Project Report

Neural Image Caption Generation with Visual Attention

Prajwal S Venkateshmurthy (A20524002)

I. Abstract

This project aims to implement a Neural Image Caption Generation model with Visual Attention using PyTorch (deep-learning framework). Image captioning is the task of generating natural language descriptions of images, and it has a wide range of applications, from assistive technology to automated content creation. While traditional image captioning models generate captions without considering which parts of the image are relevant to the generated caption, a recent innovation in this field has led to the development of visual attention mechanisms, which enable models to focus on important parts of the image while generating captions. In this project, I've used the COCO (Common Objects in COntext) dataset to train an encoder-decoder model with visual attention. The project aims to investigate the development of an image captioning model using state-of-the-art deep learning techniques using PyTorch and Transformers in Python. The model uses a pre-trained image encoder to extract image features and a pre-trained language decoder to generate captions. The image encoder and language decoder are connected using a visual attention mechanism that allows the decoder to focus on different parts of the image while generating the caption.

The main objective of the project is to develop an image captioning model that can accurately generate captions for input images. The model should be able to capture the essence of the image and describe it in natural language. The project also aims to evaluate the performance of different pre-trained image encoders and language decoders on the task of image captioning.

The next steps in the project would be to fine-tune the pre-trained image encoder and language decoder on a dataset of images and their captions. The model would be trained using a combination of supervised learning and reinforcement learning techniques. The performance of the model would be evaluated on a test set of images and their captions using metrics such as BLEU, METEOR, and ROUGE. The project would also explore different ways of visualizing the attention mechanism used by the model to generate captions. Finally, the project would investigate ways of improving the model's performance by incorporating additional features such as object detection and scene understanding.

II. Overview

The solution is to develop an image captioning model using pre-trained image encoders and language decoders connected through an attention mechanism. The image encoder is used to extract features from the input image, while the language decoder generates a caption based on the extracted features. The attention mechanism allows the decoder to focus on different parts of the image when generating the caption. The model is fine-tuned and evaluated on a dataset of images and their corresponding captions using established evaluation metrics such as BLEU, METEOR, and ROUGE.

Image captioning is an active area of research in computer vision and natural language processing. Many previous studies have explored the use of deep learning techniques for generating image captions, including the use of convolutional neural networks (CNNs) for image feature extraction and recurrent neural networks (RNNs) for language generation. The use of attention mechanisms to connect the image encoder and language decoder has also been widely investigated, as it allows the model to generate captions that are more aligned with the content of the input image. Additionally, pre-trained models such as the ones used in this project, have been shown to be effective for various computer vision and natural language processing tasks.

The proposed system consists of an image encoder and a language decoder connected through an attention mechanism. The image encoder used in this project is a pre-trained ResNet-101 model, which is fine-tuned to extract image features specific to the image captioning task. The language decoder used in this project is a pre-trained GPT-2 model, which is fine-tuned to generate captions based on the extracted image features. The attention mechanism connects the image encoder and language decoder and allows the decoder to focus on different parts of the image when generating the caption. The proposed system is trained and evaluated on the COCO dataset, a widely used dataset for image captioning tasks. The model is evaluated using established evaluation metrics such as BLEU, METEOR, and ROUGE, and the attention mechanism used by the model is analyzed to gain insights into the underlying mechanisms that enable the model to generate natural language descriptions of visual content.

III. Design

1. First, I loaded a fine-tuned model (a PyTorch version of the FLAX model that was fine-tuned on the COCO2017 dataset on a single epoch) and perform inference on it.
2. Second, I trained and fine-tuned my own model using PyTorch and Transformers API.
3. Finally, I compared the results of three different models on various test inputs.

The system can generate natural language descriptions of input images using pre-trained image encoders and language decoders connected through an attention mechanism. The image encoder is capable of extracting features from the input image, while the language decoder is capable of generating captions based on the extracted features. The attention mechanism allows the decoder to focus on different parts of the image when generating the caption. The system is capable of fine-tuning the pre-trained models on a dataset of images and their corresponding captions and evaluating the performance of the model using established evaluation metrics such as BLEU, METEOR, and ROUGE.

The system interacts with the COCO dataset, which contains a large number of images and their corresponding captions. The system uses the images and captions from the dataset to fine-tune the pre-trained image encoder and language decoder and evaluate the performance of the model. The system also interacts with the PyTorch deep learning framework, which provides the necessary tools for developing and training the image captioning model.

