A Visual Attention Image Captioner Based on Transformer

Dingzhi Yu
Artificial Intelligence, School of Data Science
ID: 20307140031
20307140031@fudan.edu.cn

Abstract

In this project, we developed a Image caption model mainly based on transformers. The architecture is inspired by [1], which uses pretrained `mobilenetV3` to extract features from image, then passed to the cross-attention layers of the Transformer-decoder. The transformer decoder is mainly built from attention layers, which uses self-attention to process the sequence being generated, and it uses cross-attention to attend to the image. To evaluate the model, we propose a image attention visualization method by inspecting the attention weights of the cross attention layers to see what parts of the image the model is looking at as it generates words.

1 Introduction

Automatically generating captions of an image is a task very close to the heart of scene understanding—one of the primary goals of computer vision. Not only must caption generation models be powerful enough to solve the computer vision challenges of determining which objects are in an image, but they must also be capable of capturing and expressing their relationships in a natural language.

It is important to point out that the framework in the project is **tensorflow**.After exploring the flickr30k dataset, we decide to change this seq2seq task into an one end sequence generation task. That's to say, we no longer regard the images as a genre of sequence, but specifically view it as images. The main benefit of this change is allowing us to tackle the input more wisely by using existed pretrained imagenet model. Therefore, we first implement an image feature extractor, i.e. image encoder, whose core part inherits from `keras.application.MobileNetV3Large`. Then, we implement the transformer decoder model layer by layer manually. The model temporarily assumes that the pretrained image encoder is efficient, and just focuses on building the text decoder. After the decoder is ready, we specifically contruct an output layer to generate logit-predictions for each token at each location. The one thing that distinguishes this output layer is that it not only handles bad tokens, but also smartly initializes to substantially decrease the loss and thus helping to optimize the initial bias. The rough structure of our model could be described in 1

Preprint. Under review.

