequences. Both normal and challenge tasks are evaluated, with the latter involving more complex and unseen data.

2. Method

2.1. Normal Task

2.1.1 Model architecture

The Seq2Seq model architecture adopted for this project comprises two primary components: an encoder and a decoder [4]. The encoder processes the input sequence and converts it into a fixed-length context vector, which encapsulates the relevant information from the input. This context vector is then passed to the decoder, which generates the output sequence one token at a time. The architecture leverages Long Short-Term Memory (LSTM) networks for both the encoder and decoder to effectively capture long-range dependencies and mitigate the vanishing gradient problem typically associated with Recurrent Neural Networks (RNNs) [1].

To enhance the feature extraction capability of the model, a custom Convolutional Neural Network (CNN) based on the ResNet architecture is integrated into the encoder. This custom ResNet processes the input data, enabling the model to handle various types of sequence data, including images and text. The extracted features are then fed into the LSTM network, ensuring that the model benefits from both spatial and temporal information.

2.2. Hyperparameters

```
kwargs = {
    'hidden_dim': 360,
    'n_rnn_layers': 2,
    'rnn_dropout': 0.5,
    'embed_dim': 28,
    'customCNN': "CustomResnet"
}
```

`kwargs` are the hyperparameters for the model. The `hidden_dim` means the hidden dimension of LSTMs at both the encoder and decoder. `n_rnn_layers` means the number of LSTM layers at both the encoder and decoder. `rnn_dropout` is the dropout probability for encoder and decoder LSTMs. `embed_dim` is the input dimension for encoder and decoder LSTMs. `customCNN` means the CNN module at the encoder. CustomResnet is Resnet18 but with the last fc layer output dimension the same as `embed_dim`.

Other than model parameters, training was implemented with Adam optimizer, with learning rate 0.001. Also, learning rate scheduler was used to train. It decays learning rate by a factor of 0.1 after every 10 epochs in Normal task. In challenge task, it decays after every 15 epochs.

2.3. Challenge Task

2.3.1 Model Architecture

To tackle a more challenging task within this project, a Transformer-based Seq2Seq model was implemented. The Transformer architecture, known for its ability to handle long-range dependencies and parallelize computations more effectively than RNN-based models, comprises multi-head self-attention mechanisms and feed-forward layers [5]. The specific implementation included a custom ResNet for feature extraction in the encoder, followed by Transformer layers to process the extracted features. The decoder utilized Transformer layers to generate the output sequence, enhancing the model's capability to understand and generate complex sequences.

Various masks were used in the Transformer model. In the encoder, the 'src_key_padding_mask' was implemented. This mask identifies the paddings, which can be at the end of the input sequence. It is generated by taking the sequence length as a parameter. Any input longer than the sequence length is marked as 'True' in the mask.

In the decoder, a target key mask and target key padding mask were implemented. These are used to ensure that the decoder does not predict the output using future information.

2.3.2 Hyperparameters

```
kwargs = {
    'hidden_dim': 128,
    'n_rnn_layers': 2,
    'rnn_dropout': 0.1,
    'embed_dim': 28,
    'customCNN': "CustomResnet",
    'nhead': 4,
    'dim_feedforward': 2048
}
```

The hidden_dim hyperparameter is d_model or embed dimension for Transformers at both encoder and decoder. nhead is the number of heads in the transformer. dim_feedforward is the feedforward hidden dimension in the transformer. Other hyperparameters are similar to those from the LSTM Seq2Seq model..

Adam was used as the optimizer with a learning rate of 0.001. The learning rate scheduler, however, was changed. For details, see the experiment results in Subsection 3.2.

