Molecular Translation: An Image Captioning Application in Organic Chemistry, CS7643

Boyuan Liu, Daniel Bears, Elbek Kamoliddinov, and Alexandre Torres
dbears3@gatech.edu
Georgia Institute of Technology

Abstract

The ability to quickly sort through documentation for information related to an organic compound is invaluable to researchers. Today, much of this documentation task is done manually due to image corruption and requires a lot of work to convert these images into a reliable machine-readable format. This project explores the implementation of an Image Captioning model using a CNN + LSTM encoder-decoder architecture with attention. To accomplish this, we experimented with several pre-trained CNN and vision transformer models for the encoder and leveraged an LSTM for the decoder with attention.

1. Introduction/Background/motivation

In the field of chemistry, there are estimated to be around 20 million organic compounds. Researchers often find themselves digging through piles of documentation searching for information related to an organic compound. This is often not an easy task because there are decades of research archived and not every compound in a chemical reaction figure is labeled. Existing tools can produce up to 90% accuracy but only under optimal conditions. In many cases, historical sources often have some level of image corruption, which reduces performance to near zero and when that happens, a large amount of manual work is required to reliably convert scanned chemical structure images into a machine-readable format [1].
Our goal for this project is to build an image captioning model that can process batches of organic images and accurately return their International Chemical Identifier (InChI) text representation.

1.1 Dataset

We used a bms-molecular-translation dataset provided by Kaggle [1]. This dataset contains 4 million images of organic chemical compounds with their corresponding International Chemical Identifier (InChI) text string. The total size of this dataset is 8.87GB. Figure 1 contains an example of a training image from the organic compound dataset, while Figure 2 contains an example of the InChI text representation for training labels and the InChI that we want to predict.

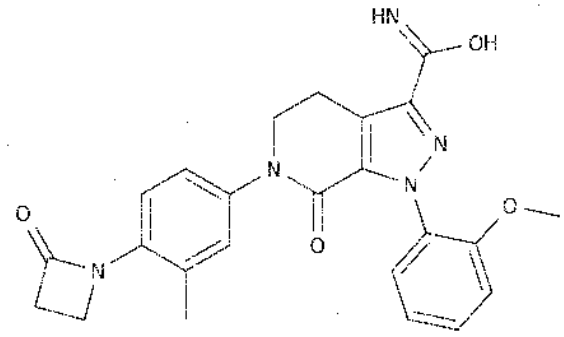

Figure 1 -Training image example from the organic compound dataset.

InChI=1S/C13H20OS/c1-9(2)8-15-13-6-5-10(
3)7-12(13)11(4)14/h5-7,9,11,14H,8H2,1-4H3

Figure 2 -Training label example from the organic compound dataset.

2. Approach

Molecular translation is an image captioning task at its core. Image captioning is an important task as it allows us to automate the task of generating text

representations for any image. Attention based encoding decoding image captioning algorithms introduce an attention mechanism in the decoding stage that combines the words generated in the previous moment with the local visual information of the image, making it possible to dynamically focus on different regions of the image as each word is generated by the language module [4]. We leveraged this through an encoder-decoder architecture with a CNN encoder + Long Short Term Memory (LSTM) decoder with attention.

2.1 CNN Encoder

We created an encoder block that can take in any CNN or vision transformer models. The encoder takes organic compound images as inputs and performs feature extraction to generate embedded feature vectors that will be passed as input to the decoder. We experimented with several pre-trained models including EfficientNet, MobileNet, ResNet50, Vision Transformer, and Transformer in Transformer and eventually chose EfficientNet because of its overall performance and efficiency. EfficientNet is a state-of-the-art computer vision architecture. It uses a compound scaling technique to scale up from a baseline model B0 to a complex B7 model [2]. For the experiments, we compared the results of the pre-trained models above, but focused on leveraging the EfficientNet model as our pre-trained model.

2.2 Decoder - LSTM with Attention

For the decoder, we leveraged a two-layer LSTM with attention. The feature vectors are fed in as input to the decoder and it generates a predicted text sequence. We chose this architecture to take advantage of the LSTM ability to deal with the vanishing gradient problem as the sequence length increases. The attention mechanism allows the decoder to attend to all the hidden states fed in from the encoder allowing it to focus multiple parts of the input to better understand the context. This ultimately leads to better generated predicted text sequences.

