IMAGE CAPTIONING USING COMPOSITIONAL SENTIMENTS

BY

YI YANG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Professor Mark Allan Hasegawa-Johnson

Abstract

This thesis presents a method to generate emotional captions of images. An adequate caption should precisely describe the contents in an image. While humans can readily identify the most emotionally salient aspects of an image, many captioning models have difficulties in detecting and generating these non-factual aspects. This is caused by lack of sentiment information in the caption dataset. We solve this issue by preprocessing the text captions in an image captioning dataset with a sentiment analyzer to determine sentiment scores of all images in the training dataset. The model trained from this dataset is able to generate captions that communicate sentiment effectively, without requiring human judges to label sentiment of the training images. The model learns contents of training images, along with embedded word and sentence sentiments. Compared with the model without sentiment, it has better text captioning performance on BLEU-2, which improved from 17.15 to 18.25, and on CIDEr, which improved from 45.21 to 45.68. Automatic sentiment classification of generated captions matches the target sentiment as specified to the captioning system, with accuracy reaching 77.30%, 66.25%, 27.05 on negative, neutral and positive sentiments respectively.

Table of Contents

Chapter 1: Introduction

Image captioning is a task that requires a machine to generate descriptions based on the contents of images. An image captioning system usually contains several components. The initial components are object detection and classification. Then machine translation is applied to make the captions more similar to what a human would produce [1]. However, machine generated captions are usually more objective than human captions, so systems have been produced that introduce style or sentiment into captions [2,3,4]. For example, adjective-noun pairs have been applied to a caption generator so the output is no longer neutral [3]. This thesis demonstrates a new method for integrating sentiment into a generated image caption.

In this thesis, Chapter 2 describes previous work in this field and the performance achieved by these approaches. Chapter 3 illustrates the method we applied and the algorithm behind it. Chapter 4 discusses our experiments with the model, including the training stage and the inference stage. In Chapter 5, experimental results are presented and analyzed. Chapter 6 is the conclusion. A copy of core code is presented in the Appendix.

Chapter 2: Related work

This work is inspired by previous work in image captioning, sentiment classification, and sentiment-scored image captioning.

2.1 Image captioning

The task of an image captioning system is to generate descriptive sentences for a given set of images. Earlier practices involved object detection and language models. Since the release of the MS COCO image captioning challenge, a competition to achieve the best BLEU score has begun [5,6]. The BLEU (BiLingual Evaluation Understudy) score is a family of automatic evaluation metrics for machine translation [6]. The BLEU score ranges from 0 to 1, reflecting the number of matches between target text and reference text. A higher BLEU score indicates the target is close to the reference. BLEU is a modified precision score, augmented with features that improve its correlation with subjective evaluations of machine translation output by human judges [6]. Since human evaluation is expensive and slow, BLEU has become one of the standard metrics for the MS COCO image captioning challenge.

Most recent Image captioning systems were developed from sequence-to-sequence models [1], which are a class of neural network models initially used in machine translation. The first researchers to bring neural networks into an image captioning task were Kiros et al. [7]. They built a

log-bilinear multimodal language model that can generate sentence descriptions without templates [7]. Mao et al. started to use RNNs instead of feed-forward networks [8]. Vinyals et al. built an end-to-end neural image caption model, using convolutional networks (CNNs) to learn the features from images and using long short-term memory (LSTM) units to generate captions [4]. With the further development of CNN architectures, variants of CNN-RNN encoder-decoder models have been developed to improve BLEU score. Karpathy and Li used RCNN and bidirectional RNN [9]. Xu et al. [10] introduced an attention mechanism [11] into a neural image captioning model. This method applied attention on image vectors and mixed them with embedded word vectors to form LSTM inputs [10]. Based on automatic evaluation results in MSCOCO, the NIC v2 by Vinyals et al. has the best BLEU-4 score as of 2019, with BLEU-4= 32.1 [4]. Figure 1 shows a network flow diagram of the NIC v2 system.

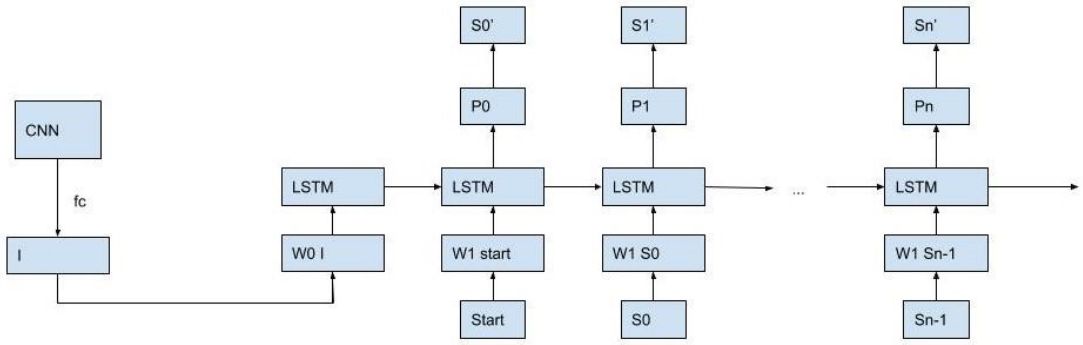

Figure 1: NIC image captioning model. CNN model generates image features, I. Image features are used to generate initial states of the LSTM cell. Succeeding LSTM cells each take an embedding of the preceding word as input. All LSTM cells share the same parameters.