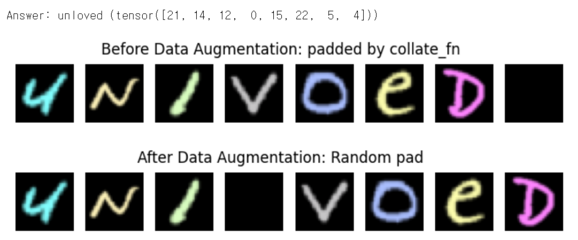
Answer: unloved (tensor([21, 14, 12, 0, 15, 22, 5, 4]))
Before Data Augmentation: padded by collate_fn
After Data Augmentation: Random pad

Figure 3. Example of data augmentation by reordering padding. The padding at the end of the sequence is moved to a random position (4th position) by data augmentation. The target of this image sequence is "unloved".

2.4. Data Augmentation

The original dataset is defined using the PyTorch Dataset class. The name of the dataset is *MLDataset*, where the original data is loaded.

Two types of data augmentation were implemented. The first involves transforming each image in the sequence. This was done by randomly rotating the image by up to 20 degrees, randomly cropping, and changing colors [2].

The second type of data augmentation involved changing the padding. In the original dataset, padding was always located at the end of the sequence by the `collate_fn()`. However, by reordering the paddings, without changing the order of non-padding images, data augmentation was applied. This can be seen in Figure 3.

These two types of data augmentation were implemented in another Dataset class called *trainMLDataset* because data augmentation should not be used during validation and evaluation.

```
train_ds = trainMLDataset(...)
valid_ds = MLDataset(...)
challenge_ds = MLDataset(...)
```

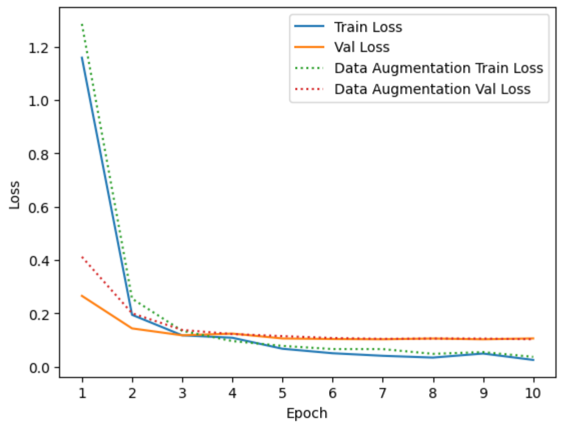

Figure 4. Training result for the original dataset and the augmented dataset.

	Original Dataset	Augmented Dataset
Validation Accuracy (%)	80.5	79.5

Table 1. Comparison of validation accuracy between the original dataset and the augmented dataset.

	Original Dataset	Augmented Dataset
Validation Accuracy (%)	87.9	75
Challenge Validation Accuracy (%)	31.3	65.3

Table 2. Comparison of validation accuracy between the original dataset and the augmented dataset in both normal and challenge tasks. The Transformer model was used in this table, whereas the LSTM model was used in Table 1.

‘

3. Experiments and Results

3.1. Normal Task

In the normal task, experiments were conducted to observe the effect of data augmentation. In Figure 4, it is shown that at the beginning of training, training without data augmentation performs better. This is because data augmentation makes the data more unpredictable, leading to harder convergence. However, as training progresses, the loss for the model trained with augmented data converges well, similar to the model trained with the original data. Nevertheless, the overall performance in the normal task was better when not using data augmentation.

3.2. Challenge Task

In the challenge task, data augmentation performed better, as shown in Table 2. This improvement is attributed to data augmentation making the model more robust to unseen data [3]. Therefore, for all other experiments in the challenge tasks, data augmentation was used.

In many cases during training, the validation loss is usually higher than the training loss, indicating poor generalization. To prevent overfitting, hyperparameters, in which the validation loss was smaller than the training loss were searched. By modifying hyperparameters, a hidden_dim of 128 was found to be the best for the first and second epochs. With the hidden_dim fixed, experiments were conducted by changing the nhead and dropout parameters. Figures 5, 6, and 7 show the training results.

