An Image Captioning Model with CNN and RNN

Jiasheng Ni, Muqing Yang, Ruochen Miao

Abstract

In our project, we try to apply the knowledge of deep learning to design an image captioning model. It can capture and understand features from random input images, and then generate a reasonable caption to describe the content of the image. We notice that many existing image captioning models do not obtain the attention mechanism and are flawed in their performance, so we add the attention mechanism into the model construction. After researching, we build an encoder-decoder model with attention for image captioning. We use the CNN model as the encoder and try on two pre-trained models: ResNet50 and VGG-19. Then we use the RNN model as the decoder and compare the performance between LSTM and GRU respectively with global attention, local attention or no attention. We apply different optimizers to test the gradient descent speed and model accuracy. After model evaluation, we choose the model that is composed of a pre-trained VGG-19 CNN encoder and a one-layered LSTM RNN decoder with local attention. This model can effectively identify the elements in the image and generate captions. As for the demonstration, we build a web application allowing users to upload an image and show the caption generated by our model.

1 Introduction

Starting from the ImageNet campaign held by Li Feifei in 2012, the combination of Computer Vision (CV) and Natural Language Processing (NLP) has always been a heated topic. Among all related research topics, Image Captioning is the most prominent and representative one in this field. Typically, Image Captioning refers to the computers generating one or multiple sentences after understanding the visual elements of a given image. To generate an appropriate and meaningful caption from a random image, the model should not only be able to detect the fore-ground objects and the confounding background, but also have a strong ability to analyze the status and peculiar features of these objects, as well as the relationship between them. It has various applications such as recommendations in editing applications, usage in virtual assistants, image indexing, recognizing visually impaired persons, generating items for social media, etc.

Although Image Captioning is a very complex task in the field of deep learning, many scientists have proposed great improvements on the accuracy and conciseness of Image Captioning models. Existing Image Captioning methods includes template-based image captioning, retrieval-based image captioning and novel caption generation. However, most of these models don't have the attention mechanism. Due to the vanishing and exploding gradient problem, RNNs cannot remember long sentences and sequences, they can only remember the part just seen. Therefore, without the attention mechanism, the encoder may create a bad summary when trying to understand a long sentence, which will lead to the bad explanation. The performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases.

Each time the proposed model generates a sentence, the attention mechanism searches for a set of positions in the hidden states of the encoder where the most relevant information is available. It improves RNNs' performance in processing lon¿r sentences and sequences, and enables RNNs to pay more attention to some input words than other words in sentence translation.

In our project, we build a flexible encoder-decoder with attention model for image captioning, in which CNN model is used as encoder and RNN model as decoder. For CNN model selection, we try two pre-trained models: ResNet50 and VGG-19 to see which one performs better. Then for RNN model construction, we compare the performance of LSTM decoder and GRU decoder under global attention, local attention and no attention respectively. After that, we apply different optimizers to test the speed and efficiency of gradient descent. Finally, As for the demonstration, we build a web application allowing users to upload an image. After the image is uploaded, the web application shows the caption generated by the trained model as well as the visualized attention assigned to each word during the generating process.

2 Data

2.1 Dataset

The datasets that we use come from Kaggle, which contain thousands of RGB images, each paired with five different captions. The captions provide clear descriptions of the salient entities and events.

We mainly use the Flickr 8k dataset[1] as training and validation set. It is an online dataset consisting of 8,000 images and 40,000 captions. The images were chosen from 6 different Flickr groups and tended not to contain any famous people or locations. Instead, they were manually selected to depict various scenes and situations. We primarily use this dataset for our model training and validation since it contains image descriptions close to human style.

We also use the Flickr 30k dataset[2] as testing set. It is an online dataset consisting of 31,800 images and 159,000 captions. It is more complete and diversified than the Flickr 8k dataset, but the format and the image type are the same. Therefore, we randomly select certain samples from the Flickr 30k dataset for

model testing.

2.2 Caption Processing

2.2.1 Caption Tokenization and Numericalization

In order to accomplish the image captioning model, we first use word splitting techniques to preprocess the origin captions. We use Spacy library to split the given captions to create word dictionaries. After that, we numericalize each token and create a mapping from each token to index, and vice versa, for future embedding operations. Finally, we get a word dictionary of 8504 unique words and 4 auxiliary tokens, which are "<PAD>", "<SOS>", "<EOS>", "<UNK>".