The NIC v2 system encodes the input image using a CNN, then provides each succeeding LSTM cell with a dense embedding of the preceding word.

$$x_{-1} = CNN(I) \quad (1)$$

$$x_t = W_e \cdot S_t \quad (2)$$

where I is the image input, and S_t is the t^{th} sequential word input. Image input is involved only in the initialization step. At each succeeding time step, encoder reads input as one additional embedded word.

The decoder LSTM structure is as follows:

$$i_t = \sigma(W_{ix} \cdot x_t + W_{ih} \cdot h_{t-1}) \quad (3)$$
$$f_t = \sigma(W_{fx} \cdot x_t + W_{fh} \cdot h_{t-1}) \quad (4)$$
$$o_t = \sigma(W_{ox} \cdot x_t + W_{oh} \cdot h_{t-1}) \quad (5)$$
$$\tilde{c}_t = tanh(W_{cx} \cdot x_t + W_{ch} \cdot h_{t-1}) \quad (6)$$
$$c_t = f_t \bigodot c_{t-1} + i_t \bigodot \tilde{c}_t \quad (7)$$
$$h_t = o_t \bigodot c_t \quad (8)$$
$$p_{t+1} = Softmax(h_t) \quad (9)$$

where i_t is the input gate, f_t is the forget gate, o_t is the output gate, h_t is the hidden state, c_t is the cell state, and the circumpunct represents array product with a gate mass function that generates the upcoming word S_{t+1}.

2.2 Sentiment analysis

Descriptions generated by humans contain not only factual terms, but also some non-factual aspects, like feelings. Sentiment analysis is used to understand the emotion or feeling in a given text. In most cases, sentiment analysis is applied to determine the polarity of data, that is, to figure out whether sentiment expressed by a given text is negative, positive or neutral. To determine a sentiment may require a single word, a word group, a sentence or even a paragraph. The task is closely related to language modeling, lexical analysis and word vector embedding. The basic approach is to analyze sentiment in each individual word. Sub-sentence level sentiment analysis or sentence level sentiment analysis requires composition of word sentiments.

The widely used “bag of words” approach adopts a simple model of sentiment composition, using word frequencies as features, and training a classifier to estimate sentiment from the observed word frequencies [12]. It works well on long documents that have one single strong sentiment. For some cases, however, only counting positive or negative sentiments will not work. Different syntactic structures are developed to solve this issue. Polanyi and Zaenen introduced contextual valence shifters, in which sentiments scores are modified by lexical items, including intensifiers, negatives, modal operators, etc. [13]. Another approach is a recursive neural network [14], in which each leaf node of a parse tree represents a word vector. A composition function is applied to two child nodes in order to compute the parent node. A classifier is attached to each parent node and gives label probabilities. Later, the matrix-vector RNN was developed. It used both a matrix and a vector to represent every word [15]. The matrix-vector RNN had too many parameters to be easily trained, so Socher et al. introduced the RNTN, or recursive neural tensor network [16]. With the same structure as a parse tree, it applied a tensor-based composition function instead of matrix multiplication. According to results published in 2013 [16], the RNTN model by Socher et al. has the best performance on classifying positive and negative sentences.

2.3 Image captioning with sentiment

Automatic image captioning methods have been improved in recent years. According to the result of the 2015 MS COCO image challenge, the Google NIC model had the best performance [4]. It has reached 0.309 in BLEU-4, outperforming human scores, which only achieved BLEU-4= 0.217 [4,6]. However, human captions had much better performance in evaluations made by human judges [4]. Only 23.7 percent of results produced by Google NIC are considered better than human captions. Unlike automatically generated text, human descriptions are more likely to contain emotional aspects. Thus, it seems plausible that adding sentiment information may improve human acceptance of the captions generated by an image captioning system.

There are a few models that integrated sentiments or other non-factual aspects into captioning models. Most of them follow the CNN-RNN structure. Shin et al. use an LSTM and add emotional sentiments into the generator [2]. Gan et al. used a factored LSTM as the decoder and trained three weight matrices under three different styles of training captions [17]. Mathews et al. created the SentiCap model, which applied a switching mechanism to add word level sentiments [3]. You et al. propose two methods to add sentiments into captions: the direct inject method stacks sentiment with LSTM input, while the Senti flow method uses an additional LSTM hidden state to store sentiment [18]. Nezami et al. introduced a model they called Senti-Attend [19]. Following

the NIC model, the Senti-Attend model applied a soft attention mechanism to join image features and embedded word vectors [10]. Then it applied another attention block to combine word vectors and embedded sentiments.

Since there is no automatic method to evaluate sentiment effectiveness, human judges are used to make comments about performance. Based on human metrics and BLEU score, the Senti-Attend model by Nezami et al. has the best performance in 2018 (13.7 on BLEU-4)[19].

