2024-1 Machine Learning Final Project Report

Jinwoo Hwang
Seoul National University, ECE
Seoul, Gwanak-gu, Gwanak-ro 1
luorix@snu.ac.kr

Abstract

This project aims to develop a robust typo correction system for word sequences presented as RGB character image sequences. Utilizing a Seq2Seq model architecture, combining Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), the system processes input sequences of varying lengths (4 to 10 characters) and outputs the corrected word sequences as integer sequences corresponding to the alphabet. The training dataset comprises 67,890 sequences, with validation and challenge sets designed to evaluate model performance on unseen and longer sequences, respectively. The project includes detailed preprocessing, data augmentation, and hyper-parameter tuning strategies. The final evaluation is based on accuracy, assessed through a Kaggle competition framework. This project contributes to advancing OCR and typo correction techniques in AI, with potential applications in document processing and automated text correction systems.

1. Introduction

In the realm of natural language processing (NLP) and optical character recognition (OCR), the accurate recognition and correction of typographical errors in text images remain a significant challenge. These errors can stem from various sources, including manual data entry mistakes, noisy environments during data capture, and limitations in OCR technology. Addressing this challenge is crucial for numerous applications, such as automated document processing, digital archiving, and real-time text correction systems.

This paper presents an approach to typo correction which leverages deep learning techniques, specifically a sequence-to-sequence (Seq2Seq) model architecture. The model integrates Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to process and correct sequences of characters presented as RGB images. For the challenge task, we also incorporate transformers into our architecture. This integration enables the model to effectively capture spatial features from the character images and temporal dependencies within the sequences.

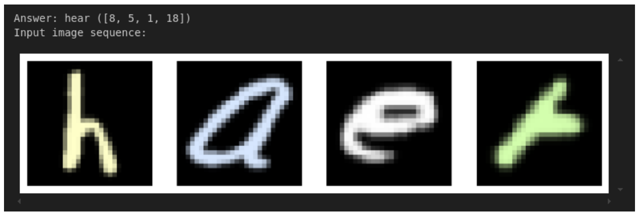
The motivation behind this work is to enhance the accuracy and efficiency of typo correction systems by developing a robust model that can generalize well across different types and lengths of sequences. The proposed model is evaluated on a comprehensive dataset consisting of training, validation, and challenge sets, each designed to test the model's performance under varying conditions. The training set includes a large number of sequences with lengths ranging from 4 to 8 characters, providing a rich basis for learning. The validation set comprises sequences of similar lengths but consists of words not present in the training set, ensuring a fair assessment of the model's generalization capabilities. The challenge set, on the other hand, includes longer sequences (9 to 10 characters) that push the model to handle more complex corrections.

Our approach is built upon the Seq2Seq framework, which has proven effective in various NLP tasks, including machine translation and text generation. By adapting this framework to the task of typo correction in character image sequences, we aim to demonstrate its versatility and effectiveness in handling visual-textual data. The model's architecture, training procedures, and evaluation methods are meticulously designed to ensure robustness and scalability.

Through this research, we contribute to the advancement of typo correction technologies, offering a solution that combines the strengths of CNNs, RNNs, and transformers in a unified framework. Our findings underscore the potential of deep learning models to significantly improve the accuracy and reliability of text recognition systems, paving the way for more intelligent and automated text processing applications.

2. Method

All text must be in a two-column format. The total allowable width of the text area is $6\frac{7}{8}$ inches (17.5 cm) wide by $8\frac{7}{8}$ inches (22.54 cm) high. Columns are to be $3\frac{1}{4}$ inches



```
Answer: hear ([8, 5, 1, 18])
Input image sequence:
```

Figure 1. Sample visualization of dataset (images, labels)

(8.25 cm) wide, with a $\frac{5}{16}$ inch (0.8 cm) space between them. The main title (on the first page) should begin 1.0 inch (2.54 cm) from the top edge of the page. The second and following pages should begin 1.0 inch (2.54 cm) from the top edge. On all pages, the bottom margin should be 1-1/8 inches (2.86 cm) from the bottom edge of the page for 8.5×11-inch paper; for A4 paper, approximately 1-5/8 inches (4.13 cm) from the bottom edge of the page.