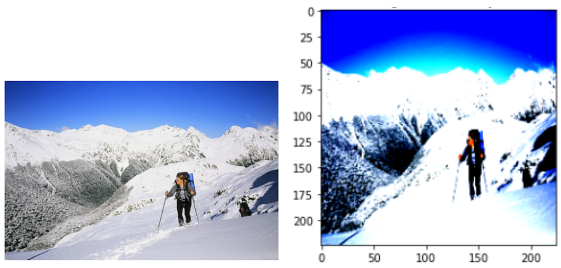
<PAD>	Pad a given caption with the same length for batch operations in RNN model.
<SOS>	Mark the start of the target caption, also notifying RNN to start generating captions for images.
<EOS>	Mark the end of the target caption, also notifying RNN to stop generating captions for images.
<UNK>	Represent the generated token that does not appear in the dictionary we defined.

2.2.2 Word Embedding

In order to accurately represent the relationship between different words and produce less context bias in caption generation, we try two word embedding techniques in the model. The first one is pytorch built-in randomized embedding (nn.embedding) with an embedding matrix of the size [vocab_size, defined_embedding_size]. The second one is pretrained Glove, a word Embedding layer based on Colinear matrix. Since its pretrained embedding mapping does not contain the embedding of 4 self-defined tokens ("<PAD>", "<SOS>", "<EOS>", "<UNK>"), we create a new embedding matrix [vocab_size, original_embedding_size+4] for them, and extend the dimension correspondingly to the rest of the words in the embedding dictionary. However, due to our undersized vocabulary size, the performance of pre-trained Glove is poor, so we finally choose the pytorch built-in embedding technique.

2.3 Image Processing

For image pre-processing, a common operation of geometric transformation is to resize the images to the same size for uniform processing. The larger the picture is, the higher the configuration required for processing. Since we only have limited computational resources(GTX1660Ti 6G), the configuration of the machine should also be taken into account. By researching, we find that 224*224*3 image size seems to work well for Resnet50 and VGG-19. Therefore in our project, we firstly resize the images to a size less than 500*500*3 for processing[3]. On top of that, we adopt a random crop to the image, which is equivalent to establishing the weight distribution between each feature of either the fore-ground objects or the background factor. In this way we weaken the weight of noise and make the model less sensitive to missing values, so as to prevent overfitting, produce better learning effect and improve the robustness of the model[4]. After that, we perform another transformation on the image, namely standardization[5]. In most cases, the pixel distribution of the red, blue and green channels of the trained images is not similar, that is, the value of one channel is too large or too small. This will make the gradient descent more difficult to converge. Therefore in our model, we normalize the image pixels to perform the transformation. After resizing, random cropping and standardizing, the comparison between the initial image and the processed image is shown in Figure 1.



Original Processed

Figure 1. Image Processing

3 Methodology and Methods

3.1 Encoder

3.1.1 Model Introduction

For the encoding part, we define a CNN model, where we take a processed RGB image as the input and extract features from the input image. Since we will input the feature matrix after convolution and max pooling directly into the decoder, we do not need to flatten the matrix. Therefore, instead of using the outputs from the last fully connected layer of the traditional CNN model, we use the layer before these fully connected layers as the extracted features for the given image.

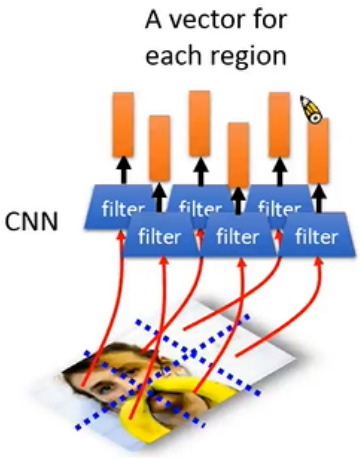
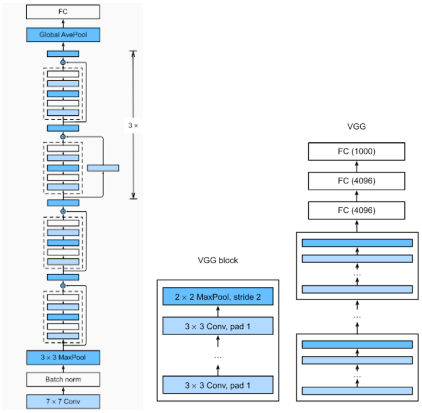


Figure 2. CNN

3.1.2 Model Choice and Parameters

Due to different encoder structures, the size of the extracted features also differ. The size of the extracted feature matrix for ResNet50 is [batch_size, 49, 2048], while [batch_size, 49, 512] for VGG-19. Although the features extracted from VGG-19 are less, it is deeper in level, therefore it has a higher degree of fitting to the training set, however suffering the risk of overfitting.