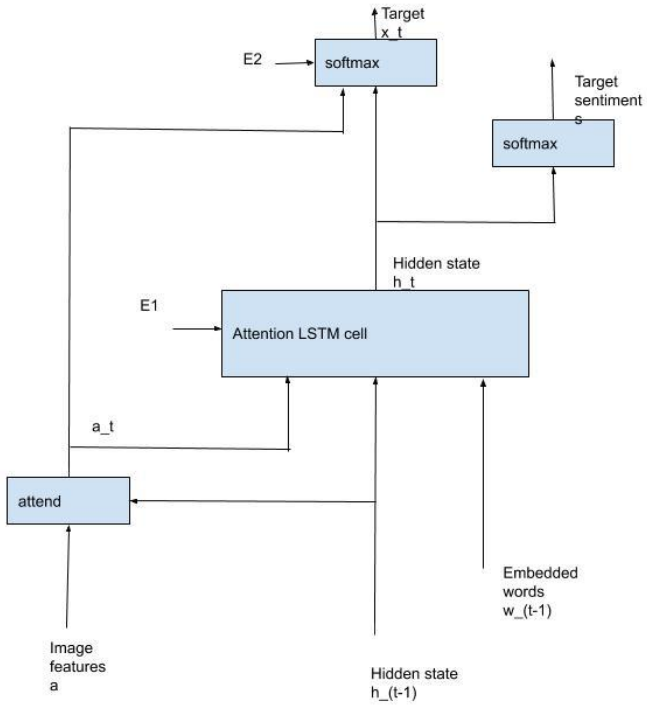

Figure 2: Flow diagram for the Senti-Attend model.

Figure 2 shows a flow diagram for the Senti-Attend model for one single step. E_1 and E_2 are embedded sentiment vectors. The attention LSTM cell takes image features, an embedded word vector, an embedded sentiment vector and the preceding hidden states as input, and generates current hidden state as output. X_t is the target word. The encoder computes an attention-weighted combination, a_t, of the image features a, and a dense embedding w_{t-1} of the previous output word x_{t-1}, where

$$x = \{x_1, x_2, \cdots x_T\} \tag{10}$$
$$a = \{a_1, a_2, \cdots a_K\} \tag{11}$$

The LSTM decoder observes the input vectors a_t and w_{t-1}, and the previous LSTM hidden state vector h_{t-1}, and computes the current hidden state vector h_t .

$$i_t = \sigma(W_i \cdot x_t + H_i \cdot h_{t-1} + A_i \cdot \hat{a}_t + B_i \cdot E_1 + b_i) \tag{12}$$
$$f_t = \sigma(W_f \cdot x_t + H_f \cdot h_{t-1} + A_f \cdot \hat{a}_t + B_f \cdot E_1 + b_f)) \tag{13}$$
$$o_t = \sigma(W_o \cdot x_t + H_o \cdot h_{t-1} + A_o \cdot \hat{a}_t + B_o \cdot E_1 + b_o)) \tag{14}$$
$$g_t = tanh(W_g \cdot x_t + H_g \cdot h_{t-1} + A_g \cdot \hat{a}_t + B_g \cdot E_1 + b_g)) \tag{15}$$
$$c_t = f_t \bigodot c_{t-1} + i_t \bigodot g_t \tag{16}$$
$$h_t = o_t \tanh(c_t) \tag{17}$$
$$p_{t+1} = Softmax(h_t W_h + \hat{a}_t W_a + E_2 W_e + b) \tag{18}$$

where i_t is the input gate, f_t is the forget gate, o_t is the output gate, h_t is the state output, c_t is the cell state, the circumpunct represents array product with a gate value, W, H, A, B and b are trained weights and biases, E_1 and E_2 are the

embedded sentiment vector, and p_{t+1} is a vector representing the probability mass function for the next output word x_{t+1}.

Chapter 3: Approach

Given an image and our desired sentiment score, our goal is to generate a caption to describe the image with style related to the proposed sentiment.

Vinyals et al. designed the NIC model to generate captions based on images [4]. In this model, the RNN cell takes CNN features as initial input, and embedded word vectors as sequential inputs. Our model concatenates sentiment input with embedded CNN features as RNN initial input.

Among approaches of image captioning with additional sentiments task, a common problem is that there is no appropriate dataset. MSCOCO [5] provides more than 400k captions with 80k images, but it does not have sentiment scores. To solve this problem, Mathews et al. use Amazon Mechanical Turk to generate a small dataset called SentiCap [3]. Another approach is to collect a weak captioning dataset, and train using the weak captioning dataset together with MSCOCO. Our approach is to use a sentiment analyzer to generate sentiment scores for MSCOCO. In this way, evaluation can also be performed automatically and there is no need for human evaluation.

Given image I, sentiment E and target sequence S, we have

$$\tilde{\theta} = argmax\Sigma \log p(S|I, E; \theta) \quad (19)$$
$$= argmax\Sigma \log p(S_t|I, E, S_0, \cdots, S_{t-1}; \theta) \quad (20)$$

And the loss function is:

$$L(I, S, E) = -\Sigma_{t=1}^{N} S_t log p_t \quad (21)$$

We fit this into an encoder-decoder model. For the encoder, we use CNN models:

$$x_{-1} = [CNN(I) \quad E] \quad (22)$$
$$x_t = W_e \cdot S_t \quad (23)$$

The CNN features, *I*, are concatenated with sentiment *E*, and served as initial input. The input at each succeeding time step is the embedded previous word S_t.