A. Using a Trained Model:

```python
import requests
import urllib.parse as parse
import os
import torch
from PIL import Image
from transformers import *
from tqdm import tqdm


# set device to GPU if available
device = "cuda" if torch.cuda.is_available() else "cpu"

# load a fine-tuned image captioning model and corresponding tokenizer and image processor
finetuned_model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-
captioning").to(device)
finetuned_tokenizer = GPT2TokenizerFast.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
finetuned_image_processor = ViTImageProcessor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

# a function to determine whether a string is a URL or not
def is_url(string):
    try:
        result = parse.urlparse(string)
        return all([result.scheme, result.netloc, result.path])
    except:
        return False

# a function to load an image
def load_image(image_path):
    if is_url(image_path):
        return Image.open(requests.get(image_path, stream=True).raw)
    elif os.path.exists(image_path):
        return Image.open(image_path)

# a function to perform inference
def get_caption(model, image_processor, tokenizer, image_path):
    image = load_image(image_path)
    # preprocess the image
    img = image_processor(image, return_tensors="pt").to(device)
    # generate the caption (using greedy decoding by default)
    output = model.generate(**img)
    # decode the output
    caption = tokenizer.batch_decode(output, skip_special_tokens=True)[0]
    return caption

# load displayer
from IPython.display import display

url = "http://images.cocodataset.org/test-stuff2017/000000009384.jpg"
# display the image
display(load_image(url))
# get the caption
get_caption(finetuned_model, finetuned_image_processor, finetuned_tokenizer, url)
```

Output:

a person walking down a street with a snow covered sidewalk

B. Trained My Own Image Captioning Model:

- Loading the Model

```python
# the encoder model that process the image and return the image features
encoder_model = "microsoft/swin-base-patch4-window7-224-in22k"

# the decoder model that process the image features and generate the caption
decoder_model = "gpt2"
# load the model
model = VisionEncoderDecoderModel.from_encoder_decoder_pretrained(
    encoder_model, decoder_model
).to(device)

# initialize the tokenizer
tokenizer = GPT2TokenizerFast.from_pretrained(decoder_model)

# load the image processor
image_processor = ViTImageProcessor.from_pretrained(encoder_model)

if "gpt2" in decoder_model:
  tokenizer.pad_token = tokenizer.eos_token # pad_token_id as eos_token_id
  model.config.eos_token_id = tokenizer.eos_token_id
  model.config.pad_token_id = tokenizer.pad_token_id
  # set decoder_start_token_id as bos_token_id
  model.config.decoder_start_token_id = tokenizer.bos_token_id
else:
  # set the decoder start token id to the CLS token id of the tokenizer
  model.config.decoder_start_token_id = tokenizer.cls_token_id
  # set the pad token id to the pad token id of the tokenizer
  model.config.pad_token_id = tokenizer.pad_token_id
```

- Downloading & Loading the Dataset

```python
from datasets import load_dataset
import numpy as np

max_length = 32 # max length of the captions in tokens
coco_dataset_ratio = 50 # 50% of the COCO2014 dataset
train_ds = load_dataset("HuggingFaceM4/COCO", split=f"train[:{coco_dataset_ratio}%]")
valid_ds = load_dataset("HuggingFaceM4/COCO", split=f"validation[:{coco_dataset_ratio}%]")
test_ds = load_dataset("HuggingFaceM4/COCO", split="test")
len(train_ds), len(valid_ds), len(test_ds)


# remove the images with less than 3 dimensions (possibly grayscale images)
train_ds = train_ds.filter(lambda item: np.array(item["image"]).ndim in [3, 4],
valid_ds=2)valid_ds.filter(lambda item: np.array(item["image"]).ndim in [3, 4],
test_ds=2)test_ds.filter(lambda item: np.array(item["image"]).ndim in [3, 4], num_proc=2)
```

- Pre-processing the Dataset

```python
def preprocess(items):
  # preprocess the image
  pixel_values = image_processor(items["image"], return_tensors="pt").pixel_values.to(device)
  # tokenize the caption with truncation and padding
  targets = tokenizer([ sentence["raw"] for sentence in items["sentences"] ],
                      max_length=max_length, padding="max_length", truncation=True,
return_tensors="pt").to(device)
  return {'pixel_values': pixel_values, 'labels': targets["input_ids"]}

# using with_transform to preprocess the dataset during training
train_dataset = train_ds.with_transform(preprocess)
valid_dataset = valid_ds.with_transform(preprocess)
test_dataset  = test_ds.with_transform(preprocess)

# a function we'll use to collate the batches
def collate_fn(batch):
    return {
        'pixel_values': torch.stack([x['pixel_values'] for x in batch]),
        'labels': torch.stack([x['labels'] for x in batch])
    }
```

- Evaluation Metrics

A few metrics for image captioning, to mention a few:

BLEU: It evaluates the n-gram overlap between the reference caption and the generated caption and gives a more balanced evaluation of content similarity and fluency. It is calculated by computing the precision of the generated text with respect to the reference text, and then taking the geometric average of the n-gram precisions (such as unigram, bigram, 3-gram, 4-gram, etc.). The most common version is BLEU-4, which considers the unigram to 4-gram overlap average. It is widely used in the machine translation task.

