Image Caption with GIT and modified mPLUG

Author - Futian Zhang
f6zhang@ucsd.edu

1 Introduction

Image captioning is the task of generating an accurate textual description for a given image, which requires both image analysis and language generation. Manually writing the captions for images is time-consuming, therefore, writing captions with a language model can automate this process. It involves natural language processing and image processing, two distinct domains of machine learning. Model-generated captions are commonly used in books and to assist visually impaired individuals. It can also help LLM to understand the images.

- Collected the COCO caption dataset: Done.
- Use CLIP to convert images into latent spaces: Done.
- Build the (LSTM Decoder) baseline model and examine its performance: Done.
- Build GIT and examine its performance: Done.
- Build mPLUG and examine its performance: Done.

2 Related work

Automatically generating image captions is a challenging research problem with real-world applications. Researchers at Google created a large image caption dataset, COCO (Chen et al., 2015), containing over 1 million captions for approximately 16,000 images, about 5 captions per image. Each caption is one simple English sentence that describes some aspects of the image. This dataset allows caption generation models to measure their performance by comparing their generated outputs against the reference captions.

A traditional approach to image caption generation is LSTM-based models. As demonstrated in (Vinyals et al., 2015), the images are encoded into visual features by CNN as the first input of LSTM. Then LSTM generates caption tokens until a stop token is found.

The performance of image captioning generation models has greatly improved since the birth of the attention mechanism (Vaswani et al., 2017). Attention enables the model to focus on specific regions of the images that are most relevant to the words generated in the caption. This interaction between visual features and textual features can greatly increase the accuracy and relevance of captions.

An example of a transformer-based caption generation model is GIT (Generative Image-to-text Transformer) (Wang et al., 2022). GIT combines an image encoder and a text encoder as inputs to a transformer decoder to generate the next token of the captions. It not only can be used for image caption generation, but also for answering questions about the image and for video caption generation.

Researchers at Alibaba Group developed the mPLUG model (Li et al., 2022), which is currently the state-of-the-art caption generation model on the COCO dataset. mPLUG is a cross-modal skip-connected network. By removing the cross-attention module of the visual encoder, mPLUG outperforms traditional cross-modal fusion networks.

In the field of image processing, CLIP (Radford et al., 2021) is a model that connects the visual with textual representations. CLIP is trained on a large dataset of image-text pairs, which allows the model to align the image contents with textual information. It is an ideal pre-trained image model for caption generation.

3 Dataset

The dataset used in this project is the COCO caption dataset (Chen et al., 2015). The original dataset is very large and consists of different caption types, such as person-focused captions. To simplify the process and allow fast training and testing, a small portion of the dataset is used, along with captions that provide general descriptions of the images.

The sub-dataset contains 5000 images, with each image associated with 4-6 captions. In total, it has 25011 captions. Each caption is a simple, one-line English sentence that describes the contents of the image, which contains no more than 45 words. For example, below is an image with its captions:

- two men in baseball uniforms running across a field.
- a couple of baseball players standing on a lush green field.
- two baseball players in the outfield of baseball field
- two baseball players running on the back of a baseball field.
- a baseball player stands deep in the outfield.

The task of this project is to generate captions for a given image to match the reference captions in various metrics. The task is challenging because it requires aligning the visual features with the textual features. Additionally, the model has to process various features of the image, since one caption may describe multiple items within the image.

3.1 Data preprocessing

Since this is a natural language processing class, this project only focused on the textual aspect of the task. A pre-trained CLIP from OpenAI (Radford et al., 2021) is used to convert images into 512-dimensional feature vectors. All of the models use the same feature vectors to represent the images.

For captions, the BERT auto-encoder is used to convert captions into tokens. The vocabulary is provided by the BERT auto-encoder, which uses a sub-word vocabulary with a size of 30,522. The maximum token length of a caption is 64 tokens.

