CS230

Describe That GIF
A GIF Description Generator

Chuma Kabaghe
Department of Computer Science
Stanford University
chumak@stanford.edu

Abstract

With the growing popularity of animated GIFs on social media, the need for improved GIF search is evident. To further this goal, we generated GIF metadata in the form of natural language descriptions. We used a pretrained resnet-34-kinetics CNN to encode the GIFs and implemented Long Short-Term Memory (LSTM) and LSTM with attention decoder models and analyzed their performance on the TGIF dataset. Previous work on GIF captioning showed the CNN-LSTM was the best performing model and our results reflected that. Our best performing CNN-LSTM model outperformed our baseline, Rand CNN-LSTM model set in the TGIF dataset benchmarks [4] by 90.8% on CIDEr score and 8.8% on METEOR score.

1 Introduction

Animated GIFs have become increasingly popular in recent years and have been shown to be more engaging than other types of media.[1] [4] Despite their popularity, searching for and finding the right GIF is tedious and frustrating. This is due to the lack of categorization and rich metadata, which is often based on hand-tagging and annotation by humans. In an attempt to improve GIF search and make GIFs more accessible, we implemented and evaluated models that generate GIF descriptions.

Describing animated GIfs is different from the image captioning task because of motion information involved between frames. Also, movie description datasets leverage professionally annotated descriptive video service (DVS) captions from commercial movies, thus, are not ideal for the GIF description task because they contain descriptions with contextual information not available within a provided clip[4]. As a result, GIF description warrants a task of its own on a dataset created specifically for this purpose.

In this paper, we tackle this task by encoding the GIFS from the TGIF dataset using a 34 layer ResNet CNN trained on the kinetics dataset and implement LSTM and LSTM with attention decoder models.

2 Related work

There is a surprising dearth of scholarly work on animated GIFs in the computer vision community. [4] To address this, Li, Yuncheng, et al. created and released the TGIF dataset and its corresponding paper. In their paper, they establish benchmarks for the dataset, some of which we use to evaluate our models. A finetuned CNN-LSTM that used weights pretrained on ImageNet-1K class categories performed the best in their experiments and that motivated our CNN-LSTM model choice.

CNNs are a standard way of encoding images and videos, and since GIFs are similar to videos in that they are a sequence of images (albeit a looping one), our model used a CNN model to encode the GIFs. Hara, Kataoka, and Satoh examined various 3D CNN architectures on video datasets and found

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

the ResNeXt-101 CNN model trained on the Kinetics dataset performed the best. However, it outputs 2,048 dimensional features per image which was too large for the amount of computation resources we had, so our model used another one of their models, the pretrained ResNet-34 Kinetics CNN.

3 Dataset

We used the TGIF dataset that contains a years worth of GIFs from Tumblr, a microblogging and social networking website. GIFs in this dataset have been deduped, and do not contain cartoons or text. We removed all broken links and GIFS containing only a single frame resulting in a dataset consisting of 90,790 GIFs, split into 82% train, 5.5% dev, and 12.5% test based off the splits provided by Li, Yuncheng, et al [4]. The training and validation set contain one reference description and the test set contains three reference descriptions per GIF.

4 Methods

Baseline
We used the Rand CNN-LSTM benchmark for this dataset set by Li,Yuncheng, et al as our baseline. The Rand CNN-LSTM model used a CNN as an encoder and an LSTM as a decoder. GIFs were sampled at 10FPS and encoded using a CNN pretrained on ImageNet-1K class categories with weights randomly intialized and fixed throughout training. The encoder output was fed into an LSTM to produce descriptions.[4]

4.1 CNN Encoder

For encoding, we used a pretrained ResNet-34 Kinetics CNN by Hara, Kataoka, and Satoh[3] modified to take GIFs as its input. The CNN produced a 512 dimension vector per image. We split GIFs into their individual frames and then encoded every 10th frame. Due to computation and time constraints, we capped the number of frames encoded per GIF to 10, resulting in a 5120 dimension output vector. This vector became the input to the decoder models.

4.2 LSTM Decoder Model

The feature vector obtained from the encoder was fed as input to the LSTM model. During training, we modified our label descriptions and target descriptions to include <start> and <end> tokens as shown below so our model could learn the start and end of a description. Additionally, the correct word inputs were fed into the LSTM at each time-step, even if it made a mistake before.

Label: <start> a man in fancy attire is pointing at his bow tie .

Target: a man in fancy attire is pointing at his bow tie . <end>

In the evaluation phase, the encoder output was input into the LSTM decoder which was initialized with a <start> token to mark the beginning of the sentence. After which, the decoder LSTM generated a description word by word making use of the hidden state and cell state from the previous LSTM unit until an <end> token was generated to mark the end of the sentence or the maximum sequence length reached. We used temperature sampling to determine the next word in the description as it performed better than simple greedy decoding.