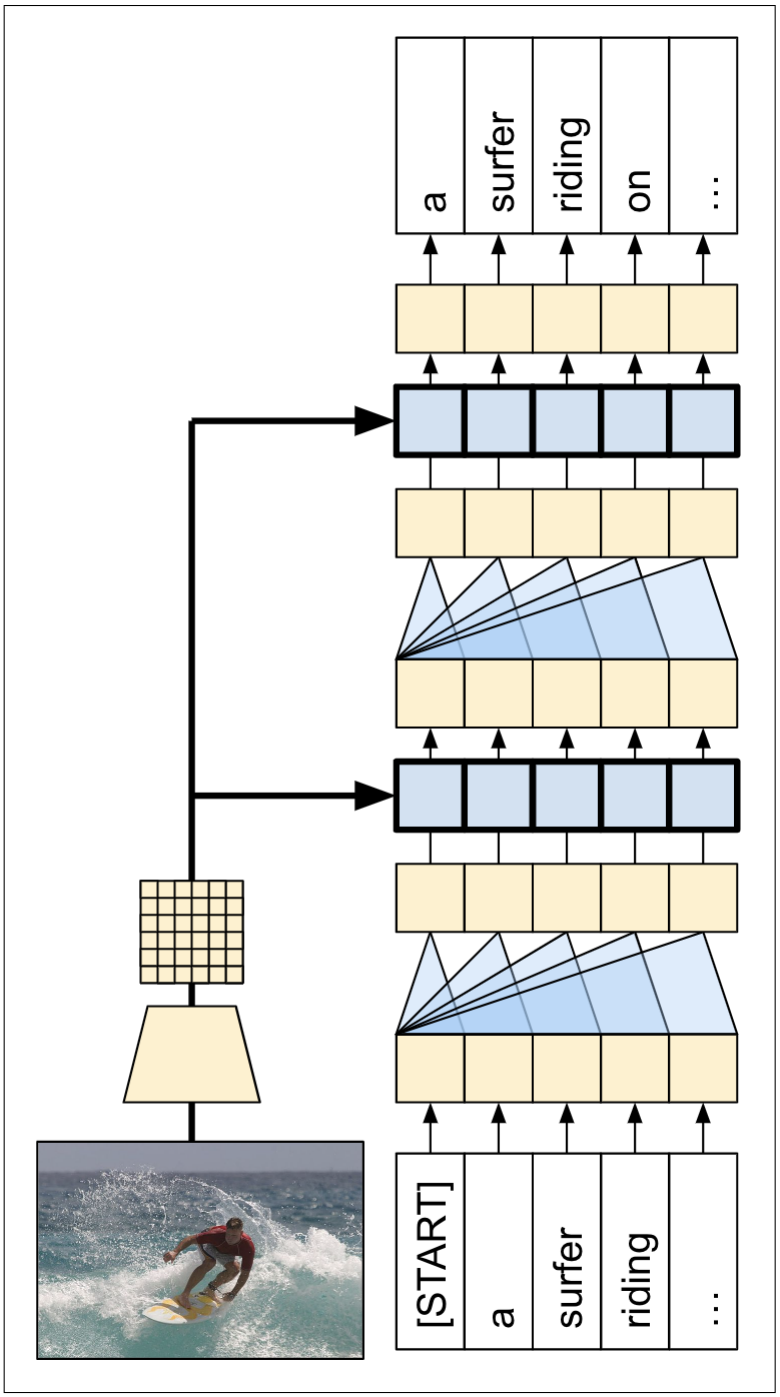

Figure 1: The model structure

2 Dataset

The flickr30k dataset[2] contains 31,783 images collected from Flickr, together with 5 reference sentences for each image provided by human annotators(158915 captions). The length of the caption approximately ranges from 5 to 50. The image has a common width of 500px. We can use `pandas` library to conveniently load the primal data and deal with it.

In the exploratory data analysis, we put out work together in `eda.py`. We first preprocess the data a little bit into something like `tf.data.Dataset` pieces and split them into train, validation and test datasets. This concludes our raw datasets. Then, we implement our image encoder with a pretrained imagenet, along with necessary functions to tackle with images. Then we implement the tokenizer and the vocabulary, constructs text tokenier and word-to-index and index-to-word layers for latter uses. Finally, we prepare ourselves with the ready datasets. They stems from the raw datasets because their items are one to one (image, caption) pairs instead of one to many. Moreover, we provide fuctions which allow you to save and load the ready dataset from memory. It makes the project easier to transplant.

3 Methodology

3.1 Image Encoder

Perhaps the most tricky part in the project was the image encoder. After exploring the dataset, we found that the images has a common width of $m = 500$. It's great news for this seq2seq task because in that way, we can see the image as stacks of vectors. The output text is also naturally a stack of vectors, if we use word vector representation along with the appropriate sentence split. Then our job is to find a function f which satisfies

$$f : \{x_1, \cdots, x_p\} \in \mathbb{R}^{n\times p} \mapsto \{y_1, \cdots, y_q\} \in \mathbb{R}^{m\times q} \tag{1}$$

where p, q is variant, $m \in \mathbb{N}$ is the dimension of the word vector. However, we abandon this approach because if not implemented well, the sequential encoder for the input images will easily lose the locality information of the image. The locality feature of the image can **not** be effectively captured by standard attention sequential encoders, and normal positional encodings still fail to read the image wisely. Therefore, we use a pretrained image feature extractor with the last classifier layer eliminated, hoping it can efficiently capture the main information of the input image. Moreover, by setting its `trainable` attribute to `False`, we avoid the further training on this network. Should the pretrained mobilenet effective, we can save a lot of time and computing power of the encoding process. Except for the encoder, we design a function to accept all sizes of any given image, rescale it and then pass it to the `mobilenet`. The rescaling method is lanczos5 interpolation method, which guarantees the high quality of the transformed image, preserving as much information as possible.

3.2 Transformer Decoder

This model assumes that the pretrained image encoder is sufficient, and just focuses on building the text decoder. We uses a 2-layer transformer-decoder. The model will be constructed in the following 3 main parts in `transformer_decoder.py`. Due to the limited time, digging into `transformer_decoder.py` where all the details are listed is strongly suggested.

3.2.1 Input

This part includes the token embedding and positional encoding (SeqEmbedding). The input text has already been split up into tokens and converted to sequences of IDs. Remember that unlike a CNN or RNN the Transformer's attention layers are invariant to the order of the sequence. Without some positional input, it just sees an unordered set not a sequence. So in addition to a simple vector embedding for each token ID, the embedding layer will also include an embedding for each position in the sequence. The SeqEmbedding layer has the following features:

1. It looks up the embedding vector for each token.
2. It looks up an embedding vector for each sequence location.
3. It adds the two together.
4. It uses `mask_zero=True` to initialize the keras-masks for the model.

What's more, our implementation learns the position embeddings instead of using fixed embeddings.