3.2 Metrics

To measure the performances of the models, the following metrics are used:

- BLEU1: calculates the ratio of the matching unigrams between the generated and reference captions by the following formula:

$$\text{BLEU1} = \frac{\#\text{ of matching unigrams}}{\#\text{ of unigrams in the generated tokens}}$$

- BLEU4: calculates of ratio of the matching unigrams, 2-gram, 3-gram, and 4-gram, then calculate the mean of them.
- Meteor: the f1 score of precision and recall. The precision equals to BLEU1, which is the ratio of the matching unigrams out of the number of unigrams in the generated tokens. The recall is calculated by the ratio of the matching unigrams out of the number of unigrams in the reference tokens.

For images with several reference captions, the final score is the one with the highest value.

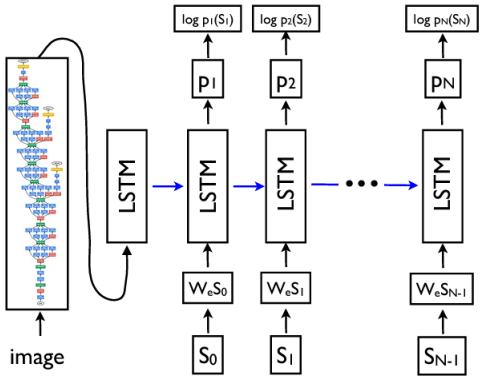
To select the best model, the one with the highest BLEU1 score on the validation set is used for evaluation on the test set.

4 Baselines

The baseline model is LSTM-based. The image feature vectors are connected to a linear layer to convert to a vector as the first input of LSTM. In each subsequent LSTM time step, the caption tokens are converted into embeddings and fed into the LSTM. The output of each LSTM time step is passed through a linear layer, which has dimensions equal to the vocabulary size. The next token is predicted deterministically by selecting the token with the highest probability from the output distribution. While generating the captions, the LSTM uses an auto-regressive manner. The output token is treated as the input of the next LSTM

time step. The model generates tokens until a stop token is detected.

Below is an image from (Vinyals et al., 2015) that summarizes the flow:



The loss function is Cross Entropy loss, and the optimization function is AdamW. Cross Entropy loss is used because the task is to predict the next token, which is a classification task. AdamW is used because it is considered the state-of-the-art optimization function for many machine-learning models.

The train/validation/test split is 80/10/10. After training, the model from each epoch with the best BLEU1 on the validation set is selected and applied to the test set to obtain the final results.

The hyperparameters:

- The embedding size and input size of LSTM: 256.
- The hidden size of LSTM: 512.
- Max token length: 75.
- LSTM number of layers: 2
- Epochs: 25
- Learning rate: 0.0005
- Batch size: 64

I tune the hyperparameters by training models with different hyperparameters and selecting the one with the highest BLEU1 score on the validation set. In addition, GPU capacity is a limitation. The hidden size and number of layers of the LSTM cannot be too large, since it would slow down the training process.

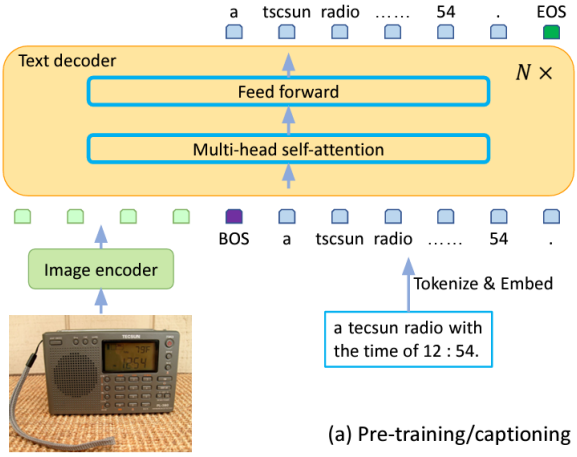
5 Approach

5.1 GIT (Generative Image-to-text Transformer)

The idea of the model is from this paper (Wang et al., 2022). During training, tokens are converted to token embeddings plus the positional encoding, and the image feature vector is passed through a linear layer to convert them into a vector of the same size as the token embeddings. The image vector is concatenated with the token embeddings and passed through a text decoder, which consists of multiple multi-head self-attention layers and feed-forward layers. Casual masking is applied to the multi-head self-attention to prevent seeing future tokens. Pre-norm is applied to the multi-head attention. The output of the text decoder is connected to a linear layer to predict the next tokens. Note, that the image vector does not have positional encoding.