2.1. Dataset

The dataset for this project is meticulously structured to support the training, validation, and evaluation of the typo correction model. It includes image sequences of characters along with their corresponding labels, designed to test the model's ability to correct typographical errors in sequences of varying lengths.

2.1.1 Training Set

The training dataset is composed of 67,890 image sequences, each with a length ranging from 4 to 8 characters. These sequences are presented as RGB images with a shape of $(T, 28, 28, 3)$, where T denotes the sequence length. The labels for these sequences are provided as 1-dimensional integer sequences corresponding to the alphabet, where each integer represents a character without distinguishing between uppercase and lowercase. This extensive dataset allows the model to learn a wide variety of typographical patterns and errors, thereby enhancing its generalization capabilities.

2.1.2 Validation Set

The validation set includes 9,670 image sequences, also with lengths ranging from 4 to 8 characters. However, the sequences in this set are composed of words that do not appear in the training dataset. This separation ensures that the model's performance is evaluated on previously unseen data, providing a more accurate measure of its generalization ability. The validation set is crucial for tuning hyperparameters and preventing overfitting during the training process. The validation set also acts as our test set for the base task.

2.1.3 Test (Challenge) Set

The challenge set is designed to push the limits of the model by including 8,714 image sequences with lengths ranging from 9 to 10 characters. Similar to the validation set, these sequences consist of words and lengths not present in the training dataset. The challenge set is used for a relative performance evaluation, comparing the model's accuracy on more complex and longer sequences.

2.2. Model Architecture

The model architecture for the typo correction system employs a sequence-to-sequence (Seq2Seq) framework that integrates Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). This section provides a detailed description of the CNN and RNN components, the encoder-decoder structure, and the implementation specifics.

Our model can be categorized into two base models, the encoder and the decoder. The encoder leverages a CNN feature extractor and subsequent RNN while the decoder mainly focuses on an RNN for generating a new sequence.

The feature extraction from the input RGB character image sequences is handled by a custom CNN, which can be configured as either a VGG-style or a ResNet-style network:

1. VGG block: Each VGG block consists of two convolutional layers with ReLU activation functions followed by a max-pooling layer. The weights are initialized using the Kaiming normal initialization method. 2. ResNet block: Each ResNet block includes two convolutional layers with batch normalization and ReLU activations. A residual connection is added to facilitate gradient flow. If the input and output dimensions differ, a shortcut connection with a convolutional layer adjusts the dimensions accordingly.

The CNN consolidates these blocks into a three-stage CNN, ending with a fully connected layer to transform the extracted features into a suitable format for the RNN encoder.

The encoder employs the custom CNN for initial feature extraction followed by an RNN to capture temporal dependencies in the sequences. The specific components are:

1. CNN feature extractor: Extracts features from the input sequences, outputting a tensor of shape (B, T, H). 2. RNN Layer: Configurable as either LSTM or GRU, it processes the sequential data from the CNN output. The RNN can be unidirectional or bidirectional, with the latter option concatenating the hidden states from both directions. 3. Fully Connected Layer: Maps the RNN output to the desired hidden dimension size.

Here, the batch size is denoted as B, sequence length is denoted as T, and hidden dimension of the RNN is denoted as H.

The encoder's forward pass handles variable-length input sequences using the **pack_padded_sequence** and

pad_packed_sequence functions to ensure efficient computation and handling of padding.

The decoder generates the corrected sequence from the encoder's hidden state:

1. Embedding Layer: Converts input token indices to dense vectors. 2. RNN Layer: Similar to the encoder, this layer can be configured as LSTM or GRU and is responsible for generating the corrected output. Here, we use a unidirectional RNN due to the sequential nature of generation. 3. Linear Layer (Language Model Head): Maps the RNN outputs to the vocabulary size, producing logits for each token in the sequence.

The decoder's forward pass takes the embedded input sequence and the hidden state from the encoder, generating the predicted sequence one token at a time.

This architecture effectively captures both spatial features from the character images and temporal dependencies within the sequences, providing a robust framework for typo correction in RGB character image sequences.

For our additional model which is evaluated on the test set, we apply more advanced attention mechanisms

2.3. Model Training

The training process for the typo correction model involves a comprehensive approach, including the configuration of hyperparameters, selection of an optimizer, definition of the loss function, and implementation of an efficient training loop. This section provides a detailed description of these components and the overall training methodology.