For the decoder, an LSTM is applied

$$i_t = \sigma(W_{ix} \cdot x_t + W_{ih} \cdot h_{t-1}) \quad (24)$$
$$f_t = \sigma(W_{fx} \cdot x_t + W_{fh} \cdot h_{t-1}) \quad (25)$$
$$o_t = \sigma(W_{ox} \cdot x_t + W_{oh} \cdot h_{t-1}) \quad (26)$$
$$\tilde{c}_t = tanh(W_{cx} \cdot x_t + W_{ch} \cdot h_{t-1}) \quad (27)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (28)$$
$$h_t = o_t \odot c_t \quad (29)$$
$$p_{t+1} = Softmax(h_t) \quad (30)$$

where i_t is the input gate, f_t is the forget gate, o_t is the output gate, *ht* is the hidden state, c_t is the cell state, the circumpunct represents array product with

a gate value, W are weight matrices and p_{t+1} is the probability mass function from which the word S_{t+1} is chosen.

To train this model, a sentiment analyzer is applied to generate a sentiment score for each caption in the MSCOCO dataset [5,6]. Since the MSCOCO captions are mostly neutral, they are augmented with non-neutral data from the SentiCap dataset, processed with natural language evaluation tools and relabeled. The output of the sentiment analyzer is an integer sentiment score in the range $\{1,2,3,4,5\}$, corresponding to sentiment labels of *{very negative, negative, neutral, positive and very positive}*. A statistical analysis of the corpus shows that the total number of very negative and very positive scores is less than 1%. It is not necessary to maintain 5 classes, so these sentiment labels are combined into the negative and positive labels, resulting in a new reduced sentiment label set with three possible labels: $\{-1,0,1\}$.

Inception V3 with pretrained image-net weights is applied to generate image features [13]. The feature vectors are taken from the last pooling layer with dimension (1,2048). An additional fully connected layer embeds this feature vector into 511 dimensions. The sentence sentiment is concatenated with the image feature vector to make a combined feature vector of dimension (1,512). This vector is the initial input of the LSTM cell.

As the input of succeeding LSTM cells, each word in the caption is transformed to a one-hot vector, then to a randomly initialized vector

embedding of length 512. A fully connected softmax layer maps LSTM output vectors into probability vectors of size equal to the length of the one-hot vector. The entire network is trained in order to minimize cross entropy between the original word and the predicted word probability vector.

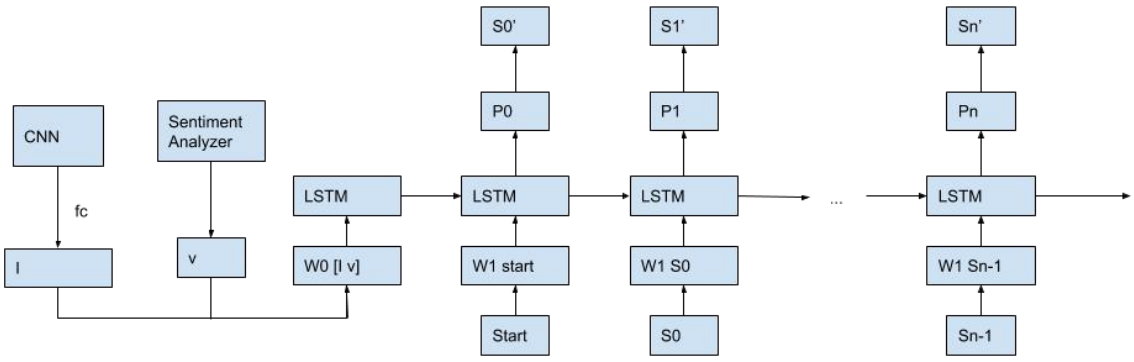

Figure 3: Image captioning model.

As shown in figure 3, the scalar integer-valued sentiment label, *y*, is concatenated to the image feature vector *I*. The CNN block uses inception v3 average pooling layer as its output. Image features are concatenated with sentiment, and fed as input to the first LSTM cell. Embedded words are then fed back as input into succeeding LSTM cells. The output layer is transformed

using a softmax layer into a probability vector. Argmax of output is the index of the generated word.