During generating the captions, the output list initially contains the start token. In each step, the image vectors are concatenated with the embedding of current output tokens to generate a prediction for the next token. The tokens are generated deterministically since it yields the best validation BLEU1.

Below is a diagram from (Wang et al., 2022) that illustrates the model:



(a) Pre-training/captioning

The hyperparameters:

- The embedding size: 256.
- The hidden size of Feed Forward in decoder: 512.
- Number of multi-head layers: 6
- Number of heads: 2
- Dropout: 0.05

- Epochs: 8
- Learning rate: 0.0005
- Batch size: 64

The loss function is Cross Entropy loss, and the optimization function is AdamW. I tuned hyperparameters the same way as the baseline.

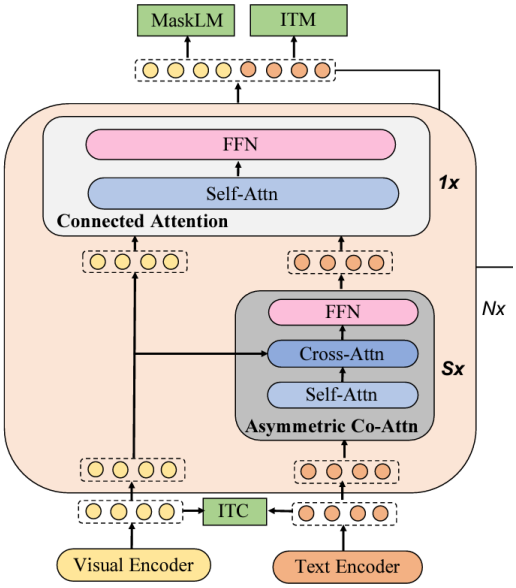
The model is successfully implemented in the file models/Attention.py and models/MultiHead.py. The multi-head attention class is copied from Assignment 2.

5.2 mPLUG

mPLUG is the state-of-the-art caption generation from this paper (Li et al., 2022). mPLUG is a large model with a lot of pre-training. To facilitate the training process, I modified the mPLUG.

The implementation of mPLUG is similar to GIT. The difference is in the multi-head attention. Without the image vector, token embeddings, plus the positional embedding, are passed to the text decoder. After each multi-head self-attention layer, an additional cross-attention layer is added to get the cross-attention of the image vector and the token embeddings. And, the last text decoder is the same as GIT, with no cross-attention layer. The image vector uses skip connections to be presented at every layer of the text decoder.

Below is a diagram from (Li et al., 2022) that illustrates the model:



Note, that the right part of this diagram is cut because the remaining part is not implemented.

The loss function is Cross Entropy loss, and the optimization function is AdamW. The hyperparameters are the same as the GIT. Both models share a lot of common structures, so their hyperparameters are the same.

The model is successfully implemented in the file models/mPLUG.py and models/MultiHead.py.

5.3 Compute

The experiments are run on i7 7700k and GTX 1080. Both GIT and mPLUG take more than half an hour to run the training.

5.4 Results

The results are shown below:

Model	BLEU1	BLEU4	Meteor
Baseline	0.622	0.199	0.419
GIT	0.628	0.213	0.423
mPLUG	**0.647**	**0.223**	**0.436**

Table 1: Compare the results

Both GIT and mPLUG are better than the baseline. mPLUG performs better than GIT in all three metrics.

6 Error analysis

GIT and mPLUG outperform the baseline because their attention mechanisms enable them to focus on relevant tokens, whereas LSTM does not have this ability. mPLUG can outperform GIT because of the skip connection of the image vector. However, both GIT and mPLUG do not significantly outperform the baseline.

Let us examine some of the unsuccessful generated captions.

Reference captions:

- a man in a kitchen pouring grain onto a scale.
- a man in a kitchen measuring out cheese.

- a man in a kitchen measuring something on a scale.
- a man pouring cheese on top of a food scale.
- a man dumps shredded cheese onto a weight.

Generated caption:

- Baseline: a man is preparing food in a kitchen.
- GIT: a man in a kitchen with a stove top.
- mPLUG: a man in a kitchen with a pot on a stove.

All of the models could not recognize scale and cheese because they are not in the training dataset. The size of training dataset influences the performance of the models.

