Show and Tell: A Neural Image Caption Generator

Final Project Report of IE534/CS598 Deep Learning

Hanwen Hu, Chunlei Liu, Renjie Wei, Xinyan Yang

December 11, 2018

1 Introduction

The Show-and-Tell paper proposed in 2015[1] makes a progress on automatically describing the content of an image. In this paper, they present a generative model based on a deep recurrent aneural network that combines a recent advance in computer vision and machine translation. The model is trained to maximize the likelihood of the target description sentence given the training image. They apply the model on different datasets and evaluated with different metrics like BLEU, METEOR and CIDER. The results show that there is a significant improvement.

We aim to generate more accurate captions for the images. In the learning process, the models can generate better captions which accord with the content of image. At last, we achieved a BLEU-4 of score, which improved by how much compared with previous one.

The MSCOCO 2014 dataset has around 83K training images and 41K validating images, with a bunch of annotations per image. We choose this dataset to ensure the robustness of models for this implementation.

2 Model Overview

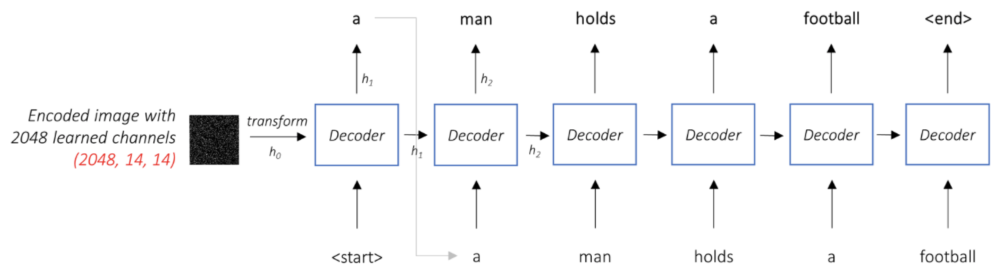


Figure 1: Encoded image and generate a caption word by word.

In this implementation, we adopt a typical model architecture of image caption as the Figure 1 shows. The Encoder maps images to embedded vectors, so its usually a Convolutional Neural Network (CNN). While the Decoder needs to generate sequences, hence would need to be a Recurrent Neural Network (RNN).

Suppose the input is one single image, the output of Encoder is a tensor with size (2048,14,14), and the input dimension of Decoder is 2048. Transform the image tensor into a 2048-dim vector by simply averaging all pixels. In addition, all words in the vocabulary are also transformed into 2048-dim vectors through a embedding layer. In this way, first input the image vector as the initial input of Decoder. Then in each timestep, the Decoder will compute for a probability distribution on the whole vocabulary, by which the next predicted word will be produced. Cross Entropy Loss is adopted as the loss function.

After a good Encoder and Decoder have been trained, we apply the Beam Search approach to generate captions for images. Suppose we choose the beam size as 3. In time step t, the 3 best sentences (i.e. with the largest posterior likelihood) will be chosen as the candidates to generate sentences in time step t+1. If any best sentence reaches its end, the beam size will be reduced by 1. This process will iterate until beam size becomes 0. Apparently, as the beam size increases, it is more possible that we may generate better captions for images.

3 Implementation

3.1 Data Preprocessing

The first thing we do is substitute words by indices, or encoding. We utilize train2014.json and val2014.json files to gather all captions and collect the vocabulary. Then we filter all words with a frequency under the specified minimum frequency (default as 5). The returned file is called a wordmap, i.e. a dictionary that maps the remaining words to the indices. Then to load the data, a custom CaptionDataset is built to read data with COCO API, and pack images with their corresponding captions.

For convenience, we do the data preprocessing in the following way. First, we unify the number of captions per image (default as 5) to avoid imbalance. Then, add '<start>' to the heads of all captions, '<end>' to their tails. Substitute the words that are not in the vocabulary with '<unk>'. We also unify the length of captions (default as 50). Then, all captions are encoded by the wordmap created above. For images with only 2 channels (i.e. grey scale images), transform they into RGB images. All images are resized to 224*224, and are performed with specified transformation, such as normalization.