All results use the Adam optimizer with a learning rate of 0.001. However, Figure 5 uses a learning rate scheduler with a decay of 10, while Figures 6 and 7 use a decay of 15. Figures 5, 6, and 7 all have hyperparameters of hidden_dim=128, n_rnn_layers=2, custom-

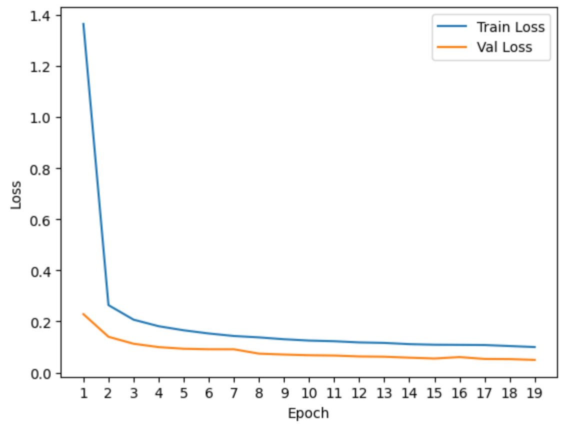

Figure 5. Training result of hyperparameters with rnn_dropout=0.2, nhead=2. Learning rate scheduler is set with decay 10.

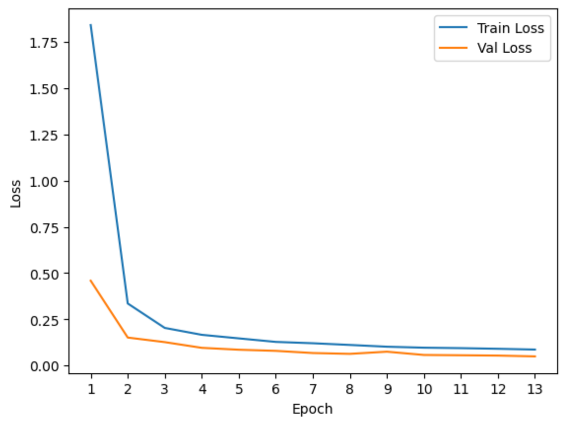

Figure 6. Training results for hyperparameters with rnn_dropout=0.2 and nhead=2. The learning rate scheduler is set with a decay of 10.

	Train Validation	Challenge Validation	Kaggle Test
Accuracy (%)	89.2	85.9	85.5

Table 3. Accuracy of model at epoch 50.

CNN=CustomResnet, and dim_feedforward=2048.

Out of the three experiments, the last case (Figure 7) had the lowest validation loss. Therefore, further training was done. The training results are shown in Figure 8. The validation accuracy is 89.2% and the challenge accuracy is 85.9%. The best validation loss occurred at epoch 50, so the model's epoch is set to 50. At the last epoch, the validation loss is greater than the training loss, suggesting the possibility of overfitting.

In Table 3, the challenge validation and test accuracy are worse than the train validation accuracy. Therefore, instead of using the train validation set, the challenge validation set was used for validation from epoch 51 onwards. The training data remained unchanged. The training results are shown in Figures 9 and 10. Each training session was stopped due to overfitting. The model saved in Figure 9 corresponds to epoch 54. Thus, in Figure 10, training starts from epoch 55. In Figure 10, the model's epoch is 62, which is the final model.

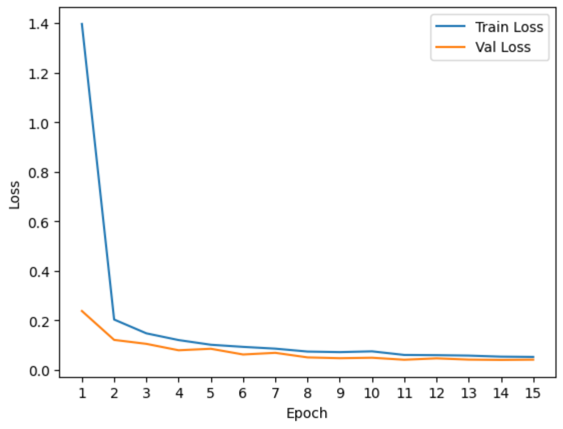

Figure 7. Training results for hyperparameters with rnn_dropout=0.2 and nhead=4. The learning rate scheduler is set with a decay of 15.