3.2.2 Decoder

The decoder is a standard Transformer-decoder, it contains a stack of DecoderLayers where each contains three sublayers:

1. A causal self attention later (CausalSelfAttention), where each output location can attend to the output so far.
2. A cross attention layer (CrossAttention) where each output location can attend to the input image.
3. A feed forward network (FeedForward) layer which further processes each output location independently.

In our implementation, we arrange these three layers into a larger `DecoderLayer`. Each decoder layer applies the three smaller layers in sequence. After each sublayer the shape of output sequence is `(batch, sequence, channels)`. The decoder layer also returns the attention scores for later visualizations.

3.2.3 Output Layer

At minimum the output layer needs a layers.Dense layer to generate logit-predictions for each token at each location. Whereas, there are a few other features you can add to make this work a little better.

1. **Handle bad tokens**: The model will be generating text. It should never generate a pad, unknown, or start token (", '[UNK]', '[START]'). So set the bias for these to a large negative value, and don't forget to ignore these tokens in the loss function as well.
2. **Smart initialization**: The default initialization of a dense layer will give a model that initially predicts each token with almost uniform likelihood. The actual token distribution is far from uniform. The optimal value for the initial bias of the output layer is the log of the probability of each token. So include an adapt method to count the tokens and set the optimal initial bias. This reduces the initial loss from the entropy of the uniform distribution (`log(vocabulary_size)`) to the marginal entropy of the distribution (`-p*log(p)`).

3.3 Captioner

To formally build the model, we need to combine the encoder and the decoder into a image captioner. The several parts are to sum up:

1. The image `feature_extractor` and the text `tokenizer`
2. The `seq_embedding layer`, to convert batches of token-IDs to vectors (`batch, sequence, channels`).
3. The stack of `DecoderLayers` layers that will process the text and image data.
4. The `output_layer` which returns a pointwise prediction of what the next word should be.

When we call the model for training, it receives an (`image, txt`) pair. To make this function more usable, be flexible about the input:

- If the image has 3 channels run it through the feature_extractor. Otherwise assume that it has been already.
- If the text has dtype `tf.string` run it through the tokenizer.

After that running the model is only a few steps:

1. Flatten the extracted image features, so they can be input to the decoder layers.
2. Look up the token embeddings.
3. Run the stack of decoderLayers, on the image features and text embeddings.
4. Run the output layer to predict the next token at each position.

To utilize the code, we wrap up the above functions into a `Captioner` class in `transformer_decoder.py`. Our code has all been carefully encapsulated for readability as well as transplantability. The whole architecture of the model can be depicted in 2.

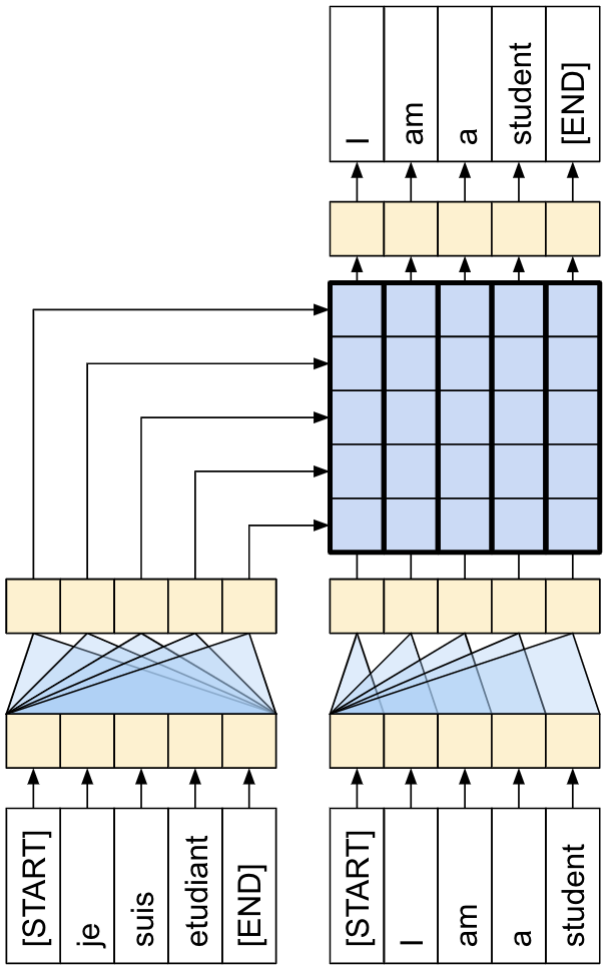

Figure 2: The architecture of transformer encoder and decoder

4 Results

4.1 Training History

The training history are listed as follows. The accuracy on training set and validation set are in 3. The loss on training set and validation set are in 4.