3. Experimental Setup and Results

3.1 Loss Function And Scoring Metric

The loss function applied was Cross-Entropy loss between the generated sequence and ground truth sequence. We chose to use Adam as our optimizer for both the encoder and decoder. To determine the success of our model, we used Levenshtein distance to measure the difference between the predicted text sequence and the ground truth text sequence [7]. The Levenshtein distance between two sequences is based on the minimum number of single-character edits in the form of insertions, deletions, or substitutions needed to convert one sequence to another [6]. Figure 3 below shows how this metric is computed.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise,} \end{cases}$$

Figure 3 - Levenshtein distance

Whenever we refer to score in our experiments and results, we are referring to the Levenshtein distance. The goal of our model is to minimize the Levenshtein distance between the prediction and true label. However, since the Levenshtein distance equation is not differentiable, in order to perform backpropagation we needed to use cross-entropy loss as the loss function.

3.2 Pre-Trained CNN Encoder Model Comparisons

We conducted several experiments on the encoder using different pre-trained state-of-the-art models including EfficientNet, MobileNet, ResNet50, Vision Transformer, and Transformer in Transformer. We trained these models using our encoder-decoder setup with the same set of parameters. The results are compared in Figure 4.

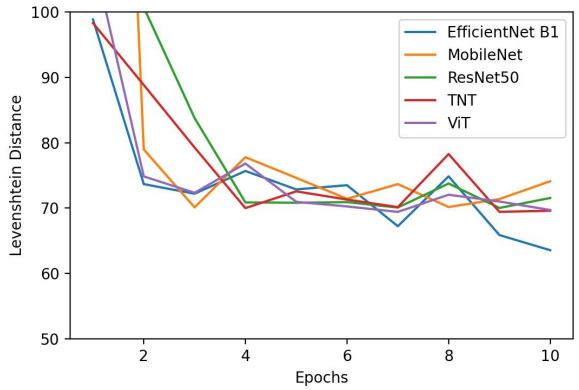

Figure 4 - Levenshtein distance of different vision models

EfficientNet B1 outperforms the other models by an obvious margin in terms of both Levenshtein distance score and cross entropy loss. We then experimented further with the family of EfficientNet models and obtained the results shown in Figure 5.

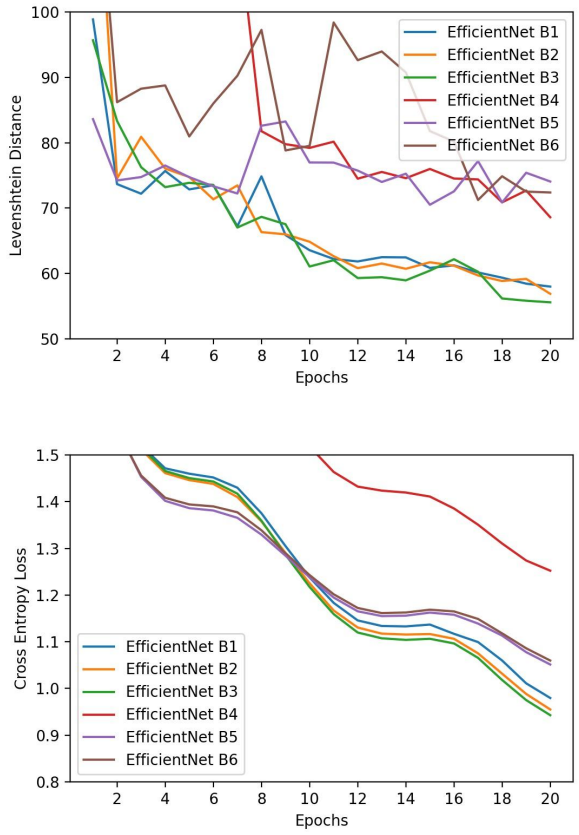

Figure 5 - Levenshtein distance and cross entropy loss of EfficientNet Models.

EfficientNet B2 and B3 performed better than the other EfficientNet models in Levenshtein distance score and cross entropy loss. We decided to choose EfficientNet B2 as our final encoder model after balancing training time and performance. Figure 6 shows the learning and Loss curve for the 50k sample size model. We can see that the model is still improving with more epochs. Also training and validation curves are very tight indicating it is possible that we may benefit with larger capacity models. We couldn't pursue further experiments on larger models with more epochs due to hardware limitations, see 4.1 for more details.