3.2 Encoder

The Encoder module is used to encode an input into a fixed form. The input of Encoder is image with 3 color channels into a smaller image with "learned" channels. To avoid training Encoder from scratch, we use transfer learning models, including the pretrained ResNet-50, ResNet-101 and ResNet-152 on ImageNet. In order to improve the model performance, we fine-tune the convolutional block 4 of the encoder in the full model. These models generate smaller images to represent the original ones, and the smaller images are more “learned” than previous ones with greater channels.

3.3 Decoder

The Decoder module is used to generate a sequence of words distributions by the output of Encoder. In this way, we can either do greedy search or beam search to obtain a full predicted sentence. We choose LSTM as the Decoder Architecture. After receiving the output, we flatten it to dimensions *[batch_size, 2048]* by averaging over all pixels. At each time-step, the hidden and cell state will be updated, and produce a probability distribution over the vocabulary, from which we can compute the prediction scores, or posterior likelihood, and generate the following word.

3.4 Training

In the training process, we apply ADAM as the optimizer for both Encoder and Decoder. An option is provided to whether the Encoder needs to be tuned. In each epoch, we store important information as a checkpoint, including current training and validation loss, top-5 accuracy, BLEU-4, Encoder and Decoder models, as well as whether this epoch produces a better BLEU-4 than the one before. Therefore, we can choose to begin the training process by reading a checkpoint rather than training from scratch.

3.5 Evaluation

During inference, we apply beam search to all images in the validation set. Then we compute the BLEU-4 scores based on the generated captions. Users can also generated their own captions by running our python file, “captions.py, in their terminal. After one image and the model to use are inputted, it only takes several seconds to generate a caption of the image by that model. Users can also appoint different beam sizes for the search of captions.

4 Experiments and Results

4.1 Experiments

We design two sets of experiments. For both sets, we apply batch size as 32, words embedding dimension as 2048, dimension of hidden states in Decoder as 512, both Encoder and Decoders learning rates as 1e-4. In the first set, we compare three different Encoder models, ResNet-50, ResNet-101 and ResNet-152. Only the last convolution block of them are tuned, and dropout rate is fixed at 0.5. As for the second set, we fix the Encoder model as ResNet-101 and compare performances of models trained with dropout rates as 0.3, 0.5 and 0.8.

4.2 Results

The total computation hours are about 600 hours.

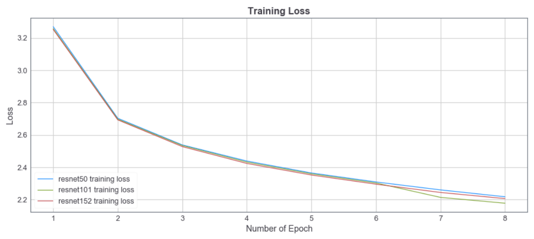
To evaluate the models performance, we used the BiLingual Evaluation Understudy (BLEU) -

4 metric to evaluate the generated captions, which is a metric used in machine translation to evaluate the quality of generated sentences. The following table 1 are the bleu - 4 scores for different models with same dropout rate and same model with different dropout rate.

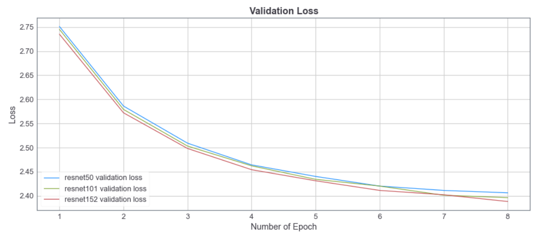
Model Parameters	BLEU-4 Score	Model Parameters	BLEU-4 Score
Resnet 50, dropout 0.5	0.2167	Resnet 101, dropout 0.3	0.2175
Resnet 101, dropout 0.5	0.2178	Resnet 101, dropout 0.5	0.2178
Resnet 152, dropout 0.5	0.2178	Resnet 101, dropout 0.8	0.2179

Table 1: BLEU-4 Scores of Different Models(beam size = 5).

