Image Captioning Project Report

Introduction

The Vision Voice project is aimed at developing an advanced image captioning system capable of automatically generating descriptive textual captions for input images. This system is designed to enhance accessibility and user interaction by providing both visual and auditory outputs, making it particularly beneficial for visually impaired individuals and applications requiring automated image descriptions. We aimed to create an app where any visually impaired person can use it with the camera on and by pressing a button, an image would be captured and the caption generated would be conveyed via audio. By bridging the gap between computer vision and natural language processing, the project seeks to create a seamless experience where users can understand and interact with visual content through both text and speech.

Central to this endeavor is the utilization of the COCO (Common Objects in Context) dataset, a large-scale collection widely recognized for its comprehensive annotations and diversity. The COCO dataset comprises over 330,000 images, each annotated with five descriptive captions. These captions capture various aspects of the images, including objects, actions, and contexts, providing a rich source of linguistic diversity essential for training robust image captioning models. The extensive annotations facilitate the development of models that can generalize well across diverse visual scenarios, enabling the generation of nuanced and contextually accurate captions.

We started with the development of a user-friendly web interface using Streamlit, allowing users to upload or capture images and receive corresponding captions. The core of the system revolves around two primary components: the Image Encoder and the Caption Decoder. The Image Encoder extracts meaningful features from the input image, while the Caption Decoder generates coherent and contextually relevant captions based on these features. Throughout the project, various model architectures and training strategies were explored to enhance the quality of the generated captions. In this project we did development of a custom CNN-LSTM model, transitioning to pre-trained encoders like ResNet-50 and EfficientNet-B3, implementing beam search for improved decoding, and integrating the state-of-the-art BLIP model for enhanced performance.

Description of Individual Work

One of my initial contributions was developing the Streamlit application, Vision Voice, which serves as the user interface for the image captioning system. This app allows users to upload images or capture them using a camera and receive generated captions using different models. Additionally, it offers a text-to-speech feature to convert captions into audio, enhancing accessibility for users with visual impairments.

My next work was on developing and optimizing the Image Encoder and Caption Decoder, implementing beam search to enhance caption generation, and designing effective training and evaluation frameworks. These components are fundamental to the system's ability to accurately interpret images and produce meaningful captions.

Image Encoder

The Image Encoder is pivotal in extracting high-level features from input images, serving as the foundation for generating accurate captions. Initially, we had explored building a custom Convolutional Neural Network (CNN) tailored to the captioning task. However, the performance of this rudimentary model was not good and captions were very bad with negligible BLEU scores. Recognizing the limitations of a custom architecture, I transitioned to leveraging pre-trained models because for their robust feature extraction capabilities.

To enhance feature extraction, I integrated EfficientNet-B3 and ResNet-50 as the backbone of the Image Encoder. These models, pre-trained on extensive datasets like ImageNet, offer rich and nuanced feature representations that significantly improve the system's ability to generate meaningful captions. By freezing the majority of their layers and fine-tuning only the last few, I ensured that the models retained their powerful feature extraction capabilities while adapting to the specific nuances of the captioning task.

EfficientNet-B3 Encoder Implementation:

```
import torch.nn as nn
import torchvision.models as models

class ImageEncoder(nn.Module):
    def __init__(self, embedding_dim, trainable_layers=0):
        super(ImageEncoder, self).__init__()
        base_model = models.efficientnet_b3(pretrained=True)
        for param in base_model.parameters():
            param.requires_grad = False
        if trainable_layers > 0:
            feature_params = list(base_model.features.parameters())[-trainable_layers:]
            for param in feature_params:
                param.requires_grad = True
        self.feature_extractor = base_model.features
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(1536, embedding_dim)

    def forward(self, images):
        x = self.feature_extractor(images)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        embeddings = self.fc(x)
        return embeddings

```

This implementation highlights the integration of EfficientNet-B3, where the feature extractor processes the input images to produce high-dimensional feature maps. An adaptive average pooling layer reduces these maps to a fixed size, and a fully connected layer maps the pooled features into a lower-dimensional embedding space suitable for the Caption Decoder.

Caption Decoder

The Caption Decoder translates the image embeddings into coherent and contextually relevant textual captions. Initially, I employed a simple Long Short-Term Memory (LSTM) network, which, while capable of handling sequential data, exhibited limitations in generating diverse and accurate captions. To address these shortcomings, I enhanced the decoder with beam search—a more sophisticated decoding strategy—and explored integrating more advanced models like BLIP.