ResNet VGGNet

Figure3. ResNet and VGGNet

3.2 Decoder

3.2.1 Model Introduction

For the decoding part, we define one-layered LSTM network and GRU network models with Global attention, local attention and no attention respectively, then input the features extracted from the encoder. The traditional image captioning system uses the extracted features generated by CNN to encode the image. Then, it decodes these extracted features using LSTM or GRU network and recursively generates each word in the caption for a given image. On this basis, we apply the attention mechanism in order to generate more accurate and eclectic captions.

3.2.2 Model Structure

In the RNN model structure of our project, we build a LSTM network and a GRU network respectively for the subsequent model evaluation.

```
MyDecoderRNN(
  (embedding_layer): Embedding(8508, 128, padding_idx=0)
  (attn): Attention(
    (Wa): Linear(in_features=2560, out_features=512, bias=False)
    (Va): Linear(in_features=512, out_features=1, bias=True)
  )
  (dropout): Dropout(p=0.5, inplace=False)
  (init_hidden): Linear(in_features=2048, out_features=512, bias=True)
  (init_cell): Linear(in_features=2048, out_features=512, bias=True)
  (lstm_cell): LSTMCell(2176, 512)
  (fcn): Linear(in_features=512, out_features=8508, bias=True)
)
```

Figure 4. LSTM

```
MyDecoderRNN(
  (embedding_layer): Embedding(8508, 128, padding_idx=0)
  (attn): Attention(
    (Wa): Linear(in_features=2560, out_features=512, bias=False)
    (Va): Linear(in_features=512, out_features=1, bias=True)
  )
  (dropout): Dropout(p=0.5, inplace=False)
  (init_hidden): Linear(in_features=2048, out_features=512, bias=True)
  (init_cell): Linear(in_features=2048, out_features=512, bias=True)
  (gru_cell): GRUCell(2176, 512)
  (fcn): Linear(in_features=512, out_features=8508, bias=True)
)
```

Figure 5. GRU

However in practice, although these two networks can store certain history information, each word generated by them only describes a very limited part of the image, often ignoring the relationship between different image pixels and the essence of the entire image. What's more, if we use all the pixels of the image to generate words, we cannot effectively generate different words for different parts of the image. That's why we introduce the attention mechanism.

3.2.3 Attention Mechanism

The fixed-length vector carries the burden of encoding the entire "meaning" of the input sequence, no matter how long it is. With all the variance in language, it is a rather difficult task. Imagine two nearly identical sentences of 20 words with only one word difference between them. Both encoders and decoders must be nuanced enough to represent this change as a slightly different point in space. In order to address such problem for a better generation of captions, we apply the attention mechanism, introduced by Bahdanau et al.[6] and improved by Luong et al.[7], to our RNN model, The attention mechanism solves this problem by giving the decoder a way to "pay attention" to parts of the input, rather than relying on a single vector. For each step, the decoder can select a different part of the input sentence to consider. Figure 6 shows hte basic algorithm behind attention.

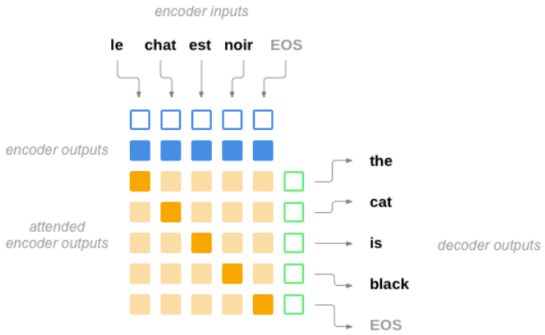


Figure 6. Attention

In our project, we compare local attention, global attention and no attention to strike a balance between computational speed and model performance. Figure 7 shows the algorithm of the global attention model and the local attention model.

