Human Consensus-Oriented Image Captioning

CS F425 Deep Learning Project

Aryan Gupta - 2019A7PS0017G
Shikha Bhat - 2019A7PS0063G
Taarush Bhatia - 2019A7PS0159G

May 20, 2022

1 Introduction

In this project, we aim to implement the image captioning model described in the Human Consensus Image Captioning paper of the Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20) written by Ziwei Wang , Zi Huang and Yadan Luo. We have experimented with different versions of the model on the Flickr 8k dataset, as described in the Contributions section.

Problem Statement Image captioning aims to describe an image with a concise, accurate, and interesting sentence. To build such an automatic neural captioner, the traditional models align the generated words with a number of human-annotated sentences to mimic human-like captions. However, the crowd-sourced annotations inevitably come with data quality issues such as grammatical errors, wrong identification of visual objects and sub-optimal sentence focuses. Existing methods treat all the annotations equally regardless of the data quality during the model training.

Approach In this project, an image captioning model is developed which explicitly identifies and tackles the annotation quality issues in image captioning, which allows the model to efficiently learn the higher quality annotations in priority. We use the human-consensus loss function for the same.

2 Dataset

We have implemented the image captioning model on the Flickr 8k dataset. It has 8,091 images that are each paired with 5 different captions which provide clear descriptions of the salient entities and events. We chose the dataset for the following reasons -

- It is a balanced dataset, because the images were chosen from six different Flickr groups, and tend not to contain any well-known people or locations, but were manually selected to depict a variety of scenes and situations.
- Initially, we had tried to use the Flickr30k dataset, which contains about 30k images each paired with 5 different captions, but due to limitations in GPU computation power and time constraints we switched to Flickr 8k which is a smaller dataset.

- The IJCAI paper used the MS COCO dataset, which is a subset of Flickr 30k. Both, MS COCO and Flickr8k datasets, are created for solving image captioning problems. It also has 5 human annotated captions for each image. Thus, our dataset is quite similar to the one used in the paper but not the same.

2.1 Preprocessing

The Flickr 8k dataset has 2 types of data: The images folder containing the 8091 RGB images in .jpg format, and the captions.txt file containing the image ID and the corresponding 5 captions.

2.1.1 Image Preprocessing

We are using the Python Imaging Library, which adds image processing capabilities. We apply the following preprocessing steps to each image:

1. **Scaling:** Since not all images are of the same resolution, we resize the images to standard 224×224 dimensions with 3 channels.
2. **ToTensor:** Converted the images to PyTorch tensor for usage in the model.
3. **Normalise:** Normalised the images to the required mean and std deviation. We selected the values of the resolution and normalisation as per the requirement of ResNet-18, which we've used for feature extraction.

2.1.2 Text Preprocessing

To make the captions usable for our model, we first performed text preprocessing for each caption:

1. Cleaning: All characters other than the space and letters are removed, and the entire text is converted to lowercase. The sentences need to be reduced to a form that can be processed by a machine more easily:
2. Tokenization and Indexing of tokens in each image description using the BERT Tokenizer in Experiment 2 and 3 of our implementation. This was removed Experiment 4 onwards and replaced by our own tokens as will be mentioned subsequently. The input ID sequences are passed through the BERT Model for creating the Word Embedding.

   ```
   tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
   ```

3. The tokens generated are also stored in a dictionary, each mapped to a unique index, which takes integer values starting from 1. These indices help mainly in mapping the output layer nodes to tokens/words; each node giving out the probability of the corresponding token/word to be the next one in the sentence.

3 Implementation

The final image caption generation model would be a combination of 2 base models - one for feature extraction of all the images, and one for finding the word embedding of the text in all captions. The output of these 2 models is then combined and passed as input to the main caption generation model.

3.1 Feature Extraction

Input Preprocessed vector of images.

Output $(512 \times 7 \times 7)$ vector of feature vectors for each of the images.

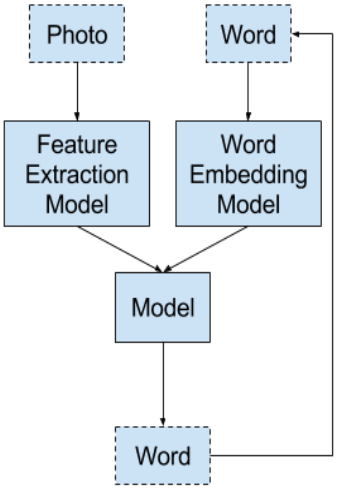

Figure 1: Image Caption Generation Model

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Figure 2: ResNet-18 architecture

Each image is converted into a feature vector by using convolutional neural network. We used the pretrained version of ResNet-18, which is a standadrd CNN model available in PyTorch. The pretrained version of the network is trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. From this, we extracted the output of layer 4 (512 * 7 * 7) to obtain the feature vector of the images.