	Train Validation	Challenge Validation	Kaggle Test
Accuracy (%)	90.745	87.537	87.171

Table 4. Accuracy of the final model. Epoch is 62.

	Train Validation	Challenge Validation	Kaggle Test
Accuracy (%)	90.941	87.755	87.458

Table 5. Accuracy after beam search.

Training was stopped due to overfitting. The final accuracy results are shown in Table 5.

Applying beam search when generating tokens, accuracy increased slightly.

4. Discussion

The results of the experiments demonstrate the effectiveness of the Seq2Seq models with both LSTM and Transformer architectures for the task of typo correction in RGB character image sequences. The normal task results validate the capability of the LSTM-based model to handle sequences of varying lengths effectively, leveraging the combined strengths of CNNs for feature extraction and LSTMs for sequence processing. The integration of a custom ResNet into the encoder significantly improved feature extraction, enabling better performance in sequence predic-

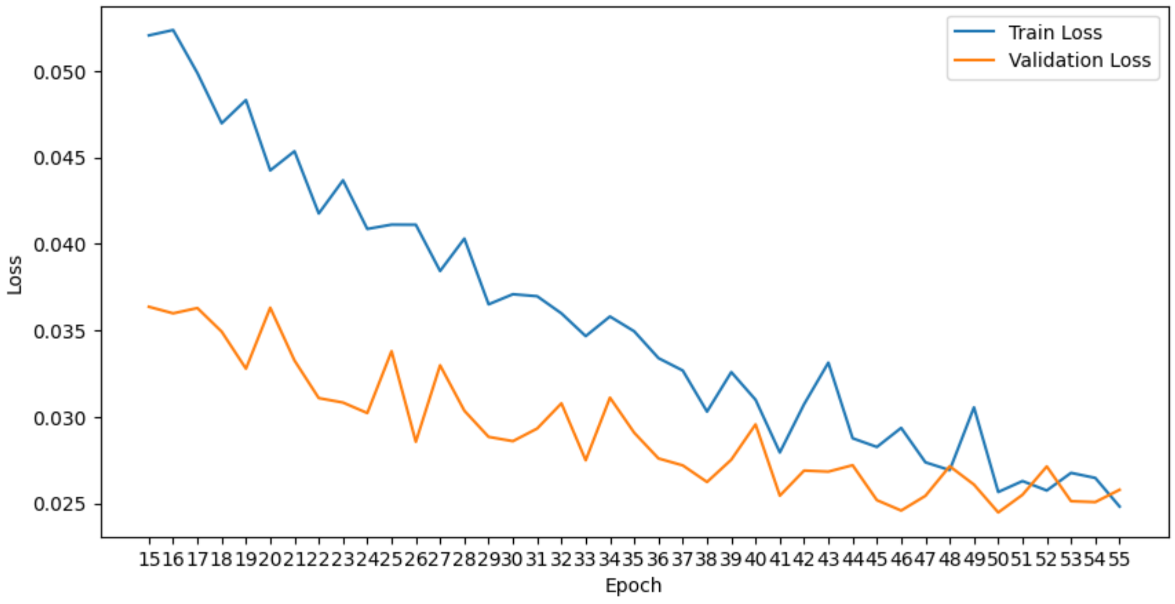

Figure 8. Training results for the model from Figure 7 (nhead=4, dropout=0.1). Epochs range from 15 to 55. The best validation was at epoch 50, so the saved model epoch is 50.