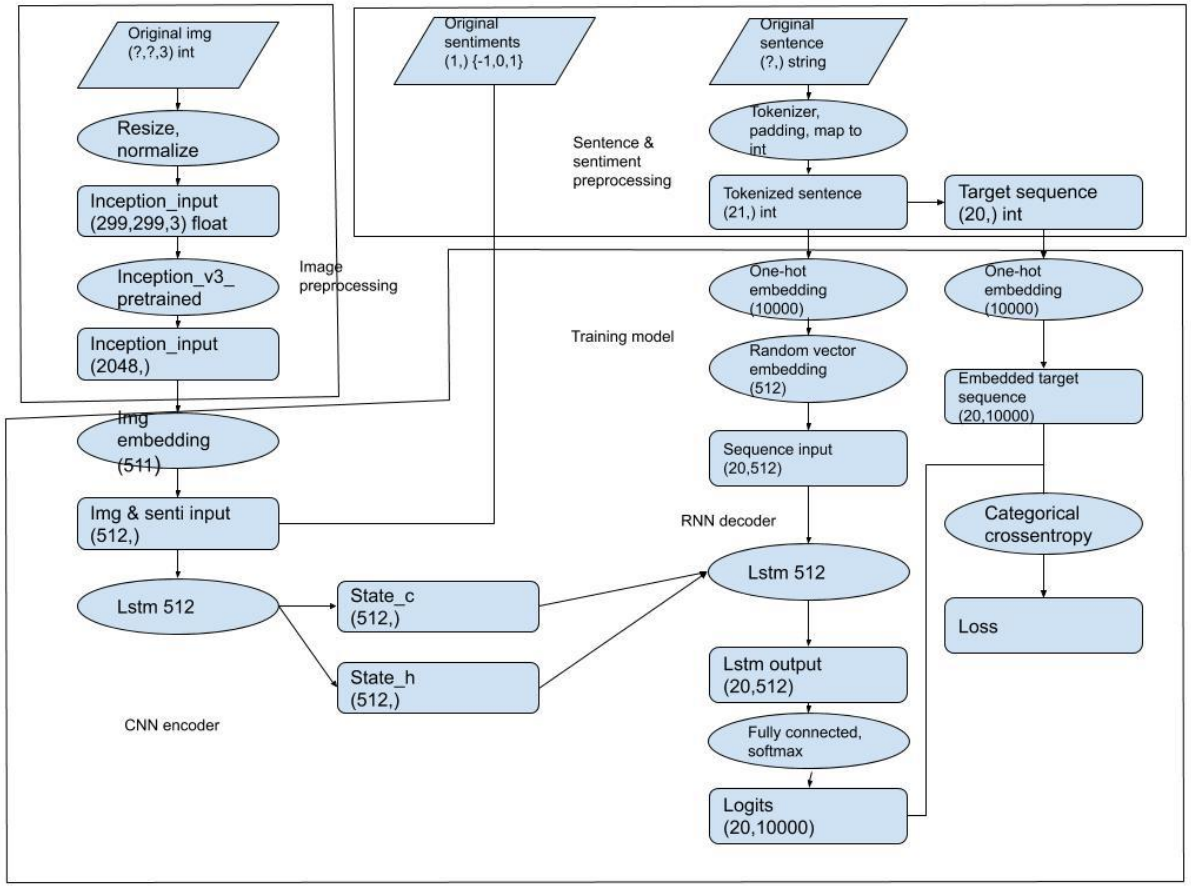

Figure 4: Flow diagram for training.

Figure 4 shows a flow diagram of the training model. Code for the training model is provided in the Appendix. The CNN-RNN model (Figure 3) is in the method RNN_train.call.