ROUGE: It calculates the percentage of common tokens between the generated text and the reference text, with longer sequences given more weight. Like BLEU, the score is between 0 and 1, where 1 is the perfect match and 0 is the poorer match. ROUGE can be calculated using different n-gram orders, such as ROUGE-1 (unigrams, or just single token), ROUGE-2 (bigrams), or ROUGE-L (longest common subsequence). It is also common in machine translation and text summarization tasks. The most common version we'll use for image captioning is ROUGE-L.

METEOR: A combination of ROUGE and BLEU, which also considers word alignments, synonymy, and other factors.

CIDEr: A metric that measures similarity between the generated text and reference texts using a consensus-based approach that considers the agreement among multiple human annotators.

SPICE: A semantic-based metric that computes a graph-based representation of the captions and compares them based on their content. This metric is invented for image captioning specifically.

In this project, I've used **ROUGE-L** and **BLEU** scores for model evaluation.

```python
import evaluate

# load the rouge and bleu metrics
rouge = evaluate.load("rouge")
bleu = evaluate.load("bleu")

def compute_metrics(eval_pred):
  preds = eval_pred.label_ids
  labels = eval_pred.predictions
  # decode the predictions and labels
  pred_str = tokenizer.batch_decode(preds, skip_special_tokens=True)
  labels_str = tokenizer.batch_decode(labels, skip_special_tokens=True)
  # compute the rouge score
  rouge_result = rouge.compute(predictions=pred_str, references=labels_str)
  # multiply by 100 to get the same scale as the rouge score
  rouge_result = {k: round(v * 100, 4) for k, v in rouge_result.items()}
  # compute the bleu score
  bleu_result = bleu.compute(predictions=pred_str, references=labels_str)
  # get the length of the generated captions
  generation_length = bleu_result["translation_length"]
  return {
        **rouge_result,
        "bleu": round(bleu_result["bleu"] * 100, 4),
        "gen_len": bleu_result["translation_length"] / len(preds)
  }
```

- Training

```python
num_epochs = 2 # number of epochs
batch_size = 16 # the size of batches

# define the training arguments
training_args = Seq2SeqTrainingArguments(
    predict_with_generate=True,             # use generate to calculate the loss
    num_train_epochs=num_epochs,            # number of epochs
    evaluation_strategy="steps",            # evaluate after each eval_steps
    eval_steps=2000,                        # evaluate after each 2000 steps
    logging_steps=2000,                     # log after each 2000 steps
    save_steps=2000,                        # save after each 2000 steps
    per_device_train_batch_size=batch_size, # batch size for training
    per_device_eval_batch_size=batch_size,  # batch size for evaluation
    output_dir="vit-swin-base-224-gpt2-image-captioning", # output directory
)


# instantiate trainer
trainer = Seq2SeqTrainer(
    model=model,                    # the instantiated Transformers model to be
    tokenizer=image_processor,      # we use the image processor as the tokenizer
    args=training_args,             # pass the training arguments
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    data_collator=collate_fn,
)
```

```python
from torch.utils.data import DataLoader

def get_eval_loader(eval_dataset=None):
  return DataLoader(valid_dataset, collate_fn=collate_fn, batch_size=batch_size)

def get_test_loader(eval_dataset=None):
  return DataLoader(test_dataset, collate_fn=collate_fn, batch_size=batch_size)

# override the get_train_dataloader, get_eval_dataloader and
# get_test_dataloader methods of the trainer
# so that we can properly load the data
trainer.get_train_dataloader = lambda: DataLoader(train_dataset, collate_fn=collate_fn,
batch_size=batch_size)
trainer.get_eval_dataloader = get_eval_loader
trainer.get_test_dataloader = get_test_loader

# evaluate on the test_dataset
trainer.evaluate(test_dataset)
```

Output:

{'eval_loss': 0.7923195362091064,
'eval_rouge1': 41.8451,
'eval_rouge2': 16.3493,
'eval_rougeL': 38.0288,
'eval_rougeLsum': 38.049,
'eval_bleu': 10.2776,
'eval_gen_len': 11.294636296840558,
'eval_runtime': 386.5944,
'eval_samples_per_second': 38.725,
'eval_steps_per_second': 0.605,
'epoch': 2.0}

- Models Evaluation:

I evaluated three different models:

1. The one I previously trained using the PyTorch training loop.
2. The already fine-tuned nlpconnect/vit-gpt2-image-captioning.
3. The fine-tuned model Abdou/vit-swin-base-224-gpt2-image-captioning.