2.3.1 Teacher Forcing

2.3.2 Hyperparamters

In this study, several key hyperparameters were meticulously selected and tuned to optimize the model's performance. The batch size was set to 128, ensuring an effective balance between memory usage and training stability. The hidden dimension for both the CNN and RNN layers was configured to 128, providing sufficient capacity for capturing intricate patterns in the data. The model employed three RNN layers, enhancing its ability to learn from sequential data. A dropout rate of 0.3 was applied within the RNN layers to mitigate overfitting by randomly deactivating a fraction of neurons during training. The GRU (Gated Recurrent Unit) was chosen for the RNN type, known for its efficiency in handling sequential data with fewer parameters compared to LSTM. Additionally, the encoder RNN was set to be bidirectional, enabling the model to consider dependencies from both past and future contexts.

2.3.3 Dataset

The training and validation datasets were loaded using custom data loaders designed to handle variable-length sequences and perform necessary preprocessing. The training dataset comprised images and corresponding labels stored in JSON files, loaded using the MLDataset class. Similarly, the validation dataset was structured to evaluate the model's performance on unseen data. The data loaders were configured with a batch size of 128 and employed a custom **collate_fn** function to manage batching and padding of sequences, ensuring efficient data handling.

2.3.4 Model Initialization (Base Task)

The Seq2Seq model was instantiated with specific hyperparameters tailored for this task. The CNN feature extractor was configured with three blocks of VGG-style convolutional layers, having dimensions 32, 64, and 128 respectively, and a fully connected layer of dimension 128. The RNN encoder was set to be bidirectional with a hidden dimension of 128 and three layers. The decoder was correspondingly configured to align with the encoder's output dimensions.

2.3.5 Model Initialization (Challenge Task)

The Seq2Seq model in this case included the same CNN feature extractor and bidirectional GRU as the encoder architecture. The CNN remained the same with three blocks of VGG-style convolutional layers with 32, 64, and 128 channels, respectively. The fully connected layer had dimension 256 and the bidirectional GRU had 3 layers with hidden dimension 256.

The main difference lies in the decoder architecture where we employed transformers using positional encoding and attention to improve performance compared to the simple bidirectionaL GRU. Our decoder had 3 layers with feed-forward dimension of 1024 and had 4 attention heads.

2.3.6 Optimizer and Loss Function

The Adam optimizer was selected for training due to its adaptive learning rate capabilities, which efficiently handle sparse gradients and improve convergence rates. The learning rate was initially set to 0.001. Cross-Entropy Loss was used as the loss function, suitable for multi-class classification tasks by comparing the predicted class probabilities with the true labels. Additionally, a learning rate scheduler was implemented to reduce the learning rate by a factor of 0.5 every ten epochs, promoting finer adjustments as training progressed.