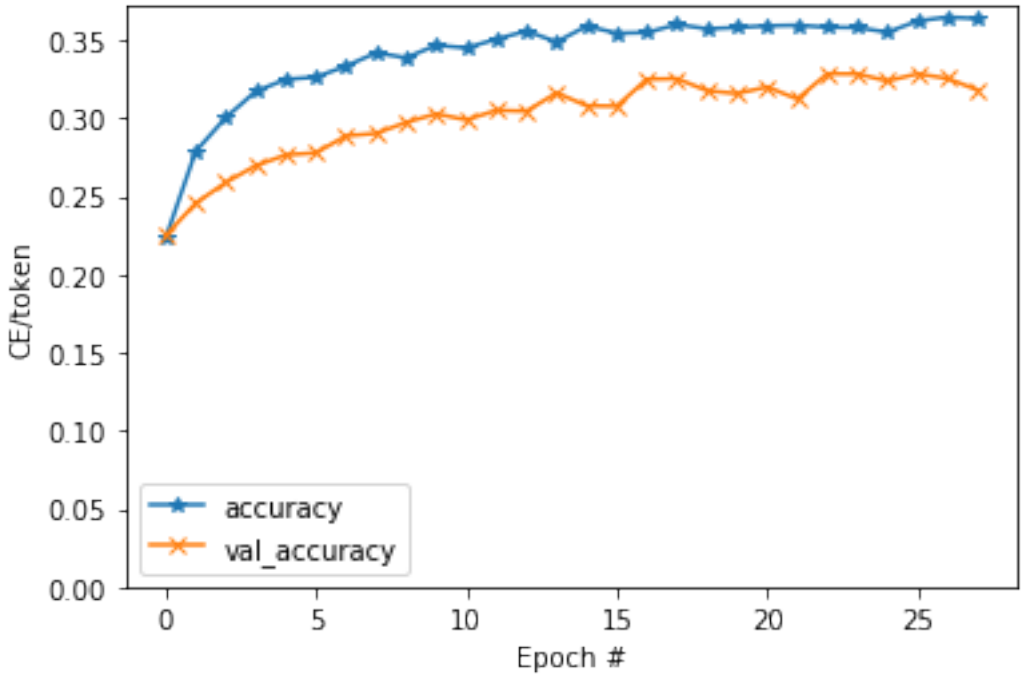

Figure 3: Accuracy

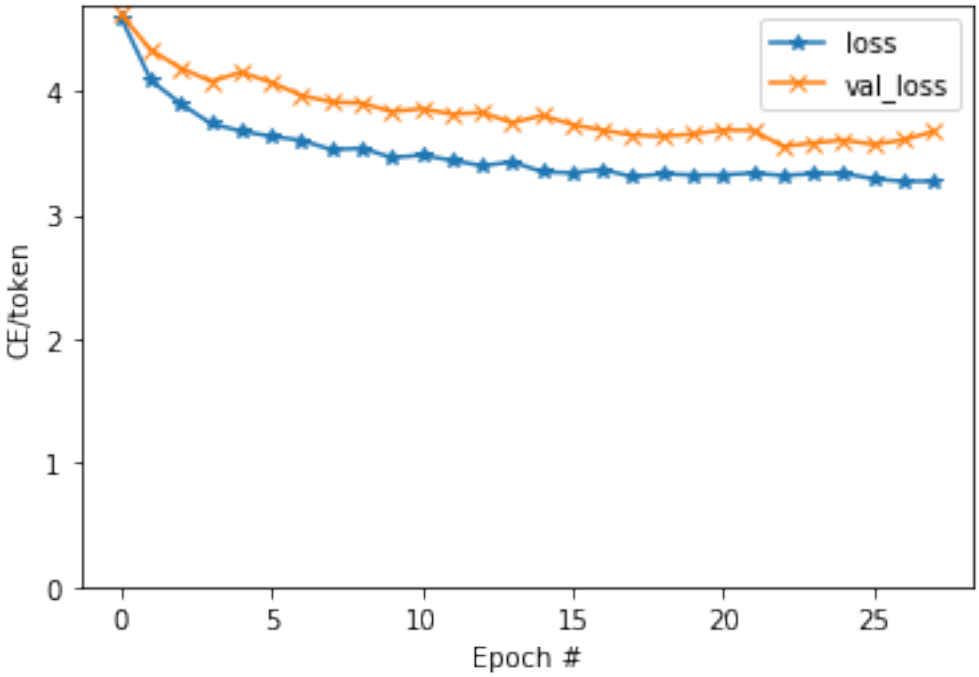

Figure 4: Loss

Our implementation of accuracy and loss are masked accuracy and masked loss. When calculating the mask for the loss, note the loss < 1e8. This term discards the artificial, impossibly high losses for the banned_tokens. On the other hand, we setup a `keras.callbacks.Callback` to generate some captions for the surfer image at the end of each epoch for feedback during training Our optimizer

choose the Adam with the initial learning rate of 1e-4. There have been experiments shown that Adam suits more for flickr30k while RMSprop is more appropriate for flickr8k. The training hyperparameter is difficult to choose. With the help of `keras_tuner`, we can find that reduce the number of attention heads as well as the number of layers of the network is not only helpful in speed, but also in accuracy.

4.2 Image Attention Visualization

It's inspirational to implement a visualization technique like this. We constructed a image attention visualization to vividly show where the model was focusing attention while generating each token of the output. We also wrap this into a usable member funtion of the class `Captioner`. Here, we select a few results of our attention plots.