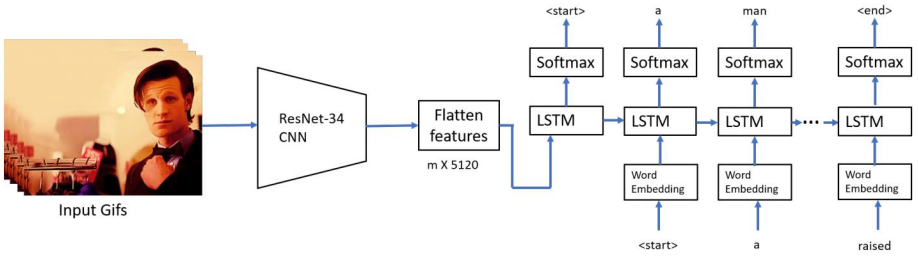

Figure 1: CNN-LSTM Model

We used the pytorch implementation of an LSTM that uses the following equations:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t c_{(t-1)} + i_t g_t \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

where h_t is the hidden state, c_t is the cell state, and x_t is the input at time t. $h_{(t-1)}$ is the hidden state of the layer at time t-1 or the initial hidden state at time 0, i_t, f_t, g_t, and o_t are the input, forget, cell, and output gates respectively. σ is the sigmoid function [2].

4.3 LSTM With Attention Decoder Model

Inspired by the success of adding attention mechanisms to machine translation models, we implemented an LSTM model with attention. To add attention, we implemented the LSTM using individual LSTM Cells and added the attention mechanism from Luong et al.[5]. At each time step t, we concatenated the input feature vector X with the hidden state h_t to produce the following attentional hidden state. Where W_c is a randomly initialized weight vector.

$$\widetilde{h_t} = tanh(W_c[X; h_t])$$

The attentional vector $\widetilde{h_t}$ was then fed through a softmax layer to obtain the alignment weight vector a_t below, where W_a is a randomly initialized weight vector.

$$a_t = softmax(W_a\widetilde{h_t})$$

The alignment weight vector was multiplied by the feature vector X to obtain an attention weighted input vector. During training, the attention weighted vector, the correct word at the current time step, and the previous hidden and cell states made up the inputs to the LSTM cell. During the evaluation phase, the inputs into the LSTM cell were the same except we no longer fed the correct label word but instead used the previous word generated by the model. We generated a description word by word, using temperature sampling to determine the best word until an <end> token or the maximum sequence length was reached.

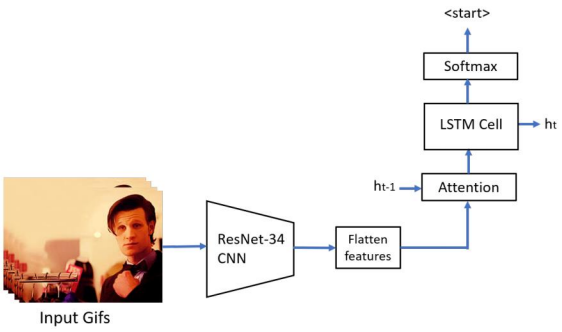

Figure 2: CNN-LSTM Cell with Attention

5 Experiments/Results/Discussion

5.1 Setting hyperparemeters

Model	batch size	hidden size	learning rate	layers	epochs	dropout	temperature
CNN-LSTM	1000	1024	0.001	1	100	0	0.8
CNN-LSTM with Attention	200	2048	0.001	1	100	0.2	0.8

Table 1: Final model hyperparameters

All CNN hyperparameters used were from CNN source code[3].

Batch size for all models was set to the maximum the GPUs could process. We experimented with one and two layers for both LSTM models, picking the best performing one. All other hyperparameters were chosen by sampling from sets of values drawn from source research papers and similar models.

The CNN-LSTM model was run on a single NVIDIA K80 GPU. We used mini-batch gradient descent with Adam optimization to minimize the cross entropy loss. After 100 epochs, training loss= 0.5464 and perplexity= 1.7270.

The CNN-LSTM model with attention was run on a single NVIDIA Tesla M60 GPU. We used mini-batch gradient descent Adam optimization to minimize the cross entropy loss. After 100 epochs, training loss= 0.6188 and perplexity= 1.8567.

5.2 Results

We report the results of our model below and compare them to the baseline Rand LSTM-CNN[4].

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE	CIDEr
Baseline (Rand LSTM-CNN)	49.7	27.2	14.5	5.2	13.6	36.6	7.6
CNN-LSTM	40.5	22.8	13.1	**7.4**	**14.8**	33.2	**14.5**
CNN-LSTM with Attention	37.4	18.0	8.7	4.5	13.3	26.6	12.3

Table 2: Model Performance Results

The CNN-LSTM model performed the best, outperforming the baseline model by 90.8% (6.9) on CIDEr score, 8.8% (1.2) on METEOR score and 42% (2.2) on BLEU-4 score. We expected the model to perform well because the best performing model in the experiments performed Li et al. was a fine tuned CNN-LSTM model[4].

We expected adding an attention mechanism to the model to improve the scores but that was not the case, which was surprising. However, the scores obtained by the model with attention were not much lower than those from the CNN-LSTM model. Thus, we believe further hyperparameter tuning and running the model for more epochs could result in equivalent or better results from the model.