For three different pretrained models, training losses are almost the same. In addition, the validation losses have small differences. In particular, the loss of ResNet152 is lower than that of ResNet101, while ResNet50 is the highest. This matches intuition that a more complicated model gives a better result. The loss results can be seen in Figure 2.



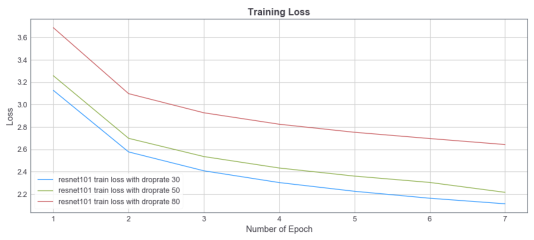
(a) Train Loss



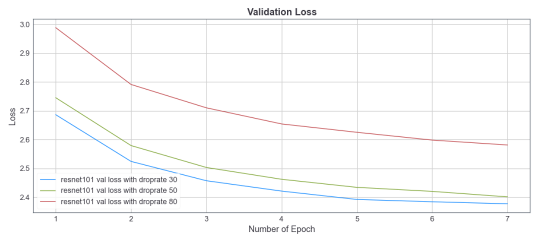
(b) Validation Loss

Figure 2: Training and validation loss of different models with same dropout rate.

For the resnet101 with different dropout rates, the training losses are very different. The loss of model with dropout rate 0.3 is lower than that with dropout rate 0.5, and that dropout rate 0.8 works worst. The validation losses follow the similar pattern. The results can be seen in Figure 3.



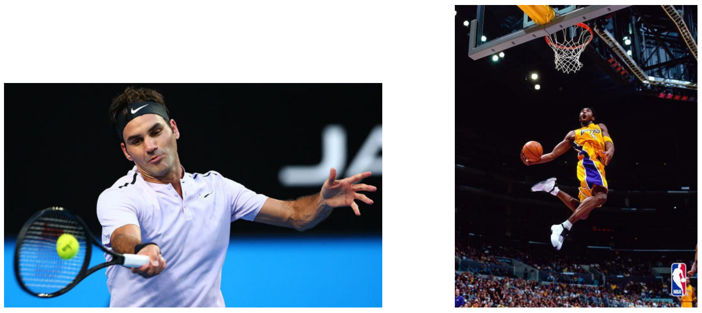
(a) Train Loss



(b) Validation Loss

Figure 3: Training and validation loss of same models with different dropout rate.

We also generate the captions of several images with different models we have trained and summarize them in the following Table 2.



(a) Federer (b) Kobe Bryant

Figure 4: Two Example Images

Model Parameters	Captions of Figure 3(b)
Resnet 50, dropout 0.5	[a man on a court with a tennis racket]
Resnet 101, dropout 0.5	[a man jumping up to catch a frisbee]
Resnet 152, dropout 0.5	[a man doing a trick on a skateboard]
Resnet 101, dropout 0.3	[a man jumping in the air with a basketball]
Resnet 101, dropout 0.8	[a man is doing a trick on a skateboard]

Table 2: Captions Generated with Different Models with Beam size = 5 for Figure 4.

For the first image, most models succeeds in capturing most information, they all produced the caption of 'a man holding a tennis racquet on a tennis court'. For the second image, we observe that only Resnet 101 with dropout rate 0.3 is the only model that works well. This implies its good performance on generalization followed by a proper dropout rate.

5 Conclusions

The BLEU-4 score in the paper is 27.7. Our BLEU-4 scores between different models are very similar. They are all about 21.8. Our model behaves worse than the original paper. Potential reasons for this are the training time is not long enough, we didnt tune enough encoder layers because of the memory limitation and the pretrained models used are different.

To improve the model in the future, we can improve the architecture of the model. We can find better hyperparameters. We can find larger datasets.

References

[1] Vinyals, Oriol et al., *Show and tell: A neural image caption generator*, (2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 3156-3164).

[2] Sagar Vinodababu, *a PyTorch Tutorial to Image Captioning*, available at `https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning`.