Caption Decoder Implementation:

```
import torch.nn as nn
```

```python
import torch.nn.functional as F

class CaptionDecoder(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, vocab, num_layers=1):
        super(CaptionDecoder, self).__init__()
        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, vocab_size)
        self.vocab = vocab
        self.start_token_id = vocab(vocab.start_word)
        self.end_token_id = vocab(vocab.end_word)

    def forward(self, image_features, captions):
        caption_embeddings = self.word_embeddings(captions[:, :-1])
        inputs = torch.cat((image_features.unsqueeze(1), caption_embeddings), dim=1)
        lstm_output, _ = self.lstm(inputs)
        outputs = self.fc(lstm_output)
        return outputs

    def generate_caption(self, image_features, beam_size=3, max_length=20):
        device = image_features.device
        vocab_size = self.fc.out_features
        assert image_features.size(0) == 1, "Batch size should be 1 for inference."
        start_token_id = self.start_token_id
        end_token_id = self.end_token_id
        h = torch.zeros(self.lstm.num_layers, beam_size, self.hidden_dim, device=device)
        c = torch.zeros(self.lstm.num_layers, beam_size, self.hidden_dim, device=device)
        seqs = torch.LongTensor([[start_token_id]]).to(device).repeat(beam_size, 1)
        top_k_scores = torch.zeros(beam_size, 1, device=device)
        image_features = image_features.repeat(beam_size, 1).unsqueeze(1)
        with torch.no_grad():
            lstm_output, (h, c) = self.lstm(image_features, (h, c))
        complete_seqs = []
        complete_seqs_scores = []
        for step in range(max_length):
            last_word_ids = seqs[:, -1]
            embeddings = self.word_embeddings(last_word_ids).unsqueeze(1)
            lstm_output, (h, c) = self.lstm(embeddings, (h, c))
            scores = F.log_softmax(self.fc(lstm_output.squeeze(1)), dim=1)
            scores = top_k_scores.expand_as(scores) + scores
               top_k_scores_flat,  top_k_words_flat  =  scores.view(-1).topk(beam_size,  dim=0,
largest=True, sorted=True)
            prev_word_inds = top_k_words_flat // vocab_size
            next_word_inds = top_k_words_flat % vocab_size
            new_seqs = torch.cat([seqs[prev_word_inds], next_word_inds.unsqueeze(1)], dim=1)
            new_scores = top_k_scores_flat
            complete_inds = (next_word_inds == end_token_id).nonzero(as_tuple=False).squeeze(1)
           incomplete_inds = (next_word_inds != end_token_id).nonzero(as_tuple=False).squeeze(1)
            if len(complete_inds) > 0:
                for i in complete_inds:
                    complete_seqs.append(new_seqs[i].tolist())
                    complete_seqs_scores.append(new_scores[i].item())
            k = beam_size - len(complete_seqs)
            if k <= 0:
                break
            seqs = new_seqs[incomplete_inds]
            top_k_scores = new_scores[incomplete_inds].unsqueeze(1)
            h = h[:, prev_word_inds[incomplete_inds], :]
            c = c[:, prev_word_inds[incomplete_inds], :]
        if len(complete_seqs_scores) == 0:
            complete_seqs = seqs.tolist()
            complete_seqs_scores = top_k_scores.squeeze(1).tolist()
        best_idx = complete_seqs_scores.index(max(complete_seqs_scores))
        best_seq = complete_seqs[best_idx]
        words = [self.vocab.idx2word[idx] for idx in best_seq]
        if end_token_id in best_seq:
            words = words[:best_seq.index(end_token_id)]
        return words

```

This implementation showcases the integration of beam search within the Caption Decoder. By maintaining multiple candidate sequences (beams) at each decoding step, the decoder explores various potential sequences, increasing the likelihood of generating coherent and contextually accurate captions. The cumulative scoring ensures that the most probable sequences are retained, while pruning maintains computational efficiency.

Beam Search Implementation

Beam search is a decoding strategy that significantly enhances the quality of generated captions by considering multiple candidate sequences simultaneously. Unlike greedy decoding, which selects the most probable next word at each step without considering future implications, beam search maintains several beams and evaluates them based on their cumulative probabilities.

The algorithm operates as follows:

1. Initialization: Begin with the <start> token and initialize multiple beams.
2. Expansion: At each timestep, expand each beam by considering all possible next words.
3. Scoring: Calculate the cumulative probability for each candidate sequence.
4. Pruning: Retain only the top *k* sequences (beam size) based on their scores.
5. Termination: Continue until all beams have generated an <end> token or reached the maximum caption length.

This approach allows the model to explore a broader space of possible captions, reducing the likelihood of getting trapped in locally optimal but globally suboptimal sequences. Consequently, the generated captions are more coherent, descriptive, and contextually appropriate.