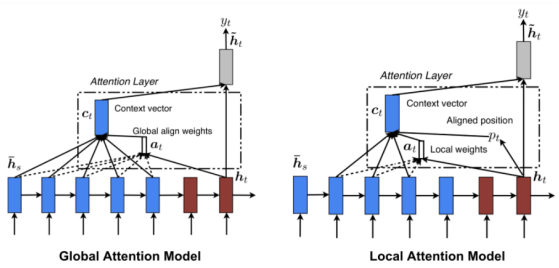


Figure 7. Global Attention and Local Attention

Global attention implies we attend to all the input words, and local attention means we attend to only a subset of words.[7] Here we use the "concat score function" proposed by Luong et al.[7] to calculate the alignment vector (attention weight in general). Given the alignment vector as weights, the context vector c_t is calculated as the weighted average of all different parts of a given image.

$score(h_t, \bar{h_s}) = v_a^T W_a[h_t \cdot \bar{h_s}]$

h_t is the hidden state for the current time setp, $\bar{h_s}$ is the context vector obtained by a weighted average of previous time step, v_a^T and W_a are adjusting vector and matrix helping to align the size of the input for RNN model.

4 Model Evaluation

Due to the limited computer resources, we test the model performance by setting batch size to 156 and epoch to 10 to perform the gradient descent.

We start with the ResNet50 as encoder, and LSTM as decoder to get a taste of the image captioning algorithm, finding that this model performs well in generating short or middle length captions. Then, we decide to compare the model performance of various kinds of neural cells, attention mechanisms and CNN model structures.

For comparison, we set the training epoch to 5 due to the time limitation. We first capture a captioned image at the beginning of training as the initial performance of this model. Then after training, we select several captioned images near the end of the epoch, in order to compare the performance of the models. We also use one same image to test these models, in order to intuitively compare the performance of various models.

4.1 Comparison of Different Attention Mechanisms

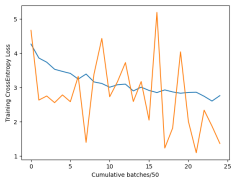
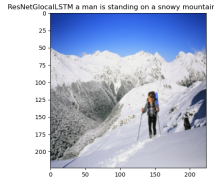
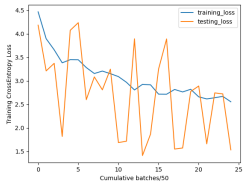
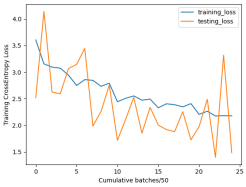
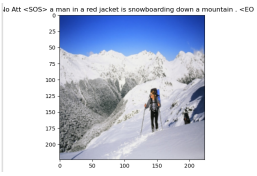
First, we compare the performance of different attention mechanisms. We use the same CNN and RNN settings along with global attention, local attention, and no attention.

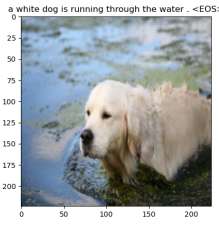
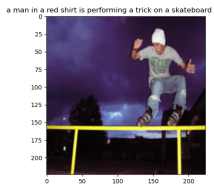
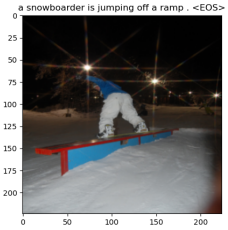
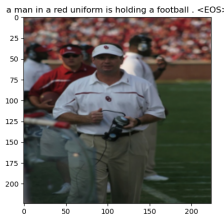
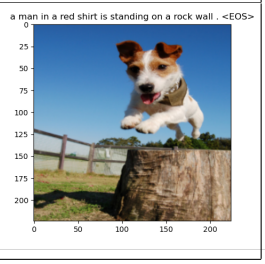
Shared settings:

CNN	RNN	Batch Size	Initial Learning Rate
ResNet50	LSTM	156	0.001

Different settings:

Attention	Dropout Rate	Five Epochs Training Time
Global Attention	0.5	3773.5s
Local Attention	0.5	3244.7s
without Attention	0.3	2854.6s

Attention	Loss	Comparison
Global		
Lobal		
Without		

Attention	Good	Bad
Global		
Lobal		
Without	doesn't exist	

Due to the limitation of computational time, we calculate the testing loss in the process of training. The results show that for image captioning tasks, the variance of test error is much larger than that of training error, and their convergence direction is the same.

We also found that after introducing the attention mechanism, the performance of the model has been greatly improved. What's more, although the training time is shortened a lot without attention, the model captures the features poorly. We will analyze this point more deeply and concretely in the summary of model comparison.