- Resnet was selected because it was modelled and trained with specific the aim of identifying and classifying objects into a 1000 classes. This would benefit us because it is necessary to obtain features for recognizing various objects **to describe an image**.
- We used it till only layer 4, and not the complete model, otherwise the features extracted would have been way too specific to the 1000 classes that Resnet identifies, deviating from our general purpose.

```
feature_extraction = torchvision.models.resnet18(pretrained=True).to(device)
feature_extraction = nn.Sequential(*list(feature_extraction.children())[:-2]).to(device)
```

3.2 Self-Attention

The Self Attention module helps focusing on the important features of the input image.

Using the attention mechanism, we place an emphasis on the most important pixels in the image.

To focus on relevant parts on each decoding step, the attention network outputs the context vector, which is the weighted sum of Encoder's output (features).

To produce the context vector:

- First, we score each of the Encoder's outputs (features) passed to the attention network with the scoring function.
- Then we get the probabilities, applying softmax function to the scores. These values express the relevance of each feature vector that we input to the Decoder.
- Calculate the weighted sum, multiplying features by corresponding probabilities.

The current model employs a Self Attention that learns attention scores using a feed-forward network during the training.

The usage of soft attention for image captioning problem is well-described in "Show, Attend and Tell" paper under the 4.2 section and can be represented schematically as follows.

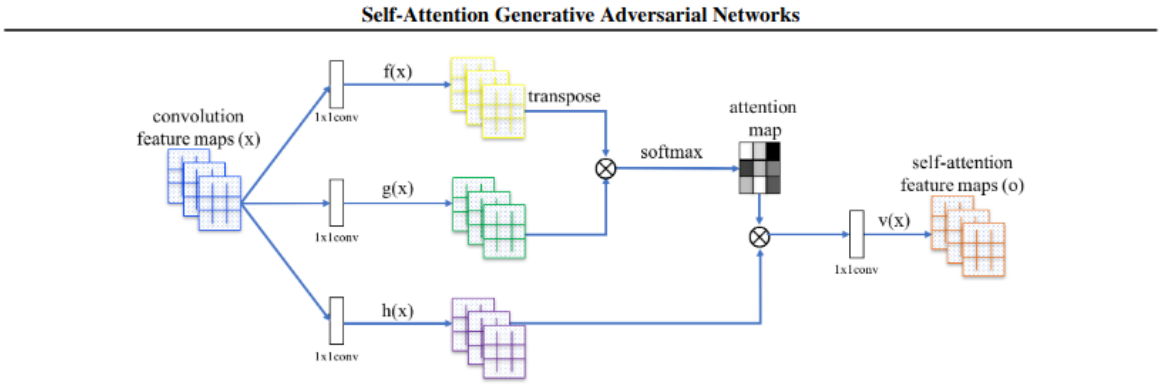

Figure 2. The proposed self-attention module for the SAGAN. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

Figure 3: Self Attention Block

The Self Attention module has been used as described here.

3.3 Word Embedding Model

The Word Embedding was generated using the following methods:

3.3.1 Pytorch Trainable Embedding Layer

[Midsem Submission] The Word Embedding layer of Pytorch was used, followed by a many-to-many LSTM layer that would help analyse the input sequence of words and give a 256-nodes layer output.

```
self.embedding = nn.Sequential(nn.Embedding(len(vocab.keys()),256), nn.Dropout(0.2))
self.lstm = nn.LSTM(256,256)
```

The Midsem submission was limited to input and output lists of fixed-lengths of 3 each. We implemented a similar model that supports variable length of lists of inputs and outputs only to observe no significance improvement in performance.

3.3.2 BERT

The BERT model is used to understand the context of words as per all the surroundings of the word. A sequence of tokens in the form of integer IDs is passed to the BERT model to obtain the Word Embedding.