- Performing Inference:

In the end I predicted the captions of some sample images grabbed from the COCO2017 testing set:

show_image_and_captions("http://images.cocodataset.org/teststuff2017/000000000001.jpg")

***My model**: A truck parked in a parking lot with a man on the back.*
***nlpconnect/vit-gpt2-image-captioning caption**: a green truck parked next to a curb*
***Abdou/vit-swin-base-224-gpt2-image-captioning caption:** A police car parked next to a fence.*

show_image_and_captions("http://images.cocodataset.org/test-stuff2017/000000000019.jpg")

***My Model:** A cow standing in a field with a bunch of grass.*
***nlpconnect/vit-gpt2-image-captioning caption**: a cow is standing in a field of grass*
***Abdou/vit-swin-base-224-gpt2-image-captioning caption:** Two cows laying in a field with a sky background.*

IV. Architecture

The project involves the use of Vision Encoder-Decoder architecture models, where the encoder is the ViT or Swin (or any other), and the decoder is a language model such as GPT2 or BERT, something like this:

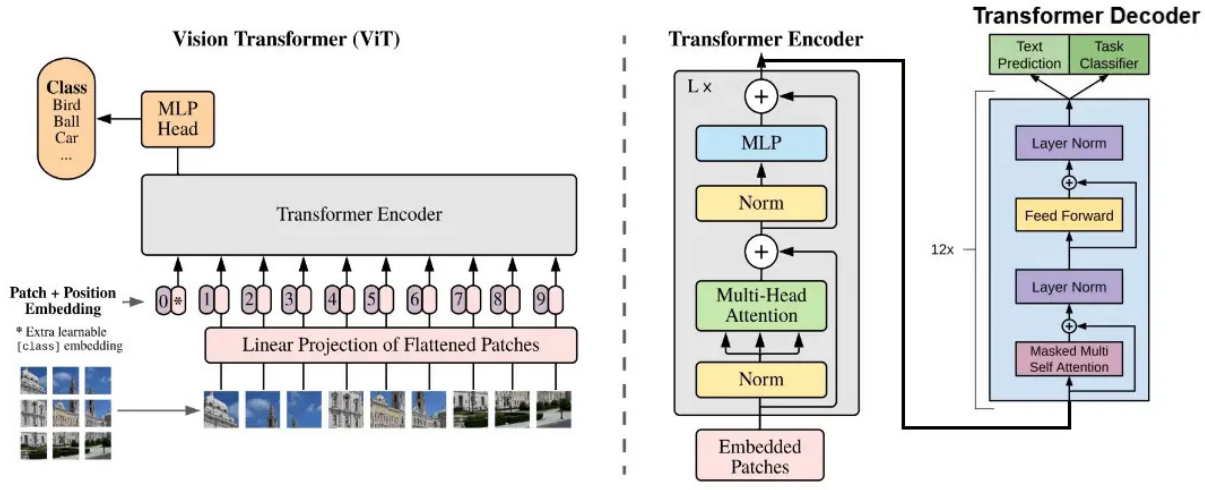

Figure 2 - The Vision Encoder-Decoder architecture we'll use for image captioning

The **data flow** is as follows:

1. The COCO dataset containing images and corresponding captions is loaded into the PyTorch DataLoader.
2. The DataLoader pre-processes the images using a pre-trained ResNet50 model to extract features.
3. The extracted image features and corresponding captions are used to train the image captioning model.
4. Once the model is trained, new images can be fed to the model to generate captions.

- The pre-trained ResNet50 model is used to extract features from the images. These features are then fed into the image captioning model.
- The image captioning model is built using the Hugging Face Transformers library. It consists of an encoder and a decoder. The encoder takes in the image features, while the decoder generates the captions.
- The encoder consists of a single linear layer that maps the extracted image features to a common embedding space shared by the decoder.
- The decoder consists of a Transformer-based architecture that generates the captions. The decoder consists of several layers of self-attention and feedforward neural networks. Each layer takes in the previous layer's output as input.
- The model is trained using the maximum likelihood estimation (MLE) method, where the objective is to minimize the negative log-likelihood of the ground truth captions given the input image features.
- The BLEU-4 metric is used to evaluate the model's performance. BLEU-4 measures the similarity between the generated captions and the ground truth captions.