Another example:

Reference captions:

- an umbrella strapped to a bar on a bicycle.
- an umbrella affixed to a bike with straps.
- a bike with an umbrella attached to it
- a bike parked next to a fountain with an umbrella attached to it.
- a photo of a bike with a red umbrella attached to it

Generated caption:

- Baseline: a bicycle with a basket and a basket on it.
- GIT: a bicycle with red umbrella attached to a red umbrella.
- mPLUG: a blue bike with a basket on top of a bicycle.

The baseline and mPLUG recognize the umbrella as a basket, and both baseline and GIT's captions repeat themselves. It shows that when items are stacked together in the images, these models tend to not perform well.

Another example:

Reference captions:

- a couple of people standing in a kitchen preparing food.
- a man making a funny face and a woman holding a pot in a kitchen
- a man looks angry as he stands in the kitchen.
- a young man and older woman are shocked about something.
- a woman holds a pan in the kitchen as a man in a hat makes a face near her.

Generated caption:

- Baseline: a man and a woman standing in a kitchen.
- GIT: a woman standing in a kitchen with a stove and a stove.
- mPLUG: a man standing in a kitchen with a woman in a kitchen.

The models failed to read facial expressions. And mPLUG starts to repeat itself.

After examining these failed examples, it is clear that the models can recognize items in an image. However, they struggle to understand the positions, interactions, and actions of the people and items within the image. It also struggles when the items stack together. The model tends to repeat some words.

To make the model understand positions, interactions, and actions, a larger model is needed to store and process more visual information during caption generation. Additionally, pre-trained image features are not enough to capture all the aspects needed for caption generation. It would be better if the pre-trained image model could be fine-tuned during training. Then, training on a larger dataset can help the model recognize some rare items. Lastly, a pre-trained language model can teach the model to speak English with proper grammar (which is the part that I did not implement in the original mPLUG).

7 Conclusion

In conclusion, for the given dataset and image feature vectors, mPLUG and GIT perform better than the baseline model.

My takeaway is that the transformer-based model is better than the LSTM-based model in the caption generation task. In addition, mPLUG outperforms GIT because the skip connections allow the model to process visual information at every step.

It has proven surprisingly difficult to convert visual information into textual features without losing any details. The failed examples show that the models cannot detect actions or facial expressions in the images, which means this information is lost either during image feature generation or caption generation.

I am a bit surprised that the baseline model (LSTM decoder) is not significantly worse than the transformer-based models. Although LSTM is an older model, it can still generate some appropriate captions. The design of the LSTM cell allows it to generate sequential tokens while remembering the visual information provided in the initial input.

If I could continue working on this project, I would first investigate how to teach the model to speak proper English. The generated captions may capture the main idea of the image, but some are not readable by either humans or machines. Perhaps a caption generator guided by a LLM could produce more suitable captions

8 Acknowledgements

For the writing part of this project, I use ChatGPT as a grammar corrector. I wrote the sentences and put them into ChatGPT to check for any grammar errors. I will edit the sentences if ChatGPT spots any grammar errors. Major changes are made to ChatGPT's output.

For the coding part of this project, I ask ChatGPT to learn more about this topic, such as "List state-of-the-art pre-trained image models to convert the images to latent space representations." Also, I ask ChatGPT how to use a package, such as "How do I use pre-trained CLIP from hugging face to convert image into features?" or "How do I load a json file?". Major changes are made before running the codes.

References

Chen, X., Fang, H., Lin, T.-Y., Vedantam, R., Gupta, S., Dollar, P., and Zitnick, C. L. (2015). Microsoft coco captions: Data collection and evaluation server.

Li, C., Xu, H., Tian, J., Wang, W., Yan, M., Bi, B., Ye, J., Chen, H., Xu, G., Cao, Z., and et al. (2022). Mplug: Effective and efficient vision-language learning by cross-modal skip-connections.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., and et al. (2021). Learning transferable visual models from natural language supervision.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator.

Wang, J., Yang, Z., Hu, X., Li, L., Lin, K., Gan, Z., Liu, Z., Liu, C., and Wang, L. (2022). Git: A generative image-to-text transformer for vision and language.