- Each sequence, input with batch size of 1, starts with a "[CLS]" token and ends with "[SEP]" token. The sequence is of max length of 35, including the two previously mentioned tokens. The sentences longer than this are truncated from the end, and the ones shorter are padded with "[PAD]" token.
- The output of the BERT model is a tuple as explained here. The one that is of use to us is the first element of the tuple of dimensions: [batchSize × sequenceLength × 768], here being equal to [1 × 35 × 768]. We have used the last 768-length 1D array as the embedding.

The pretrained base-uncased BERT model.

```
bert_WE = BertModel.from_pretrained('bert-base-uncased').to(device)
```

3.4 Caption Generation Model

Input X1, X2

X1 is the feature vector we got from the ResNet-18 model for the images. X2 is the n-gram word-gram where n is the sequence length.

Output A Tensor of logits corresponding to each word in the dictionary.

The module enables the output of the Resnet18 Layer4 to the Attention block, and the input prefix caption to the BERT Model to generate a word embedding. The outputs are concatenated and passed to the LSTM layer to generate the logits corresponding to each word in the dictionary.

Greedily, the word with the maximum probability is chosen as the next word during evaluation.

```
class Caption_Generation(nn.Module):

    def __init__(self):
        super().__init__()

        self.attention = SelfAttention()
        self.flatten = nn.Flatten()
        self.img_proc = nn.Sequential(nn.Linear(7*7*512,4096), nn.Dropout(0.2), nn.ReLU(),
                                      nn.Linear(4096,1024), nn.Dropout(0.2), nn.ReLU()
                                     )

        self.embed = BERT_WE()
        self.lstm = nn.LSTM(1792,1024)

        self.cap_proc = nn.Sequential(
                                      nn.Linear(1024,512), nn.Dropout(0.2), nn.ReLU(),
                                      nn.Linear(512,256), nn.Dropout(0.2), nn.ReLU(),
                                      nn.Linear(256,index), nn.Dropout(0.2), nn.ReLU()
                                     )

        self.cap_gen = nn.Sequential(
            nn.Linear(256,128), nn.ReLU(),
            nn.Linear(128,index), nn.ReLU()
        )

    def forward(self, feat, desc):

        context, att_wts = self.attention(feat)
        ip1 = self.flatten(context)
        ip1 = self.img_proc(torch.tensor(ip1))

        att_mask = torch.IntTensor([[1]*len(desc) + [0]*(max_len-len(desc))]).to(device)
        pad_desc = (desc + [0]*(max_len-len(desc)))
        t_pad_desc = torch.IntTensor([pad_desc]).to(device)
        bert_WE.eval()
        ip2 = (bert_WE(t_pad_desc, attention_mask = att_mask, output_all_encoded_layers = False))[0][
        ip2 = torch.tensor(torch.unsqueeze(ip2,0)).to(device)

        ip = torch.cat([ip1, ip2], 1)
        ip = self.lstm(ip)[0]
        out = self.cap_proc(ip)

        return out
```