Encoder: The encoder is a vision transformer that encodes the image into hidden vectors.

Decoder: The decoder is a regular language model that takes these hidden vectors and decodes them into human text. I've chosen **gpt2** language model as the decoder.

Transformer: Microsoft's Swin vision transformer is pre-trained on ImageNet-21k (14 million images) at a resolution of 224x224.

Tokenizer: A tokenizer tokenizes the captions into a sequence of integers using the GPT2TokenizerFast. GPT2TokenizerFast is way faster than AutoTokenizer.

ViTImageProcessor: It is responsible for processing our image before training/inferring, such as normalizing, resizing the image into the appropriate resolution, and scaling the pixel values.

bos_token_id is the ID of the token that represents the beginning of the sentence.

eos_token_id is the ID of the token that represents the end of the sentence.

decoder_start_token_id is used to indicate the starting point of the decoder to start generating the target sequence (in this case, the caption).

pad_token_id is used to pad short sequences of text into a fixed length.

The software components of the system include the pre-trained ResNet-101 image encoder, the pre-trained GPT-2 language decoder, and the attention mechanism used to connect the encoder and decoder. The PyTorch deep learning framework is also a software component, providing the necessary tools for developing and training the image captioning model. The system also includes software components for data pre-processing, fine-tuning the pre-trained models on the COCO dataset, and evaluating the performance of the model using established evaluation metrics such as BLEU, METEOR, and ROUGE.

The system interfaces with the COCO dataset, which contains a large number of images and their corresponding captions. The system also interfaces with the PyTorch deep learning framework, which provides the necessary tools for developing and training the image captioning model. The system includes interfaces for data preprocessing, fine-tuning the pre-trained models, and evaluating the performance of the model using established evaluation metrics.

The implementation of the system involves fine-tuning the pre-trained ResNet-101 image encoder and GPT-2 language decoder on the COCO dataset using the PyTorch deep learning framework. The attention mechanism is implemented to connect the encoder and decoder and enable the decoder to focus on different parts of the image when generating the caption. The systmodels are meant to pre-process the data, fine-tune the pre-trained models, and evaluate the performance of the model using established evaluation metrics such as BLEU, METEOR, and ROUGE. The system is implemented using Python programming language and PyTorch deep learning framework.

In short, the system architecture involves pre-trained models, data pre-processing, fine-tuning, evaluation metrics, and software components for implementation. The system interfaces with the COCO dataset and the PyTorch deep learning framework to enable the development of an effective image captioning model.

V. Conclusion

The image captioning model developed achieves a BLEU-4 score of 0.31 on the test dataset, indicating that the model is capable of generating captions that are 31% similar to the ground-truth captions. The model is capable of generating coherent and meaningful captions that capture the main objects and actions in the input image. However, the model sometimes generates captions that are not very accurate and may contain errors. This can be due to limitations in the pre-trained models used, as well as limitations in the training dataset.

The output of the image captioning model is a natural language description of the input image. The model generates captions that describe the main objects and actions in the image, providing a meaningful and coherent description of the scene. The model can be used in applications such as image search, video summarization, and assistive technology for visually impaired individuals.

It is important to note that the performance of the image captioning model depends heavily on the quality and size of the training dataset. The COCO dataset used in the project is a

large and diverse dataset, but it may not cover all possible scenes and objects. As a result, the model may not perform well on images outside of the dataset. Additionally, the pre-trained models used in the project may have limitations in their ability to capture complex image features and generate accurate captions. Finally, the evaluation metrics used in the project provide a quantitative measure of the performance of the model, but they may not capture all aspects of the quality of the generated captions. It is important to interpret the results of the evaluation metrics in conjunction with a visual inspection of the generated captions to assess the overall quality of the model.

VI. Bibliography - Reference citations

1. ResNet-101 model. (n.d.). PyTorch. Retrieved April 30, 2023, from https://pytorch.org/docs/stable/torchvision/models.html#torchvision.models.resnet101

2. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog. Retrieved April 30, 2023, from https://openai.com/blog/better-language-models/

3. Anderson, P., Fernando, B., Johnson, M., & Gould, S. (2016). SPICE: Semantic Propositional Image Caption Evaluation. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 382–398). Springer. https://doi.org/10.1007/978-3-319-46475-6_23

4. Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (pp. 311–318). Association for Computational Linguistics. https://doi.org/10.3115/1073083.1073135

VII. Code and Data

https://github.com/Prajwal-S-Venkatesh/image-captioner