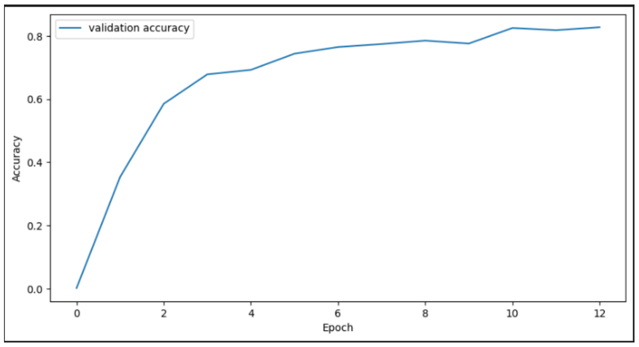


Figure 2. Validation set accuracy by epoch

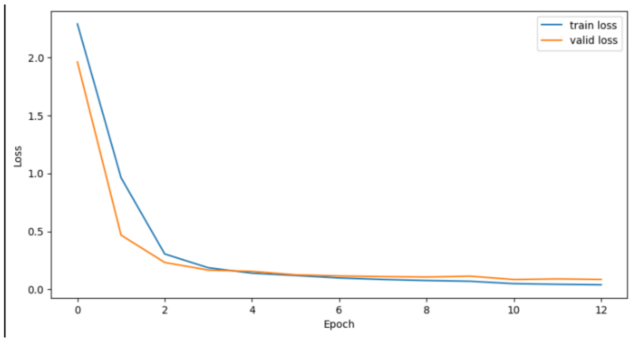


Figure 3. Training and validation loss by epoch

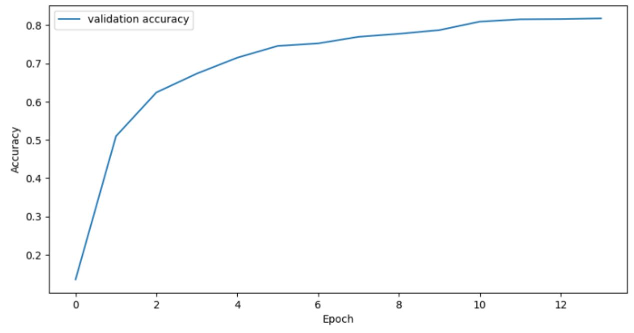


Figure 4. Validation set accuracy by epoch

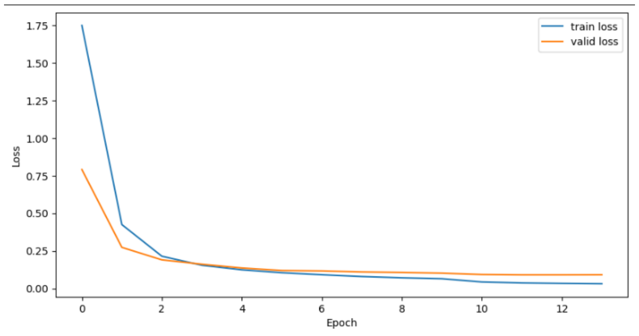


Figure 5. Training and validation loss by epoch

3. Experiments

3.1. Base Task

3.1.1 Base Experiment

In this experiment, we trained the base model for 30 epochs and evaluated on the validation set. The base model used a bidirectional GRU in the encoder and a unidirectional GRU in the decoder. Feature extraction was performed with 3 VGG blocks stacked with a fully-connected layer at the end to generate a 1D vector of features.

In 3, we plot the training loss and validation loss by epoch. Our objective is to minimize the validation loss so we employed early stopping with tolerance 2 to ensure our model does not overfit. We note that early stopping occured around the 12th epoch.

Inf 2, we plot the performance of our model on the validation set by epoch. We notice that the performance consistently increases and ends with 82.5%.

3.1.2 Ablations

For ablations, we implemented variations of our base model and tested the performance on the validation set.

First, we tried a ResNet-based CNN feature extractor instead of a VGG-based one. ResNets are traditionally thought to be advantageous due to their skip connections acting as a ”super-highway” for gradients during backpropagation.

However, we observed that the ResNet feature extractor had no clear benefits compared to the VGG feature extractor. The model showed performance of 81.6% which is actually slightly lower than our base model. Plots outlining the model training process are given in 4 and 5.

Second, we replaced our GRUs with LSTMs which are generally known to better capture longer-range dependencies (hence, the name Long Short Term Memory). We obtained 60.1% accuracy for one of our runs.

Overall, the LSTM variant did not perform well and was much more difficult to train than the GRU case. We hypothesize that LSTMs can be more sensitive to weight initialization and various hyperparameters.

Finally, we replaced our bidirectional GRU in the encoder with a unidirectional GRU. Our design choice to use a bidirectional GRU was based on the intuition that the model should be able to learn from relationships not only with preceding tokens, but subsequent tokens as well. Therefore, we predicted that removing the bidirectional GRU and using a unidirectional GRU would cause a drop in performance.