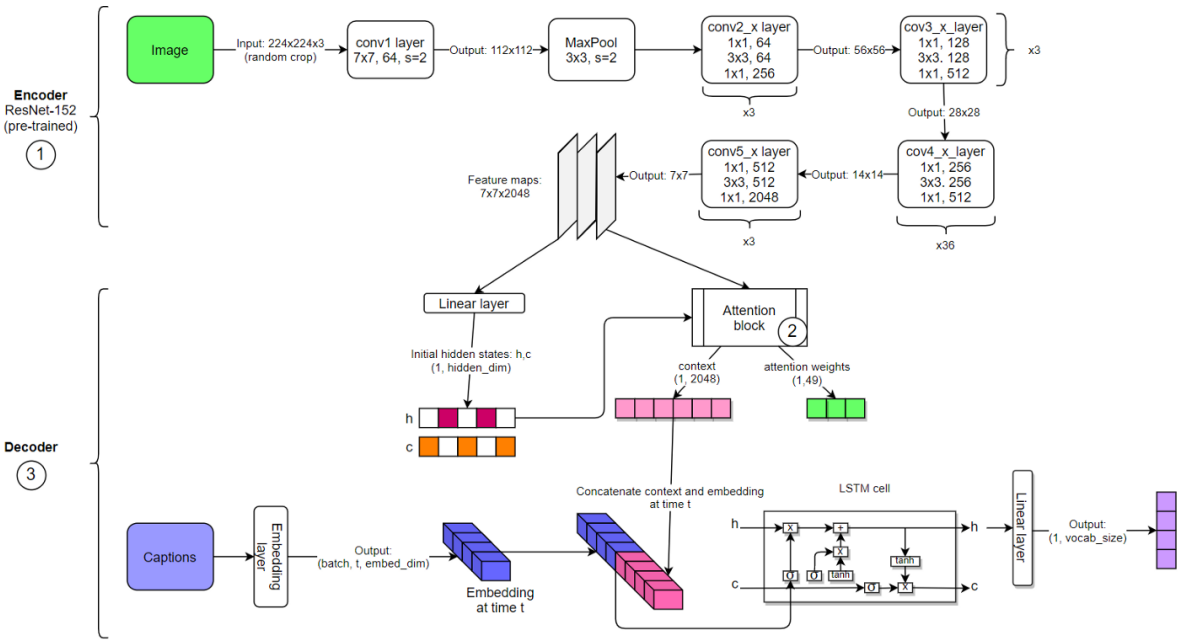

Figure 4: Model Architecture

4 Consensus-Loss

In this work, we explicitly engage human consensus to measure the quality of ground truth captions in advance, and directly encourage the model to learn high quality captions with high priority.

4.1 CIDEr Score

CIDEr calculates the cosine similarity between the candidate sentence and a set of reference sentences using its TF-IDF weightings ($r^n(.)$). Finally, all the n-gram cosine similarity are averaged to form a single score for each candidate sentence. By referencing the main paper, The CIDEr score is computed as follows -

The $C_{\text{score}}(\mathbf{R_i^I})$ is computed as follows:

$$\text{CIDEr}_n(\mathbf{R_i^I}, \mathbf{Q_i^I}) = \frac{1}{M-1} \sum_j \frac{\boldsymbol{r^n}(\mathbf{R_i^I}) \cdot \boldsymbol{r^n}(\mathbf{Q_{ij}^I})}{\|\boldsymbol{r^n}(\mathbf{R_i^I})\| \|\boldsymbol{r^n}(\mathbf{Q_{ij}^I})\|} \quad (1)$$

$$C_{\text{score}}(\mathbf{R_i^I}, \mathbf{Q_i^I}) = \sum_{n=1}^{4} \text{CIDEr}_n(\mathbf{R_i^I}, \mathbf{Q_i^I}), \quad (2)$$

Figure 5: CIDEr Score

4.2 Loss function

The standard cross-entropy (CE) loss measures the classification model performance based on the probability outputs from the model. The consensus loss was supposed to adjust the sentence-level loss, but we implemented consensus loss for a word predicted from a prefix of a sentence. This increases the loss of high quality captions, but reduces the loss of the low quality ones, by multiplying the calculated cross entropy loss with the cider score of the caption.

We write the simple Consensus Loss (CL) as:

$$\mathrm{CL}(I, \mathbf{S}) = -C_{\text{score}}(\mathbf{R_i^I}) \frac{1}{N} \sum_{t=1}^{N} \log p_t(S_t), \qquad (3)$$

where N is the length of caption, $\mathbf{R_i^I}$ is the i-th annotation for image I, $p_t \in [0, 1]$ is the likelihood for the correct word S_t.

Figure 6: Consensus loss

```
class ConsensusLoss(torch.nn.Module):

    def _init_(self):
        super(ConsensusLoss,self)._init_()

    def forward(self, outputs, targets, cider):

        den=0
        op = torch.exp(outputs[0])
        den = torch.sum(op)
        op = torch.div(op, den)

        ce = -1*torch.log(op[targets[0]])
        cl = (ce*cider)

        return cl
```

The consensus loss can be implemented in most of the image captioning models trained by cross entropy loss.

5 Contributions

We implemented the model on a different dataset, **Flickr 8k**, instead of the MS-COCO dataset used in the paper. We also changed the feature extraction model to **ResNet-18** instead of Faster-RCNN, to help apply the concepts learned in the DL lab. We experimented with **both LSTM and BERT**as our word embedding model, and combined the feature extraction and word embedding processes through a **fully connected network.**

While our final implementation(Version 5) is mentioned in detail in previous sections, we think it is important that we mention the experimentation we carried out that led to the final version.

- **Experiment 1** was the model pipeline we had built for the Mid-semester submission. The pipeline was very generic where we used pre-trained resnet-18 for extracting the image features, a PyTorch Embedding layer followed by a LSTM to extract the caption features, and finally added the image and caption features element-wise to form the input for our final dense layers. We trained the model on a small subset of train data. The results were pretty inaccurate with many repititions of phrases and we did not bother to measure the accuracy achieved on acual test data.