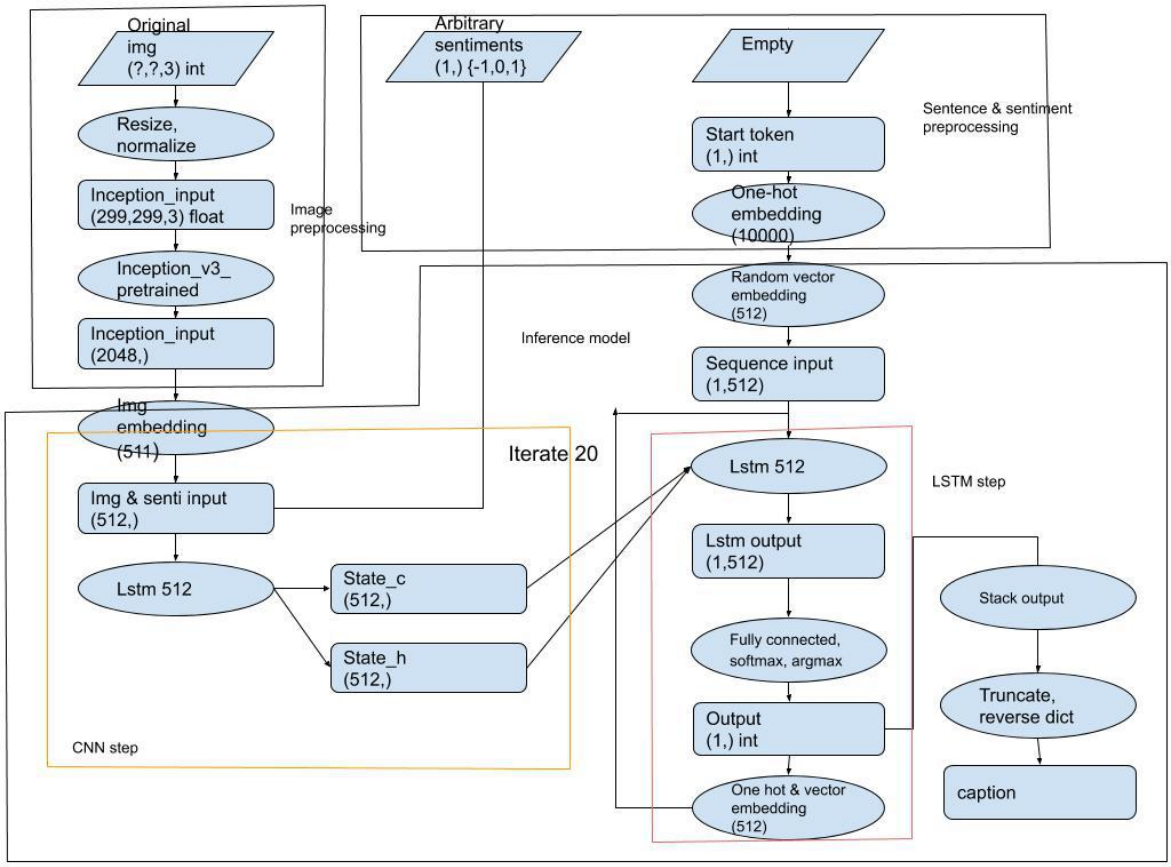

Figure 5: Flow diagram for evaluation.

Figure 5 shows the flow diagram of the inference model. Comparing with the training diagram, the input has changed. The sentiment input is defined arbitrarily, rather than using the original sentence sentiment. The sequence input is a fixed start token. The LSTM cell output is transformed using a softmax function in order to generate a vector of probabilities. The predicted word is embedded, using a trained dense embedding matrix, in order to serve as the next cell input. Code for the reference model is provided in the Appendix, in the methods cnn_step and lstm_step. These two functions,

respectively, implement the areas labeled ”CNN_step” and “LSTM_step” in Figure 5.

Chapter 4: Experimental methods

In this chapter, we present the methods that are applied in the experiment, including dataset, metrics and implementation details.

4.1 Dataset

The experiment uses two datasets: MSCOCO and SentiCap.

- MS COCO. This dataset includes 82783 images and 414113 captions in its training portion, and 40504 images and 202654 captions in the validation portion [5]. After preprocessing with a sentiment analyzer and removing some corrupt cases, the training portion contains over 80k images and 387294 captions, and the same number of sentiment scores. The training portion is applied to train our model, and the validation portion is applied to evaluate results.

- SentiCap dataset. This dataset takes 8861 images in total from the MSCOCO dataset [5]. AMT workers generate additional captions for those images with arbitrarily pre-specified sentiment (positive or negative) [3]. Although SentiCap provides a sentiment score for each caption, our model is trained using a sentiment analyzer to generate a new sentiment score for each caption, in

order to increase data coherence between SentiCap and the rest of the training corpus.

4.2 Metrics

The model is evaluated using four standard automatic metrics for machine translation quality: BLEU, CIDEr, ROUGE-L and METEOR [6,20,21,22]. BLEU (BiLingual Evaluation Understudy) measures the number of overlapping N-grams between the target sentence and reference sentence [6]. ROUGE (Recall-Orient Understudy of Gisting Evaluation) also measures N-gram overlap [21]. Unlike BLEU, which measures a modified precision, ROUGE uses recall. The other two metrics, METEOR and CIDEr, use both of them. METEOR (Metrics for Evaluation of Translation with Explicit Ordering) uses both precision and recall [22]. It uses the harmonic mean of precision and recall, plus a penalty term. CIDEr (Consensus-based Image Description Evaluation) uses cosine similarity of precision and recall instead [20]. CIDEr claims to have higher correlation with human judges than BLEU and METEOR [20].

To analyze the sentiments value of each caption, both the original and generated caption are processed using the Stanford sentiment annotator [16]. The confusion matrix of generated sentiment score and original sentiment score is reported as an experimental result.

4.3 Implementation details

Images are re-sized and set to zero mean before input. CNN parameters are kept fixed during training. The maximum word length is set to 20 characters. The tokenizer only embeds the most frequent 10000 words; other words are replaced by an out-of-vocabulary symbol (oov_token). The image feature vector is the output of inception v3's last average pooling layer with dimension (1,2048), which is then embedded to 511 dimensions and concatenated with the scalar integer sentiment score. The size of the LSTM cell is set to 512. Each word is embedded into vectors in 512 dimensions. The entire network is trained in order to minimize a cross-entropy loss function, *L*, defined as

$$L = \sum p(w_{target}) \log(p(w))$$

where $p(w_{target})=1$ for the reference word w_{target}, and *0* for all other words, $p(w)$ is an element in the softmax output of the LSTM, and the summation is over all words and over all time steps.

Chapter 5: Experimental results

Two thousand examples from the MSCOCO validation set are used for evaluation. Table 1 scores the text output of the caption generation systems with sentiment and without sentiment, using standard machine translation metrics including BLEU, ROUGE, METEOR and CIDEr. Tables 2 and 3 show confusion matrices between the intended sentiment of a generated caption (as specified at the input of the caption generator) and the measured sentiment (as classified using the Stanford sentiment parser).