Our results showed that the unidirectional case is actually not that different from the bidirectional case. The model took 16 epochs to converge and had performance of

Model	Components	Number of Parameters
Base Model	VGG + GRU	1,860,156
Ablation 1	ResNet + GRU	1,872,060
Ablation 2	VGG + LSTM	2,321,980
Unidirectional GRU	-	1,052,732

Table 1. Number of parameters for different model configurations

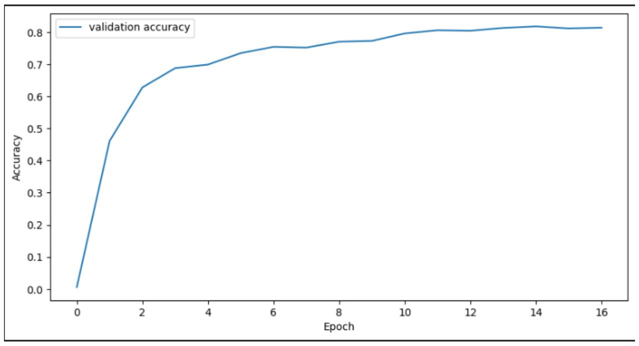


Figure 6. Validation set accuracy by epoch

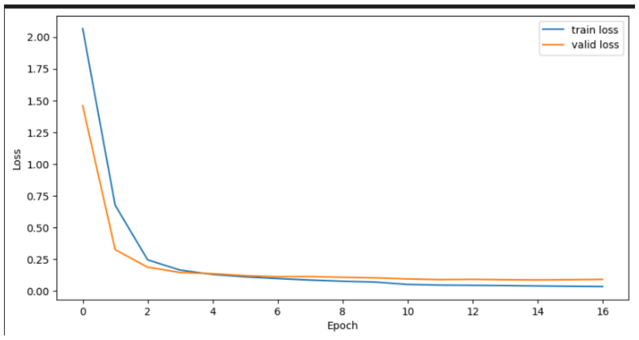
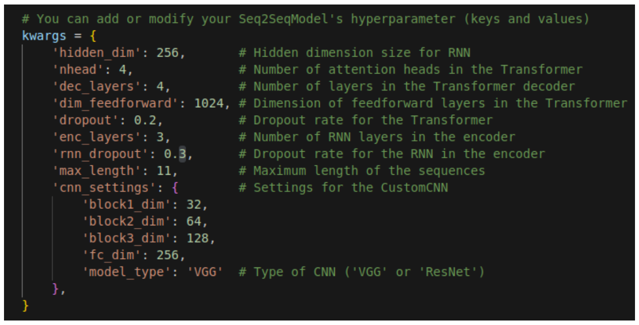


Figure 7. Training and validation loss by epoch

81.8% on the validation set. As such, we see that although using a bidirectional RNN has some advantages, it does not directly translate to a large increase in performance. Plots outlining the model training process are given in 6 and 7.

Note that for the decoder, we do not use a bidirectional RNN due to the sequential nature of the generation process.

In 1, we compare the number of parameters for our base model and each of our ablations. We can see that the ResNet variant is slightly larger than our base model and that using LSTM causes a significant increase in model parameters (24.8% increase). Finally, replacing our bidirectional RNNs with unidirectional ones caused a significant decrease in model parameters (43.4% decrease) with slightly lower performance.



```
# You can add or modify your Seq2SeqModel's hyperparameter (keys and values)
kwargs = {
    'hidden_dim': 256,       # Hidden dimension size for RNN
    'nhead': 4,              # Number of attention heads in the Transformer
    'dec_layers': 4,         # Number of layers in the Transformer decoder
    'dim_feedforward': 1024, # Dimension of feedforward layers in the Transformer
    'dropout': 0.2,          # Dropout rate for the Transformer
    'enc_layers': 3,         # Number of RNN layers in the encoder
    'rnn_dropout': 0.3,      # Dropout rate for the RNN in the encoder
    'max_length': 11,        # Maximum length of the sequences
    'cnn_settings': {        # Settings for the CustomCNN
        'block1_dim': 32,
        'block2_dim': 64,
        'block3_dim': 128,
        'fc_dim': 256,
        'model_type': 'VGG'  # Type of CNN ('VGG' or 'ResNet')
    },
}
```

Figure 8. Validation set accuracy by epoch

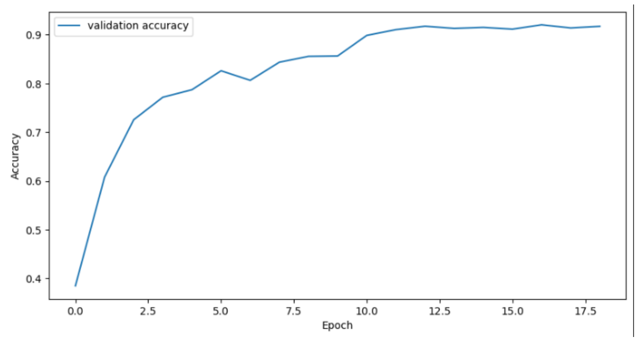


Figure 9. Validation set accuracy by epoch

3.2. Challenge Task

3.2.1 Base Experiment

For the challenge task, we trained the model for 20 epochs and plotted the loss and accuracy over all epochs. Again, we trained with patience 2 so the model was subject to early stopping.