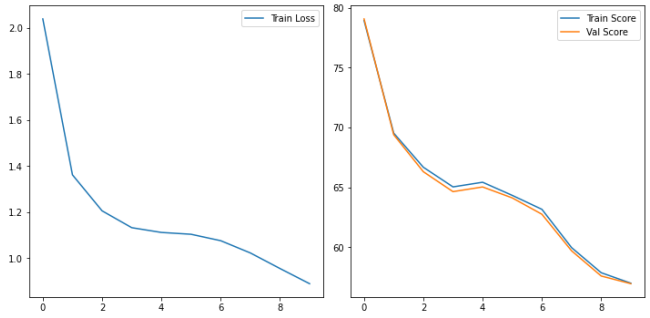

Figure 6 - Learning and Loss curve.

3.3 Hyper-Parameter Tuning

We conducted several experiments to determine the best hyper-parameters for our model and training dataset. We varied the pre-trained models, encoder size, encoder learning rate, decoder learning rate, number of epochs, training sample size, and dropout. For the sake of brevity, we only highlight the interesting results in Table 1 and Table 2 below.

Set	Pre-trained model	encoder LR	decoder LR	Epochs	Sample Size	dropout
1	b2	1E-04	4E-04	4	10000	0.5
2	b2	2E-04	8E-04	4	10000	0.5
3	b2	2E-03	8E-03	4	10000	0.5
4	b2	1E-04	4E-04	10	100000	0.5
5	b3	1E-04	4E-04	2	100000	0.2
6	b3	1E-04	4E-04	10	100000	0.2
7	b4	1E-04	4E-04	10	100000	0.2

Table 1 - Summary of hyper-parameter tuning.

Set	Loss	Score
1	1.336	69.88
2	1.3631	76.65
3	1.3328	69.126
4	0.5122	36.0635
5	0.8255	52.6313
6	0.3989	36.1672
7	0.685	50.2357

Table 2 - Loss and Levenshtein distance corresponding to each set of hyper-parameters in Table 1.

After determining the best pre-trained model to use for our encoder, we focused on hyperparameter tuning. We experimented with several hyperparameters using a subset of the larger dataset with 10k samples to establish a trend before training with a larger subset. The hyperparameters tuned were encoder size, encoder learning rate, decoder size, decoder learning rate, batch size, dropout, and number of epochs. Doubling the learning rate for the encoder and decoder from 1E-04 and 2E-04 to 2e-04 and 8E-04, resulted in overfitting, so we reduced the rate and increased the epochs to improve training results. Additionally, increasing the dropout from 0.2 to 0.5 resulted in better performance due to the regularization effect of dropout. From our experimentation, we obtained the best performance with the pre-trained EfficientNet B2 model, an encoder size of 1048, an encoder learning rate of 1E-04, a decoder learning rate of 4E-04, batch size of 32, a sample size of 100k, and ten epochs. This corresponds to set 4 in Table 2 above. Figure 7 below contains the loss curve and validation score for this model.

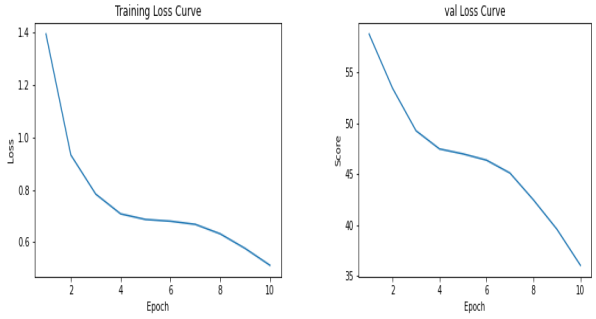

Figure 7 - Learning and Loss curve for best model

We found that dataset size and number of epochs had more impact on the initial training result. Baseline models started to overfit around the 7th epoch. Thus, we trained the model with each one of these configurations using a dataset containing 100k randomly selected records. We also trained multiple models on different dataset sizes to see how the dataset impacts training time as well as model performance. Results are shown in Table 3.

Dataset size	Score	loss	training time of each batch
10000	63.375	1.0131	541
50000	52.99	0.81	2013
100000	36.06	0.51	3962

Table 3 - Impact of dataset size to model performance and training time.

Training time grew near linearly with the dataset size.

3.4 Results

From our experiments, we drew the following conclusions that we felt should be highlighted.