Table 1. Evaluation score

	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-L	METEOR	CIDEr
With senti	32.74	18.25	11.52	7.93	31.06	13.50	45.68
Without senti	32.17	17.15	10.27	6.88	28.71	11.97	45.31

Table 1 shows BLEU, ROUGE-L, METEOR and CIDEr scores. BLEU is reported for four different N-gram lengths: *N=1*, *N=2*, *N=3*, and *N=4.* The first row is the result of configuration 1, in which an arbitrary sentiment label is concatenated with image features at the input to the first LSTM cell. The second row is the configuration without concatenated sentiment. It is clear than the model with sentiment has better performance in BLEU, CIDEr , ROUGE-L and METEOR. It has improved CIDEr by 0.34, METEOR by 1.53, ROUGE-L by 2.35, BLEU-2 by 1.1, and BLEU-4 by 1.05.

Table 2. Confusion matrix of sentiments of model with sentiment input

Truth\Prediction	-1	0	1
-1	1546	362	92
0	428	1325	247
1	242	1217	541

Table 3. Confusion matrix of sentiments of model without sentiment input

Truth\Prediction	-1	0	1
-1	812	767	421
0	812	767	421
1	812	767	421

Tables 2 and 3 are confusion matrices for generated caption sentiment and reference caption sentiment. In each table, the row label specifies the sentiment label assigned as input to the caption generator, while the column label specifies the evaluated sentiment of the generated caption (evaluated using the Stanford sentiment analyzer) [16]. In Table 3, each row has the same statistics because the caption generator does not have access to the introduced sentiment; therefore, the generated captions have exactly the same words (and consequently the same measured sentiment) in every row. Table 2 shows the difference when sentiment is involved in training. Diagonal cells (cell (-1,-1), (0,0), (1,1)) are true positives. Negative sentiment (-1) and neutral sentiment (0) are reproduced with good accuracy, but positive sentiment (+1) is not. Cell (1,0) of Table 2 shows that captions with the input label “positive” (+1) are classified as having neutral sentiment (0) 60.85 percent of the time. This is not what we expect to see. Comparing row 2 and row 3, we can see that they have similar behavior: neutral takes the highest

percentage. This model cannot generate captions that distinguish neutral and positive sentiment.

The results are generated based on 2000 images in the MSCOCO validation dataset. The BLEU, ROUGE-L, METEOR, and CIDEr scores are slightly different between the two models. The one without sentiment has slightly lower BLEU score. This difference is the result of introduced sentiment. The confusion matrices in Tables 2 and 3 show that the sentiment alignment is communicated reasonably well (for negative versus non-negative sentiment) by the caption generator with sentiment scores as its input. Due to the limit of the MSCOCO dataset, most captions can be classified into two classes: neutral or negative sentiment. There is no significant difference between the captions generated in response to a positive-sentiment target label ($v=+1$) and those generated in response to a neutral-sentiment target label ($v=0$).

Chapter 6: Conclusion

Compared to the performance of a model without sentiment, the captioning accuracy can be improved by introducing sentiment as part of the input to the caption generator. The model with sentiment inputs produces captions with better BLEU score and improved CIDEr. Of these two, CIDEr is considered to be a more accurate measure of the caption quality, because it has higher correlation with the quality evaluations given by human judges. The observed improvements in all four machine translation measures suggest that, by using sentiment to guide the generation of image captions, the model is enabled to generate captions that better resemble the style of human image descriptions.

Comparing the confusion matrices between intended and classified sentiment, one observes that the caption generation system with sentiment inputs has acceptable performance in generating captions that distinguish negative and neutral sentences, but the result also indicates that the model cannot generate captions that distinguish neutral and positive sentiment. To improve this, we need more positive-sentiment labeled captions.

The methods in this thesis provide an automatic way to evaluate sentiment effectiveness without expensive and time-consuming human

sentiment labels, using only an automatic sentiment classifier applied to an existing corpus of captioned images.

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In NIPS, pages 3104–3112, 2014.

[2] A. Shin, Y. Ushiku, and T. Harada. Image captioning with sentiment terms via weakly-supervised sentiment dataset. BMVC, 2016.

[3] A. Mathews, L. Xie, and X. He. SentiCap: Generating image descriptions with sentiments. In Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[4] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In CVPR, pages 3156–3164. IEEE, 2015.

[5] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollar, and C. L. Zitnick. Microsoft COCO captions: Data collection and evaluation server. arXiv preprint arXiv: 1504.00325, 2015.

[6] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. BLEU: A method for automatic evaluation of machine translation. In ACL, pages 311–318. Association for Computational Linguistics, 2002.

[7] R. Kiros, R. Salahutdinov, and R. Zemel. Multimodal neural language models. In International Conference on Machine Learning, pages 595-603, 2014a.

[8] J. Mao, W. Xu, Y. Yang, J. Wang, and A. Yuille. Deep caption with multimodal recurrent neural networks (m-RNN). arXiv:1462.6632, December 2014.

[9] A. Karpathy and F. Li. Deep visual-semantic alignments for generating image descriptions. arXiv: 1412.2306, December 2014.

[10] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In: ICML. pages 2048–2057, 2015.

[11] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv: 1409.0473, 2014.

[12] B. Pang, and L. Lee. Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval, 2(1-2) pages 1–135, 2008.

[13] L. Polanyi, and A. Zaenen. Contextual valence shifters. In W. B. Croft, J. Shanahan, Y. Qu, and J. Wiebe, editors, Computing Attitude and Affect in Text: Theory and Applications, volume 20 of The Information Retrieval Series, chapter 1, 2006.