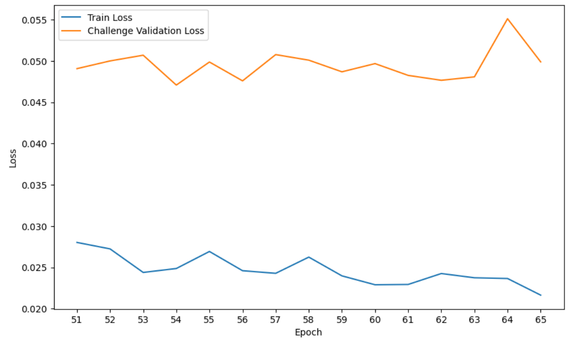

Figure 9. Training result using challenge validation data from epoch 51. The best validation loss occurred at epoch 54, so the saved model's epoch is 54.

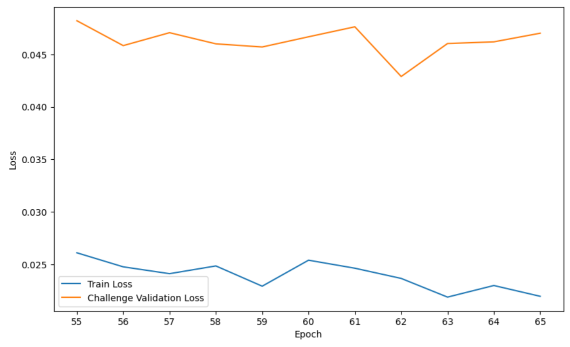

Figure 10. Training result using challenge validation data from epoch 55. The best validation loss occurred at epoch 62, so the saved model's epoch is 62.

tion.

The challenge task highlighted the advantages of the Transformer-based model, particularly in handling longer and more complex sequences. The implementation of multi-head self-attention mechanisms and feed-forward layers allowed the model to capture long-range dependencies more efficiently than traditional RNN-based models. The use of various masks ensured accurate sequence prediction without future information leakage, demonstrating the model's robustness in generating correct sequences.

Hyperparameter tuning played a critical role in enhancing model performance. The experiments revealed that a hidden dimension of 128 provided an optimal balance between model complexity and generalization capability. Adjusting the number of heads in the Transformer and fine-tuning dropout rates further contributed to reducing overfitting, as evidenced by the lower validation loss compared to training loss in the final model configuration.

Training the model required significant time, making it challenging to find the optimal hyperparameters. The hyperparameters of the final model were chosen based on the first 10-20 epochs. However, the optimal settings during the initial training epochs are not always the global optimum, so there may be other hyperparameters that outperform this final model.

Data augmentation techniques proved to be beneficial in increasing the robustness of the models. Random transformations of images and reordering of padding sequences introduced variability into the training data, preventing overfitting and enhancing the models' ability to generalize to unseen data.

5. Conclusion

This project successfully developed and evaluated Seq2Seq models for typo correction in RGB character image sequences, showcasing the effectiveness of both LSTM and Transformer architectures. The normal task demonstrated the capability of LSTM-based models with custom ResNet encoders to handle varying sequence lengths. The challenge task highlighted the superiority of Transformer-based models in managing longer and more complex sequences, thanks to their efficient handling of long-range dependencies.

The critical role of hyperparameter tuning and data augmentation in enhancing model performance was evident. The final Transformer-based model achieved a validation accuracy of 90.941%, a challenge validation accuracy of 87.755% and challenge test accuracy of 87.458%, underscoring its potential for practical applications in typo correction and similar sequence prediction tasks.

Future work could explore further enhancements such as incorporating attention mechanisms in the LSTM-based models, experimenting with different types of data augmentation, and applying these models to other sequence prediction tasks. Additionally, exploring more advanced architectures and techniques could further improve accuracy and generalization capabilities, making these models even more robust and versatile.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[2] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint*, 2017. arXiv:1712.04621.

[3] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(60):1–48, 2019.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 27:3104–3112, 2014.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 5998–6008, 2017.