In the comparison between global and local attention, since the global attention model takes more pixels into consideration, it slightly improves the accuracy of the model at the cost of less

computing speed. Generally speaking, the difference between these two attention mechanisms is not obvious.

4.2 Comparison of Different RNN Models

Next, we compare the performance of different RNN neural cells among GRU and LSTM.

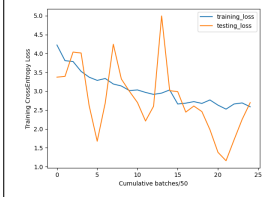
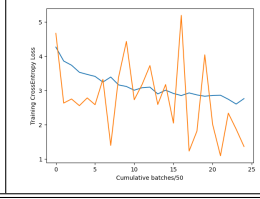
Shared settings:

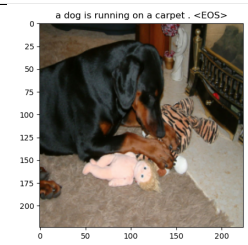
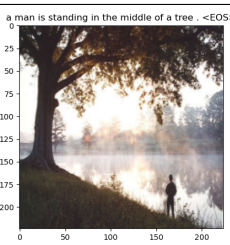
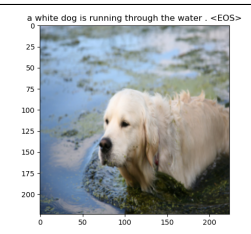
CNN	Attention	Batch Size	Initial Learning Rate	Dropout Rate
ResNet50	Global Attention	156	0.001	0.5

Different settingss:

RNN	Five Epochs Training Time
GRU	3738.8s
LSTM	3244.7s

The detailed parameter list is in Appendix I.

RNN	Loss	Comparison
GRU		
LSTM		

RNN	Good	Bad
GRU		
LSTM		

We find that the training time of the GRU model is slightly faster than the LSTM model, and the accuracy of the testing results is similar. This is possibly because we use a relatively small dataset in model training.

4.3 Comparison of Different CNN Models

Finally, we compare the performance of different CNN models among VGG-19 and ResNet50. Due to limited computational resources, we only test on local attention to perform the comparison.

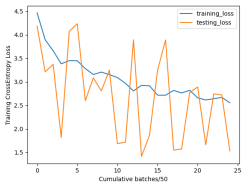
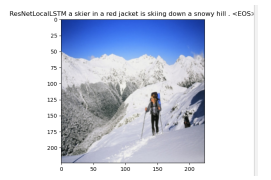
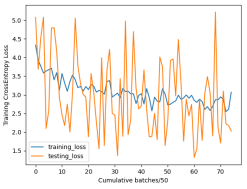
Shared settings:

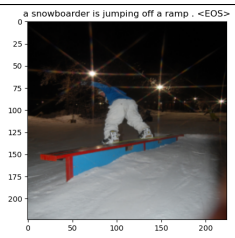
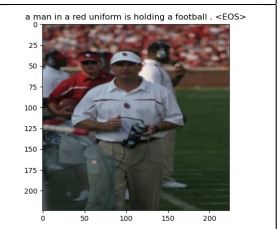
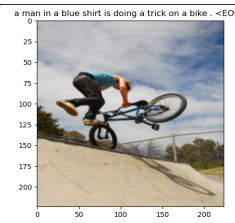
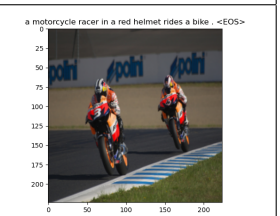
RNN	Attention	Batch Size	Initial Learning Rate	Dropout Rate
LSTM	Local Attention	156	0.001	0.5

Different settingss:

CNN	Five Epochs Training Time
ResNet50	3244.7s
VGG-19	4694.2s

The detailed parameter list is in Appendix II.

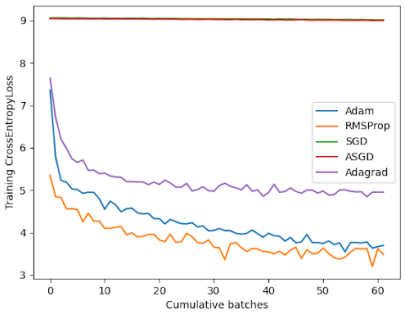
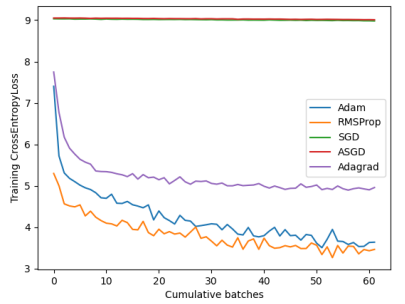
CNN	Loss	Comparison
ResNet50		
VGG-19		

CNN	Good	Bad
ResNet50		
VGG-19		

4.4 Optimizer Choices

For the model evaluation part, we start with Adam optimizer, followed by RMsProps, SGD, ASGD, Adagrad to test the gradient descent efficiency.