6 Analysis

We qualitatively evaluate our best performing model the CNN-LSTM and compare its outputs with that of the CNN-LSTM with Attention model. In this section GT refers to the ground truth description, CL refers to the output of the CNN-LSTM model and CLwA the outputs of the CNN-LSTM with attention model.

Example Model Outputs

GT: a guy plays the guitar on stage for his friends played.
CL: a man is playing guitar and singing on stage.
CLwA: a man is playing a guitar and points to the crowd

Figure 4: Example Outputs

GT: three teenagers are dancing on a glass stage.
CL: a group of boys are dancing in rhythm.
CLwA: a group of people are dancing and holding hands.

Figure 6: Example Outputs

GT: a furry dog turns around and wags its tail.
CL: a dog is sitting on a couch and wagging his tail.
CLwA: a dog is in the <unk> on a sofa.

Figure 5: Example Outputs

GT: a man and woman are eating a noodle and kiss.
CL: a man and woman are kissing in a room.
CLwA: a man and woman are kissing each other.

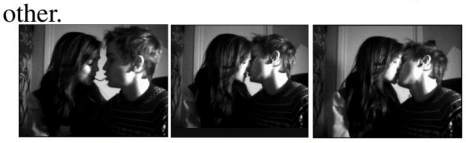

Figure 7: Example Outputs

The CNN-LSTM model produces good, complete, grammatically correct descriptions and contains relatively few <unk> tokens indicating our vocabulary is large enough for this task.

The CNN-LSTM model produces more accurate and complete descriptions on shorter GIFs than longer ones. Which could be because we cap the number of frames encoded per GIF at 10. Thus, longer GIFs have missing information while shorter ones are fully encoded and contain all the information.

The model correctly identifies objects in GIFs but sometimes wrongly classifies people's gender. Also, the descriptions produced by our model include information about larger background objects and lack information about smaller more complex details in GIFs such as glasses or fingers. Both of these failings are due to our choice of CNN. The ResNet-34 CNN we used is only 34 layers deep, a deeper CNN would be able to recognize these more complex features.

The description generated for the GIF below shows wrongly classified gender and information about the background but lacks detail about the person's fingers and glasses.

GT: a woman with glasses talked and moved her fingers.
CL: a man is talking to someone else indoors.

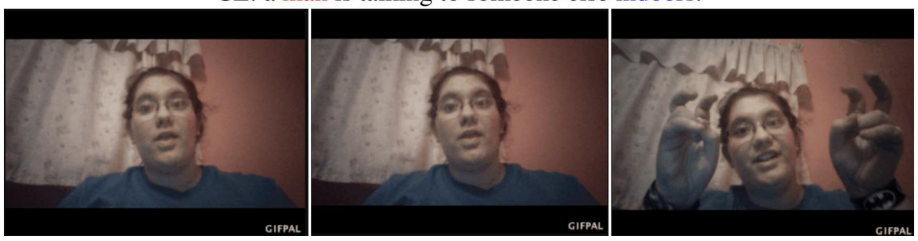

Figure 8: Erroneous GIF

7 Future Work

Based on our results and analysis we highlight some areas of future work:

- Train the CNN-LSTM model on complete GIFs without the 10 frame cap, as that will likely improve the performance of the model.
- Encode the GIFs using a deeper CNN model like the ResNext-101 trained on the Kinetics dataset that was shown to have the best performance in the experiments run by Hara, Kataoka, and Satoh [3].
- Experiment with different types of attention and visualize what the model is attending to during decoding.

8 Conclusion

Of the two models we presented, the best performing one was the CNN-LSTM which outperformed the Rand CNN-LSTM benchmark for the TGIF dataset by 90.8% on CIDEr score and 8.8% on METEOR score. We qualitatively evaluated our models and provided explanations for their shortcomings. Our experimentation was limited by the amount of time and computation resources available, thus, we also provide areas of further improvement to our models.

9 Contributions

This project began as a collaboration with Gordon Blake - `gblake@stanford.edu` in Winter 2019, however, Chuma was unable to complete it due to illness so they split the project into two separate individual projects. Chuma and Gordon performed all work until the project milestone ie. the literature search, proposal, and milestone, as well as working together to modify the pretrained CNN architecture to work on GIFs. Chuma did all the remainder of the work implementing the GIF description generation models. **Link to GitHub repository**.

References

[Bak+16] Saeideh Bakhshi et al. “Fast, Cheap, and Good: Why Animated GIFs Engage Us”. In: *CHI*. 2016.

[Con19] Torch Contributors. *LSTM*. Apr. 2019. URL: https://pytorch.org/docs/1.0.0/nn.html?highlight=lstm#torch.nn.LSTM.

[HKS17] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. “Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?” In: *arXiv preprint* arXiv:1711.09577 (2017).

[Li+16] Yuncheng Li et al. “TGIF: A New Dataset and Benchmark on Animated GIF Description”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[LPM15] Thang Luong, Hieu Pham, and Christopher D. Manning. “Effective Approaches to Attention-based Neural Machine Translation”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421. DOI: 10.18653/v1/D15-1166. URL: https://www.aclweb.org/anthology/D15-1166.