Training and Evaluation

Next I focused on the training part where I utilized the data preprocessing and data loaders and vocabulary already build by my teammates. I used the following-

- Data Preprocessing: Applied transformations such as resizing, random cropping, and horizontal flipping to augment the training data. These augmentations enhance the model's ability to generalize across diverse visual scenarios, preventing overfitting and improving robustness.
- Multiple Captions per Image: Instead of training on a single caption per image, I leveraged all available captions (typically five in the COCO dataset). This approach exposes the model to a variety of linguistic expressions and vocabulary, enriching its ability to generate diverse and descriptive captions.
- Optimization: Utilized the Adam optimizer with learning rate schedulers to facilitate stable and efficient convergence. The learning rate was adjusted dynamically to ensure that the model converges without overshooting minima.

- Loss Function: Employed cross-entropy loss, configured to ignore padding tokens. This ensures that the model focuses on predicting meaningful words, enhancing the quality of the generated captions.
- Batch Processing: Handled batches of images and their corresponding captions, allowing efficient utilization of computational resources and accelerating the training process.
- Gradient Handling: Ensured proper backpropagation by zeroing gradients before each optimization step and accumulating loss across multiple captions per image. This approach maintains the integrity of gradient updates, preventing gradient accumulation from previous batches.
- Epoch Management: Trained the model over multiple epochs, systematically improving performance with each iteration. Early stopping mechanisms can be incorporated in future iterations to prevent overfitting.

Training on All Captions

Training the models on all available captions per image was instrumental in achieving higher BLEU scores. This approach exposed the Caption Decoder to a variety of linguistic expressions and vocabularies, enabling it to generate more diverse and descriptive captions. The enrichment in training data directly contributed to the improvement in BLEU scores from 0.07 to approximately 0.149.

The benefits observed from this strategy include:

- **Linguistic Diversity:** The model could generate captions with varied sentence structures and word choices, making the outputs less repetitive and more natural.
- **Enhanced Vocabulary:** Exposure to multiple captions enriched the model's vocabulary, allowing it to describe images with greater precision and detail.
- **Contextual Understanding:** The model developed a better understanding of contextual cues, leading to captions that better reflect the scene's complexities.

Evaluation Metrics

To quantitatively assess the model's performance, I utilized the BLEU (Bilingual Evaluation Understudy) score—a standard metric for evaluating the similarity between generated captions and reference captions. BLEU measures the overlap of n-grams between the generated captions and reference captions, providing an objective measure of caption quality.

The evaluation process involved generating captions for each image in the validation set using the trained models and comparing them against all reference captions. Higher BLEU scores indicate greater similarity and, by extension, better caption quality.

Results

Throughout the development and training phases, the model's performance was evaluated using BLEU scores. The progression from initial to final models showcases significant improvements, reflecting the effectiveness of the implemented strategies.

The following was the scores-

1. Custom CNN + LSTM (BLEU ≈ 0.004):
    - Performance: The initial custom CNN encoder paired with an LSTM decoder produced nearly incoherent captions.
    - Reason: The simplistic architecture of the custom CNN was insufficient for capturing complex visual features necessary for meaningful caption generation.
2. ResNet-50 + LSTM (BLEU ≈ 0.07):
    - Performance: Transitioning to a pre-trained ResNet-50 encoder markedly improved caption quality, trained on one caption.
    - Reason: ResNet-50's robust feature extraction capabilities provided richer image embeddings, enabling the decoder to generate more coherent and relevant captions.
3. ResNet-50 + LSTM + Beam Search (BLEU ≈ 0.149):
    - Performance: Introducing beam search and training on all available captions per image resulted in a substantial increase in BLEU score.
    - Reason: Beam search enhanced the decoder's ability to generate coherent and contextually accurate captions by considering multiple candidate sequences. Training on multiple captions enriched the linguistic diversity, allowing the model to produce more varied and descriptive outputs.
4. EfficientNet-B3 + LSTM + Beam Search (BLEU ≈ 0.139):
    - Performance: While EfficientNet-B3 offered computational efficiency and strong feature representations, its BLEU score was slightly lower than ResNet-50 under similar conditions.
    - Reason: This indicates that, within the context of this project, ResNet-50 and efficient Net B3 provided more effective features for caption generation, possibly due to its deeper architecture capturing more intricate visual details.

Integration of the BLIP Model

To further enhance the system's performance and provide users with a state-of-the-art alternative for caption generation, I integrated the **BLIP (Bootstrapping Language-Image Pre-training)** model developed by Salesforce. BLIP combines transformer-based architectures with extensive pre-training on vision-language data, offering superior performance in image captioning tasks.