We test different optimizers with the same settings. We use ResNet50 for CNN, LSTM with Global Attention for RNN, 0.001 as the learning rate, 128 as the batch size, cross entropy loss as the loss function and train to the fifth epochs. The detailed parameter list is Appendix III.

Learning Rate	Training Loss
0.001	
0.0003	

We can see that when the learning rate is relatively high, SGD and ASGD, which each gradient descent depends on a very small batch size, seem to converge to the local minimum and get stuck in there while the other optimizer performs well. Since we use a pre-trained CNN model and a self-defined RNN model, the main computation is concentrated on the RNN model. The graphs clearly show the descent performance of RNN model, in which RMSProps seems to be the most effective for the sequential data, especially our simple image captioning model. Due to the particularity of our model, different choices of learning rate does not seem to bring too much difference in gradient descent performance between different optimizers.

4.5 Summary

From the graph of cross entropy loss v.s. cumulative batches, it can be observed that the training losses of all models have a similar amount of gradient descent. However, different models seem to generate completely different captions.

First of all, we notice that without attention, the model fails to generate an idiomatic caption. For example, in the skiing case, key features like “snowy” will disappear completely from the caption, instead the sequences like ”skateboarding down” will take place. In addition, it also generates some irrelevant features such as “a red jacket” in the caption, which reflects a relatively bad captioning. Moreover, we test several images at the end of the 5th training epoch and find that the caption format generated by this model is often similar to ”a man/woman in a red shirt/jacket”, which means that the model fails to capture image features well. Since the model only uses LSTM as the memory cell, its ability to interpret the image as a whole is poor.

Next, in the rest of the models with attention, we find that the global attention mechanism and the local attention mechanism enjoys similar captioning ability. The only difference is the nuance of training time.

After that, through the comparison between GRU and LSTM, we find that there is no significant difference in model performance. Since GRU has fewer parameters, GRU captures less information at the end of the same training epochs, which leads to the generation of some unrelated words like “red jacket”. However, the overall caption is semantically flowing. On top of that, as the result of fewer parameters, the models with GRU train a little faster, which we later use in our model demonstration.

Finally, by comparing the performance of different pre-trained CNN models, we find that VGG takes up a lot of computing resources and longer training time to generate captions of better quality. Since there are 64 channels in the first layer of the VGG network, and the number of channels in each subsequent layer is doubled, up to 512 channels at most[8], it has a much deeper network structure compared with ResNet. With the increase of the number of channels, more information can be extracted from the image. We also notice that in the attention visualization graph, each word comes from the exact region of the image, which proves that VGG extracts more information from the image than ResNet.

To sum up the evaluation, we conclude that when the encoder is set to VGG-19, the decoder is set to LSTM with local attention, and the optimizer is set to RMSProps, the model outputs the captions of the best quality. So we eventually choose this model for training, testing and model demonstration. This model can properly detect the fore-ground objects such as dogs and people as well the background with arbitrarily monotonous color such as grassland, ocean and pitch road.

5 Results

We use VGG-19 together with LSTM with local attention to construct our image captioning model. The final training loss is 1.53, while the final average validation loss generated by calculating the average loss of several randomly selected images is 1.89. It’s worth noticing that the validation loss has a high variance, which equals to 0.79.

We utilize the Matplotlib package to visualize the attention assigned to a given image behind each generated word. Here we can see that different amounts of attention are distributed to different regions of the image. In the sample image, we can clearly see that when the word “hiker” is being generated, most of the attention is on the hiker’s face in the image. After that, when the word “snowy” is being generated, the attention is focused on the white region of the image. This effectively visualizes how our model analyzes the images and gives the captions.