We can see that the model had peak performance on the validation set of 92.6%. On the unseen test set, the model had 83.9% accuracy.

The setup used is shown in 8.

3.2.2 Ablations

For ablations, we tested multiple variants of our model including number of encoder layers, number of decoder layers, number of attention heads, and dimensions of the output from the fully connected layer in the CNN feature extractor. We also experimented with dropout. Overall, our hyperpa-

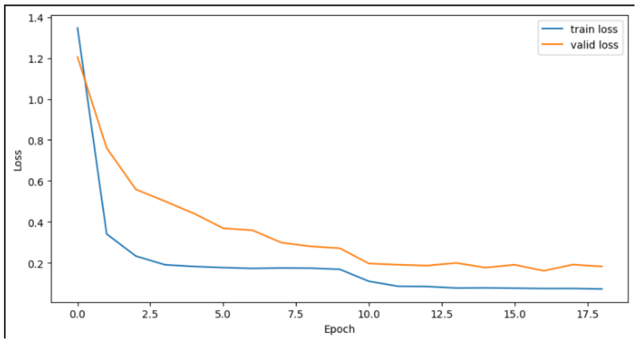


Figure 10. Training and validation loss by epoch

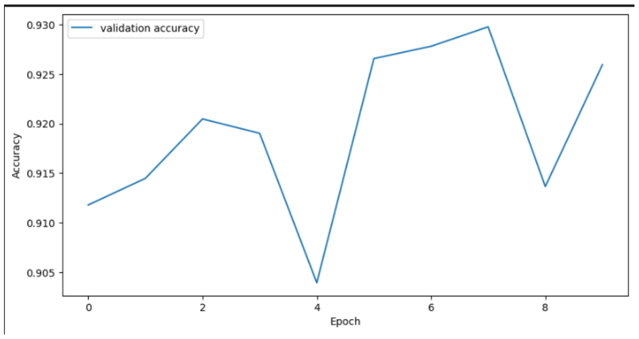


Figure 11. Validation set accuracy by epoch

rameter search spanned the following set of values but our base model consistently performed the best.

- **Encoder Layers**: 2, 3, 4
- **Decoder Layers**: 2, 3, 4
- **Attention Heads**: 4, 8
- **Dimension of Output from Fully Connected Layer**: 128, 256, 512
- **Dropout Rates**: 0.2, 0.3, 0.4

We also tested various augmentations and their effects on the final checkpoint performance. Data augmentation is employed to increase the diversity of the training data without collecting new data and can emulate training on a larger and more varied training set. Our dataset includes several augmentation techniques, which we analyze through ablation studies to understand their individual and combined effects on model performance.

- **Random Translation**: This technique applies small translations to the images, helping the model become invariant to slight positional shifts of the characters. The augmentation is controlled by `transforms.RandomAffine(degrees=0, translate=(0.1, 0.1))`.
- **Color Jitter**: This technique adjusts the brightness, contrast, saturation, and hue of the images. It aims to improve the model's robustness to variations in lighting conditions. This is implemented via `transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2)`.
- **Random Rotation**: By rotating the images slightly, this augmentation helps the model handle variations in character orientation. We use `transforms.RandomAffine(degrees=8)` for this purpose.
- **Gaussian Blur**: Applying Gaussian blur can help the model learn to recognize characters despite slight blurriness, which might occur in real-world scenarios. This is achieved through `transforms.GaussianBlur(kernel_size=(3, 3))`.
- **Random Swap**: This novel augmentation introduces randomness in the character sequence by swapping two characters with a small probability. It helps the model learn to correct sequences that have minor positional errors. This is controlled manually in the `__getitem__` method.