- **Experiment 2** was an attempt to incorporate BERT embedding into our model. We used the BERT tokenizer on top of our tokenizer and then used pre-trained BERT to generate the embeddings. This along with the image features extracted from resnet-18 was added element-wise and passed in dense layers to produce logits for the next predicted word. Cross-Entropy was used as the Loss function. We used a prefix of the actual sentence as the input and the output was the logits corresponding to the next word. There was some improvement as the repitition of predicted words was reduced considerably, but the results were still inaccurate. Moving on from this version, we thought we should try generating an entire sentence first, and then backpropagating the cumulative loss.

- **Experiment 3** had two major changes and other insignificant ones. Firstly, we changed the entire pre-processing and forward pass function to incorporate sentence generation before loss backpropagation, and removing the prefixed sentence data generated for earlier versions. Secondly, we built our own version of the Consensus loss function as mentioned in Section 4 to be more consisent with research paper selected. This calculated the loss for the entire sentence for each caption, taking the cider score for each caption as its weight. There were many unseen words predicted by this model but an overall sentence structure was lacking. There was not much correlation of the words with the image and the loss was giving values in 4 digits before the decimal point. We decided to switch back to the Prefix-style model but with a few changes. We also inferred from this version that BERT alone cannot extract a good enough embedding to work with only dense layers, and the additional BERT tokenizing might be causing problems in reverse vocabulary

- **Experiment 4** This is the final model we present. As per our inference from Version 3, we removed the BERT tokenizer and modidfied our tokenization to be compatible with BERT Embedding. We added an LSTM after BERT to further analyse the caption data and generate a better representation of its features. Here, we decided to concat the image features and the caption features instead of adding them element-wise before passing it into the dense layers. As mentioned before, we reverted back to the prefix-style input output where a prefix of sentence is given as input to the model, and outputs a tensor of logits for each word in the dictionary. We also changed our Consensus Loss module to calculate word-based loss instead of the earlier sentence-based one. The model was working better, but the words generated from the output had very little correlation with the images. We wanted to fix this next.

- **Experiment 5** is the final version of our implementation. Owing to the fact that there was less correlation among the words and the images, we tried adding a Visual Attention layer on the image features and passing them in the LSTM along with the caption embeddings. Further details about the final Version are mentioned in the previous sections.

6 Results

Even after trying out multiple experiments, we didn't quite get the results we were hoping for. We would like to note here that the paper did not have much information about the implementation of the architectures used. The authors have mentioned that details will be released in a Github repository, but we could not find the code publicly available. The paper was very hard to follow and had vague references to the actual architecture used, which is why we tried out 5 experiments using different ways to implement the image captioning model. Please refer the notebook to have a look at the training for our final version - which also includes the loss at different stages of the training.

7 Code

The full code for this project is publicly available in the Github repository

8 Acknowledgements

We would like to earnestly acknowledge the sincere efforts and valuable time given by Tirtharaj Dash sir and Tanmay Tulsidas Verlekar sir. Their teachings have helped us build up our knowledge base in Deep Learning. Their guidance and feedback has helped us choose this topic of research and in completing our project in the best way possible.

Also, we would like to express our gratitude to the teaching assistants of the course - Ramanathan Rajaraman, Ameya Thete, Param Biyani and Tanmay Devale, Lavish Ramchandani and Taresh Batra, who have helped us immensely along the way. Thank you.

9 References

- https://www.ijcai.org/proceedings/2020/0092.pdf
- https://towardsdatascience.com/a-guide-to-image-captioning-e9fd5517f350
- https://arxiv.org/pdf/1411.4555.pdf
- https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learni
- https://towardsdatascience.com/3-types-of-contextualized-word-embeddings-from-bert-using-transf
- https://arxiv.org/pdf/1502.03044.pdf
- https://medium.com/analytics-vidhya/image-captioning-with-attention-part-1-e8a5f783f6d3
- https://medium.com/analytics-vidhya/image-captioning-with-attention-part-2-f3616d5cf8d1
- https://discuss.pytorch.org/t/attention-in-image-classification/80147
- https://github.com/heykeetae/Self-Attention-GAN/blob/master/sagan_models.py#L8
- https://arxiv.org/pdf/1805.08318.pdf