- EfficientNet B4 has an equivalent performance but is more time-consuming (e.g., six and a half hours for EfficientNet B2 versus ten and a half hours for EfficientNet B4 , given the same hyperparameters).
- The dropout had a low to zero effect in the smaller dataset. This is expected, as the regularization would have a more significant impact in bigger datasets.
- Bigger learning rates fail to improve both the encoder and decoder results and result in overfitting.

With our best model, we achieved a loss of 0.5122 and a score of 36.0635. The score value would classify the model in the 683rd position in Kaggle's Bristol-Myers Squibb – Molecular Translation competition. The competition awarded the first 100 places, all with a score below 2.0. Despite the competition

classification, we succeeded in building an effective and improvable model. Given the constraints highlighted in section 4.1, we had satisfactory results and touch on some potential future improvements in section 5.

4. Experience

4.1 Challenges

One of the main challenges we faced was the amount of time it took to train the model due to the size of the dataset, 4 million images and labels. To overcome this, we made the decision to use a random subset of the dataset for training. This allowed us to reduce the training time significantly from many hours to less than an hour depending on the pre-trained model used for the CNN encoder. This reduction in training time reduced the overhead in terms of time of hyper-parameter tuning and ultimately let us explore the hyperparameter space more.

Another challenge we faced, related to the first challenge, had to do with memory limitations on the single GPU Google collab notebook we used for training. We hit the memory limit on the notebook for most of the larger models and datasets. This forced us to limit our selection of larger baseline models, which affected both the encoder and decoder due to the reduction in available training samples and model complexity.

5. Future Improvement

There are several aspects of the model that we might be able to improve with future experiments. Our model currently uses cross entropy loss as the loss function. As shown Figure 5, the cross entropy losses continuously decrease with training. The Levenshtein distances show a declining trend but with obvious fluctuation. This shows that a lower cross entropy loss does not always lead to a lower Levenshtein distance, which means that there is some level of disconnection between cross entropy loss and Levenshtein distance. We plan to experiment with more metrics in the future and try to find metrics that can more closely represent Levenshtein distance.

Another area for future improvement relates to availability of hardware resources. With access to more GPU time or more GPUs for training and memory, we could train larger models on the full dataset of 4 million samples. From the results that we saw in our experimentation, we expect that access to more training samples would improve the loss and performance of the model. This can be seen in the continual decrease of the loss curve on our plotted results. Additionally, the Kaggle competition that posed this challenge had a leaderboard where the high performing models trained on the full 4 million samples.

6. Work Division

Summary of contributions is provided in Table 4 in the appendix.

References

[1] Bristol-Myers Squibb – Molecular Translation, https://www.kaggle.com/c/bms-molecular-translation/overview

[2] MingXing Tan, Quoc V Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2019.

[3] Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1808.07233*, 2018.

[4] Yanchi Li, Automated Recognition of Chemical Molecule Images Based on an Improved TNT Model, 2022.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, “Attention is All you Need”, *Advances in Neural Information Processing Systems 30*, 2017.

[6] Levenshtein, Vladimir I, "Binary codes capable of correcting deletions, insertions, and reversals", *Soviet Physics Doklady*, 1966.

[7] Haldar, Rishin and Mukhopadhyay, Debajyoti. (2011). Levenshtein Distance Technique in Dictionary Lookup Methods: And Improve. https://arxiv.org/abs/1101.1232d

APPENDIX

Github repo link for the notebook: https://github.com/bryanliu94/Molecular_Translation_with_Image_Captioning

Student	Contribution	Details
Daniel Bears	Hyperparameter Tuning, Model experiments and plot generation, Identifying best performing model, Report sections, testing with multiple models and dataset sizes.	See section 1, 1.1, 2, 2.2, 3.1, 3.3, 4.1, 5
Boyuan Liu	Construct the encoder-decoder pipeline, Experiment with different encoder models, hyperparameter Tuning	Put together the initial version of our working notebook which included the encoder/decoder blocks, training loop and helper functions. Section 2.1, 3.2, 5.
Elbek Kamoliddinov	Running tuned models and generating learning and loss curves. Running on different dataset sizes to measure dataset impact	Contributed to 3.3 and 3.4, analyzed learning curves for different dataset sizes.
Alexandre Torres	Hyperparameter tuning and model experiments. Report parts and contribution	Several model runnings, parts of sections 3.3 and 3.4

Table 4 - Summary of contributions.