BLIP Model Integration:

```
import requests
from PIL import Image
from transformers import BlipProcessor, BlipForConditionalGeneration

```

```python
# Initialize BLIP processor and model
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")

def generate_caption_blip(image, text=None):
    if text:
        inputs = processor(image, text, return_tensors="pt")
    else:
        inputs = processor(image, return_tensors="pt")
    out = model.generate(**inputs)
    return processor.decode(out[0], skip_special_tokens=True)

```

This integration provided an alternative pathway for generating high-quality captions. The BLIP model leverages transformer-based architectures, which excel in capturing long-range dependencies and contextual nuances, thereby producing more coherent and contextually accurate descriptions compared to traditional LSTM-based decoders.

- **Processor and Model Initialization:** Loaded the BLIP processor and model from Salesforce's pre-trained repository, ensuring access to state-of-the-art captioning capabilities.
- **Caption Generation Function:** Defined a function to generate captions using BLIP, supporting both conditional and unconditional image captioning. This flexibility allows users to provide prompts or generate captions solely based on the image.

5. Summary and Conclusions

The Vision Voice project has successfully developed an image captioning system that evolves from a basic custom model to a sophisticated setup utilizing pre-trained encoders, beam search, and advanced models like BLIP. The progression of BLEU scores from 0.004 to 0.149 underscores the effectiveness of the implemented strategies and enhancements.

Key Learnings:

1. **Robust Feature Extraction:** Leveraging pre-trained models like ResNet-50 and EfficientNet-B3 significantly enhances the quality of image feature extraction, providing a strong foundation for accurate caption generation. These models capture intricate visual details that are crucial for generating meaningful and contextually relevant captions.
2. **Beam Search Decoding:** Implementing beam search improves the coherence and contextual relevance of generated captions by considering multiple candidate sequences during decoding. This strategy mitigates the limitations of greedy decoding, resulting in more descriptive and accurate captions.
3. **Diverse Training Data:** Training on all available captions per image exposes the model to a wide range of linguistic expressions, enhancing its ability to handle a variety of scenes and objects. This diversity is essential for generating varied and natural-sounding captions.
4. **Integration of Advanced Models:** Incorporating state-of-the-art models like BLIP provides alternative pathways for caption generation, maintaining high performance and offering flexibility in model selection. The transformer-based architecture of BLIP further elevates the system's capability to produce nuanced and contextually rich captions.

These results demonstrate how each incremental improvement—from adopting powerful pre-trained encoders to implementing sophisticated decoding strategies and enriching training data—contributed to the overall enhancement of the image captioning system.

Future Improvements:

1. **Incorporate Attention Mechanisms:** Integrating attention layers can allow the decoder to focus on specific parts of the image while generating each word, potentially enhancing caption relevance and detail. Attention mechanisms facilitate a more dynamic and context-aware caption generation process.

2. **Explore Transformer-Based Decoders:** Adopting transformer architectures could further improve sequence generation capabilities, leveraging their superior handling of long-range dependencies. Transformers have shown remarkable performance in various natural language processing tasks and could elevate the quality of generated captions.

3. **Expand Training Datasets:** Utilizing larger and more diverse datasets can enhance the model's generalization, making it more robust across varied image contexts. Access to a broader range of images and captions would enable the model to learn from a more extensive array of visual and linguistic patterns.

4. **Enhanced Evaluation Metrics:** Incorporating additional evaluation metrics such as METEOR, ROUGE, and CIDEr can provide a more comprehensive assessment of caption quality. These metrics capture different aspects of similarity and can offer deeper insights into the model's performance.

In conclusion, the Vision Voice project demonstrates the critical interplay between model architecture, training methodologies, and decoding strategies in developing an effective image captioning system. The significant improvements in BLEU scores reflect the success of these strategies, paving the way for future enhancements and broader applications in accessibility and human-computer interaction.

Percentage of Code Sourced Online

Total lines – 533

Borrowed from external – 60

Modified -20 Added- 50

Percentage = (60-20/60+50)*100 = 36.36%

References

1. https://cocodataset.org/#captions-2015
2. https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/

3. https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/ (Avg, tensorflow, pretrained xception + lstm, can be referred)
4. Image Captioning (Flikr30k): https://github.com/nirajankarki5/Flickr30k-Image-Caption-Generator-Using-Deep-Learning?tab=readme-ov-file (fine, tensorflow, can be referred for Glove embeddings use)

5. 👎Image Captioning (Flikr30k): https://www.youtube.com/watch?v=fUSTbGrL1tc (tensorflow, pretrained, simple)

6. Multi30k - https://github.com/multi30k/dataset