Color Jitter and Random Translation did not have much effect on the model as the final checkpoint showed 91.2% performance on the validation set and 77.8% on the test set which was lower than our base challenge model.

Gaussian Blur and Random Rotation also did not show much improvement as it showed 90.0% on the validation set and 75.8% on the test set.

Second, we attempted all augmentations including Random Swap, an ablation where any two images in the sequence are swapped so as to further misspell the word or possibly (unintentionally) fix the existing typo. We trained this for 20 epochs but it also experienced early stopping. The model showed 91.1% accuracy on the validation set and 77.1% accuracy on the test set.

Finally, we also attempted continual learning by loading our best performing checkpoint into our training loop and performing a few epochs of fine-tuning. However, even with learning rate lowered to 1e-4 instead of 1e-3 as in our experiments training from scratch, our model failed to improve.

As a final ablation, we tested our base model using a GRU for the decoder on the test set. We found that our best performing model showed 38.4% accuracy on the test set as opposed to the 82.5% on the validation set. We observe that the ability of the GRU to generalize to unseen lengths was lesser than that of the transformer.

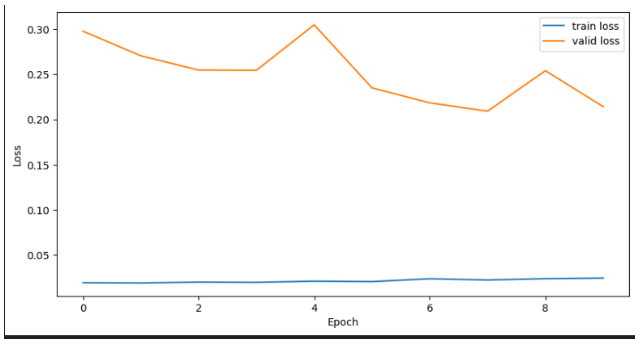


Figure 12. Training and validation loss by epoch

4. Conclusion

Overall, in this paper, we explored a sequence-to-sequence model leveraging CNNs, RNNs, and Transformers to perform correction of misspelled words from a series of 28*28 RGB images. We performed various ablations as to the optimal architecture choices for the feature extractor, encoder, and decoder, respectively and showed the effects of various augmentations on our model as well.

We noticed that the performance of the transformer during training is subject to higher variance than RNN-based models. This can be seen in 11 and 12. This higher variance can be attributed to the transformer's ability to capture long-range dependencies and complex relationships within the data, which, while powerful, can also make the model more sensitive to initialization and training conditions. Transformers rely heavily on self-attention mechanisms, which can lead to fluctuations in learning dynamics, especially in the early stages of training.

Moreover, we observed that the transformer model successfully generalizes to sequences of unseen lengths, while the GRU-based model failed to do so. This is because transformers use positional encodings to manage sequence length variations, allowing them to handle input sequences of varying lengths more effectively. In contrast, GRU and other RNN-based models, which process sequences sequentially, often struggle to generalize beyond the specific sequence lengths seen during training due to their inherent limitations in capturing long-range dependencies.

Several avenues for future research can be explored to further enhance the performance and applicability of our spelling correction model:

- Hybrid Models: Combining the strengths of transformers and RNNs or CNNs to create hybrid models could leverage the advantages of both architectures, potentially improving robustness and performance.

- Advanced Data Augmentation Techniques: Investigating more sophisticated data augmentation methods, such as generative adversarial networks (GANs) for generating synthetic training data, could help in further improving model generalization.

- Transfer Learning: Applying transfer learning techniques by pre-training the model on large, diverse datasets before fine-tuning on specific spelling correction tasks may yield better performance and faster convergence.

- Real-Time Applications: Developing real-time spelling correction systems that can operate efficiently on mobile devices or edge computing environments would be a valuable extension of this research.

- Multilingual Support: Extending the model to support multiple languages and incorporating language-specific spelling rules and nuances could broaden the model's applicability and usefulness.

By pursuing these directions, future work can build upon our findings to develop even more robust, efficient, and versatile models for spelling correction and related tasks.

References

[1] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 1171–1179, 2015.

[2] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems*, volume 29, pages 4601–4609, 2016.

[3] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, volume 27, pages 3104–3112, 2014.

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

[6] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.