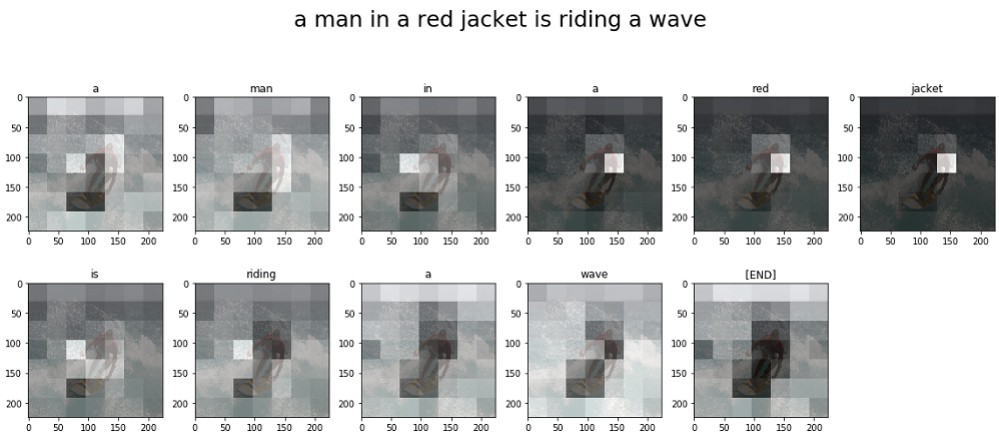

Figure 5: Test case 1

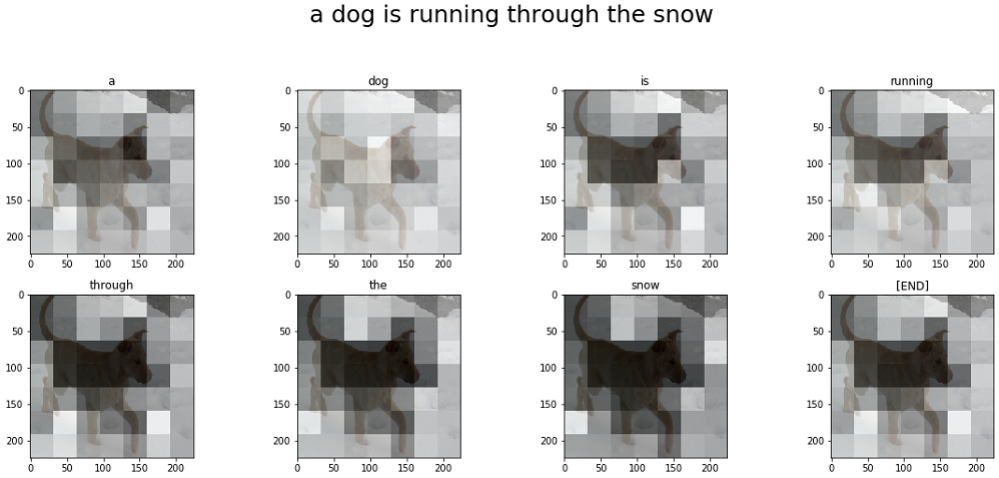

Figure 6: Test case 2

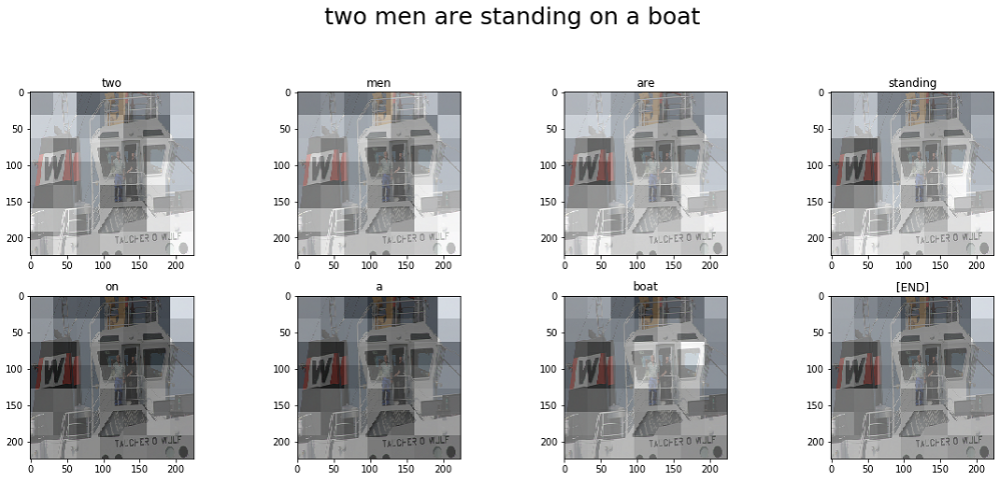

Figure 7: Test case 3

As in Figure 5,6,7, our model has some ability to generate a caption for a given image. When it comes to model evaluation, we shouldn't be excessively superstitious about metrics like accuracy or losses. They only partially reflects the ability of the model and can only directly contribute to interpretability not capability.

5 Conclusion

In this final project, we implement a transformer-based model with a pretrained image encoder and a transformer decoder on flickr30k dataset. We did **not** apply any data augmentation or image processing techniques to the dataset because we'd like to focus more on the performance of the model. Using a pretrained network also contributes to concentrating to upgrade our decoder.

After some exploration, the performance of the model is not satisfactory enough due to the limited samples of training set and the architecture of the model. In exploratory data analysis, we found that the dataset also has imbalanced sample. For instance, the pictures may have too many elements in common like dogs, water, surfer, etc. This results in the caption with too many 'blue' ,'man', 'stand', etc. When it comes to model, our architecture is not perfect enough. The relationship between our encoder and decoder is not close enough, and our ability of the model is highly likely to be affected by the pretrained `mobilenetV3`. It's not safe to say that transformers is dominant over RNN. At least on this task, RNN also has its stage as in [3]. Moreover, our pretrained encoder