[14] C. Foller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. Proceedings of the International Conference on Neural Networks (ICNN-96), 1996.

[15] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. Semantic compositionality through recursive matrix-vector spaces. EMNLP, 2012.

[16] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1631-1642, 2013.

[17] C. Gan, Z. Gan, X. He, J. Gao, and L. Deng. StyleNet: Generating attractive visual captions with styles. CVPR, 2017.

[18] Q. You, H. Jin, and J. Luo. Image captioning at will: A versatile scheme for effectively injecting sentiments into image descriptions. arXiv: 1801.10121v1, 2018.

[19] O. Nezami, M. Dras, S. Wan, and C. Paris. SENTI-ATTEND: Image captioning using sentiment and attention. arXiv: 1811.09789v1, 2018.

[20] R. Vedantam, C. L. Zitnick, and D. Parikh. Cider: Consensus-based image description evaluation. In CVPR, pages 4566–4575. IEEE, 2015.

[21] C. Y. Lin. Rouge: A package for automatic evaluation of summaries. In Text Summarization Branches Out, 2004.

[22] M. Denkowski and A. Lavie. Meteor Universal: Language specific translation evaluation for any target language. In WMT, pages 376–380, 2014.

Appendix: Core code

```
class RNN_train(tf.keras.Model):
    def __init__(self,vocab_size, max_len, units, img_embedding_size):
        super(RNN_train, self).__init__()
        # self.image_model= Image_model()
        self.vocab_size=vocab_size
        self.max_len= max_len
        self.units=units
        # self.reg= reg
        self.image_embedding_layer= Dense(img_embedding_size, use_bias=False, name =
'emb_img')
        self.seq_embedding_layer=Embedding(vocab_size, units, mask_zero = False, name =
'emb_text')
        self.BN_layer= BatchNormalization(name='batch_normalization')
        self.expand_dim= Lambda(lambda x : K.expand_dims(x, axis=1))
        # self.h= tf.zeros()
        self.lstm_layer=  LSTM(self.units, return_sequences = True, return_state = True,
dropout=0.2, name = 'lstm')
        self.softmax_layer= Dense(vocab_size, activation='softmax')
        self.time_distribute= TimeDistributed(self.softmax_layer)
        self.argmax_layer=  Lambda(lambda x : (K.argmax(x)))

    def call(self, inputs):

        image_input = inputs[0]

        seq_input = inputs[1]
        senti_input = inputs[2]
        h = inputs[3]
        cell_state = inputs[4]

        x= Dropout(0.2)(image_input)
        x= self.image_embedding_layer(x)
        senti_input= K.cast(senti_input,dtype=tf.float32)

        x= Concatenate(axis=-1)([senti_input, x])
        x= self.BN_layer(x)
        x= self.expand_dim(x)

        x_seq= self.seq_embedding_layer(seq_input)

        x_seq= Dropout(0.2)(x_seq)

        _, init_h, init_cell_state = self.lstm_layer(x, initial_state=[h, cell_state])

        out, _, _ = self.lstm_layer(x_seq, initial_state=[init_h, init_cell_state])
        output= self.time_distribute(out)

        return output

    def inference_call(self, inputs):
        image_input = inputs[0]

        seq_input = inputs[1]
        senti_input = inputs[2]
        state_h0 = inputs[3]
```

```
        state_c0 = inputs[4]
        x= image_input
        x= self.image_embedding_layer(x)

        x_senti= K.cast(senti_input,dtype=tf.float32)
        x_senti= tf.expand_dims(x_senti, axis=-1)

        x= Concatenate(axis=-1)([x_senti, x])
        x= self.BN_layer(x)
        x= self.expand_dim(x)

        x_seq= self.seq_embedding_layer(seq_input)
        x_seq= tf.expand_dims(x_seq,axis=1)
        o, h, cell_state = self.lstm_layer(x, initial_state=[state_h0, state_c0])
        outputs = []

        for i in range(self.max_len):

            out, h, cell_state = self.lstm_layer(x_seq, initial_state=[h,cell_state])
            output = self.softmax_layer(h)
            x_seq = self.argmax_layer(output)
            outputs.append(x_seq)
            x_seq= K.expand_dims(x_seq)
            x_seq = self.seq_embedding_layer(x_seq)
        return outputs

    def lstm_step(self,seq_input,ht,ct):
        x_seq= K.expand_dims(seq_input)
        x_seq= self.seq_embedding_layer(x_seq)
        out, h, c= self.lstm_layer(x_seq, initial_state=[ht,ct])
        new_out= self.softmax_layer(h)

        return new_out,h,c

    def cnn_step(self,inputs):

        image_input = inputs[0]
        senti_input = inputs[1]
        h0= tf.zeros([1,self.units])
        c0= tf.zeros([1,self.units])

        x= self.image_embedding_layer(image_input)
        x_senti= K.cast(senti_input,dtype=tf.float32)

        x= Concatenate(axis=-1)([x_senti, x])
        x= self.BN_layer(x)
        x= self.expand_dim(x)
        o, h, cell_state = self.lstm_layer(x, initial_state=[h0, c0])

        return h, cell_state
```