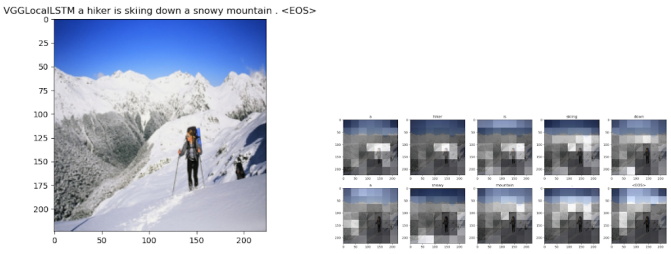


Image Attention

Figure 8. Visualized Attention

In order to let more people experience the fun of machine learning as well find out the practical application of neural network models in life, we define a user interface. Users can upload images for the model to analyze, and the model will give captions for the images.

In order to compare the models and give users more captioning results to refer to, we provide two models in the user interface: VGG+LSTM+Local attention and ResNet+GRU+Global attention.

Figure 9. User Interface

6 Conclusion and Future Work

In our project, we implemented a simplified yet flexible image captioning model using data pre-processing tools and deep learning techniques. We try on different neural networks and multiple kinds of embedding techniques for parallel comparison. Finally, we successfully visualize our results with decent generated captions.

However, due to the limitation of time and computational resources, we only train the model on the Flickr 8k dataset. Thanks to image preprocessing, we are fortunate to be able to add some noise to the dataset to prevent overfitting and increase the generalization ability of the model. But what we expect for the future is to train and test the model on larger datasets (such as Flickr 30k) with higher resolution images and more varied captions. Based on larger datasets, we will try to dig deeper into the multi-layered RNN models with more adjustable parameters to control the overall performance of the model, so that the models can generate longer and more complicated captions, even tell stories.

In the selection of loss function, we only try cross entropy loss for the image captioning model in our project, which only reflects the matching probability between each generated word and the corresponding word in the target caption. In other words, cross entropy loss is doing word-to-word comparison between prediction and target, which cannot reflect the overall rationality of a sentence. This can partly explain why the validation loss has a high variance. In contrast, BLEU metric[9] appears to address the limitations of word-to-word comparison by comparing several words with several words, or by comparing the sentences as a whole. Since BLEU uses 4-gram by default and we only have 8508 words in our dictionary, a slight difference will result in a very low score in caption evaluation, which is not suitable for the evaluation of our current models. But we expect to use this evaluation metric to train our model in the future, when we will have larger datasets and a larger dictionary.

As for the user interface, we are considering building a more flexible user interface, such as the Neural Network Playground[10] , so that everyone can feel the fun and power of image captioning and thus the field of deep learning. We also consider allowing users to provide the correct captions of the images they uploaded, so as to achieve the purpose of extensively collecting data for model training.

References

[1] adityajn105, "Flickr 8k dataset," 4 2020, online; accessed 6-May-2021. [Online]. Available: https://www.kaggle.com/adityajn105/flickr8k

[2] Hsankesara, "Flickr image dataset," 6 2018, online; accessed 6-May-2021. [Online]. Available: https://www.kaggle.com/hsankesara/flickr-image-dataset/metadata

[3] "Deep learning (5) data processing - resize," online; accessed 6-May-2021. [Online]. Available: https://programmersought.com/article/39774290802/

[4] J. Nelson, "Why and how to implement random crop data augmentation," 2 2020, online; accessed 6-May-2021. [Online]. Available: https://blog.roboflow.com/why-and-how-to-implement-random-crop-data-augmentation/

[5] M. Tiva, "Batch normalization in convolutional neural networks," 3 2021, online; accessed 6-May-2021. [Online]. Available: https://www.baeldung.com/cs/batch-normalization-cnn

[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 5 2016.

[7] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 8 2015.

[8] @Irene, "Deep learning 10-introduction to classic deep learning model 2 + vgg + resnet + goolenet," 3 2020, online; accessed 6-May-2021. [Online]. Available: https://blog.csdn.net/qq_42871249/article/details/104836621

[9] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[10] D. Smilkov and S. Carter, "A neural network playground," online; accessed 6-May-2021. [Online]. Available: https://playground.tensorflow.org/#activation=relu&regularization=L2&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0.003&noise=10&networkShape=4,3,5,2&seed=0.29213&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=true&xSquared=false&ySquared=false&cosX=false&sinX=true&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

[11] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 67–78, 02 2014. [Online]. Available: https://doi.org/10.1162/tacl_a_00166

[12] spro, "Practical pytorch: Translation with a sequence to sequence network and attention," 7 2018, online; accessed 6-May-2021. [Online]. Available: https://github.com/spro/practical-pytorch/blob/c520c52e68e945d88fff563dba1c028b6ec0197b/seq2seq-translation/seq2seq-translation.ipynb

[13] S. Sarkar, "Image captioning using attention mechanism," 3 2020, online; accessed 6-May-2021. [Online]. Available: https://medium.com/swlh/image-captioning-using-attention-mechanism-f3d7fc96eb0e

[14] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," 2 2014.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2013, pp. 746–751.