can also be replaced by transformer encoder, if the transformer encoder is well implemented. Although the pretrained imagenet is powerful in image classification and feature extraction, it's still potientially weaker than a transformer feature extractor. Have we get enough time, we can use fine-tuning technique to optimize the hyperparameters. We can also try more transfer learning to increase the generalization ability of the model. As for the two-end separate transformer encoder-decoder model widely used, it can still be improved into a shared multi-layer transformer network for both encoding and decoding as in [4]. But in this case, the model presented in this report still has its advantages, like we said in the previous parts. The ideal scenario when designing the architecture is that with the help of the image encoder, the model should be strong when dealing with images that are far different from the training dataset. After the entire semester of this course, I could gradually handle this real life task and improve the performance of the model step by step.

References

[1] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention, 2015.

[2] M. Hodosh P. Young, A. Lai and J.Hockenmaier. From image description to visual denotations: New similarity metrics for semantic inference over event descriptions., Transactions of the Association for Computational Linguistics (to appear).

[3] Xu Jia, Efstratios Gavves, Basura Fernando, and Tinne Tuytelaars. Guiding long-short term memory for image caption generation, 2015.

[4] Luowei Zhou, Hamid Palangi, Lei Zhang, Houdong Hu, Jason J. Corso, and Jianfeng Gao. Unified vision-language pre-training for image captioning and vqa, 2019.