[18] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth, "Every picture tells a story: Generating sentences from images," in *European conference on computer vision*. Springer, 2010, pp. 15–29.

[19] J. Lu, C. Xiong, D. Parikh, and R. Socher, "Knowing when to look: Adaptive attention via a visual sentinel for image captioning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 375–383.

Appendix I: Parameter List for RNN Comparison

Training_Batch_Size	156
Testing_Batch_Size	1024
Device	torch.device(“cuda” if torch.cuda.is_available() else “cpu”)
Encoder_Dim	2048
Decoder_Dim	512
Attention_Dim	200
Learning_Rate	0.001
Embed_Dim	200
VOCAV_Dim	len(dataset.word_dictionary)
PAD_IDX	dataset.word_dictionary.tokens_to_index[“<PAD>”]

Training_Batch_Size: Control the amount of data delivered to train the model each time.
Testing_Batch_Size: Control the amount of data delivered to validate the model each time.
Device: Perform the training on CPU/GPU.
Encoder_Dim: The feature vector of each partition of a given image in CNN model.
Decoder_Dim: The dimension of the hidden state in RNN model.
Attention _Dim: The amount of attention allocated to each feature vector, serving to save some computational time without losing too much feature information of the images.
Learning Rate: Control the convergence speed of gradient descent.
Embed_Dim: Control the dimension of the embedding of a given token.
VOCAB_Dim: The size of the token dictionary.
PAD_IDX: The token that does not need to be updated during gradient descent.

Appendix II: Parameter List for CNN Comparison

Training_Batch_Size	156
Testing_Batch_Size	1024
Device	torch.device(“cuda” if torch.cuda.is_available() else “cpu”)
Encoder_Dim	512
Decoder_Dim	400
Attention_Dim	128
Learning_Rate	0.001
Embed_Dim	200

Training_Batch_Size: Control the amount of data delivered to train the model each time.
Testing_Batch_Size: Control the amount of data delivered to validate the model each time.
Device: Perform the training on CPU/GPU.
Encoder_Dim: The feature vector of each partition of a given image in CNN model.
Decoder_Dim: The dimension of the hidden state in RNN model.
Attention _Dim: The amount of attention allocated to each feature vector, serving to save some computational time without losing too much feature information of the images.
Learning Rate: Control the convergence speed of gradient descent.
Embed_Dim: Control the dimension of the embedding of a given token.

Appendix III: Parameter List for Optimizer Comparison

Training_Batch_Size	156
Testing_Batch_Size	1024
Device	torch.device(“cuda” if torch.cuda.is_available() else “cpu”)
Encoder_Dim	2048
Decoder_Dim	448
Attention_Dim	156
Learning_Rate	3e-4
Embed_Dim	200
VOCAV_Dim	len(dataset.word_dictionary)
PAD_IDX	dataset.word_dictionary.tokens_to_index[“<PAD>”]
pre_transform	Compose([Resize((224,224)) , RandomCrop(224) , ToTensor() , Normalize((0.485,0.456,0.406) , (0.229,0.224,0.225))])
dataset	Flickr8K(root_path=DATA_PATH + “/Images”,captions_path=DATA_PATH+“/captions.txt” , transform=pre_transform,train=True)

Training_Batch_Size: Control the amount of data delivered to train the model each time.
Testing_Batch_Size: Control the amount of data delivered to validate the model each time.
Device: Perform the training on CPU/GPU.
Encoder_Dim: The feature vector of each partition of a given image in CNN model.
Decoder_Dim: The dimension of the hidden state in RNN model.
Attention _Dim: The amount of attention allocated to each feature vector, serving to save some computational time without losing too much feature information of the images.
Learning Rate: Control the convergence speed of gradient descent.
Embed_Dim: Control the dimension of the embedding of a given token.
VOCAB_Dim: The size of the token dictionary.
PAD_IDX: The token that does not need to be updated during gradient descent.