Northeastern University
Vancouver

Module 4: Using an “Off-the-shelf” Model

Applications of Artificial Intelligence

EAI 6010

Prepared by:

Vruditkumar Patel (002780055)

Presented to: RJ Munthali (PhD.)

Date: December 6th, 2023

Introduction

In the 4th module of this course, I am instructed to select any model other than image or text classification. After devoting a good amount of time to research on various real-world problems, I finally found one amazing topic. It is "Image Caption Generation". This topic is very interesting since we develop an ability into machine to recognize the content of the given image and describe it in a sentence. Hence, it involves computer vision and natural language processing. Itself it seems something difficult, and in the real world it is, because we merge two fields of deep learning to solve the problem.

As per the instructions, after doing extensive research and selecting the problem statement, I found a tutorial on this topic from Kaggle where this problem was solved by using a famous dataset named "COCO 2017". This dataset has around 200000 images and 80 object categories. This dataset is mainly used in object detection. The dataset has captions stored in JSON file. In addition, the author has used a deep learning architecture that involves "**Vision Transformer**" as an **Encoder** and "**GPT-2**" as a **Decoder**. Furthermore, "**Seq2SeqTrainer**" from the hugging face was used for fine tuning the model. The link to the original article is given as a first reference and reference for the dataset that I used in second reference in the reference section at the end of the document.

Which dataset I have used?

Now after selecting the problem, I could find some related datasets on many open-source platforms that includes Flickr8k, Flickr30k, UIT-ViLC, Open Images V7 etc. Among all these, by considering the limitation of my machine and other available online resources, I decided to go ahead with Flickr8K from Kaggle. This is because, It is relatively smaller than others and it has ideal structure in which images and its corresponding images are stored. There are a folder named "Images" and a text file named "Caption.txt". "Images" stores all the images and "Caption.txt" holds the captions of the images.

	image	caption
0	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	A girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a woo...

Figure 1: Overview of the data-structure.

NUID: 002780055

What changes have I made?

As I described earlier, the dataset was well-structured thus I did not need to alter any feature of the dataset. However, as name suggest, It has around 8000 images. Initially, I could not anticipate the complexity associated with such a huge dataset and therefore, I tried to train the model with 80% of the data. But to complete just 10% of the training, it took around 1.5 hours. Thus, I decided to interrupt the execution and reduced the size to just 22% of the original data. In this way, I could manage to utilize the colab workplace and got the things done!

```
df=  pd.read_csv("/content/drive/MyDrive/Module 4/Assignment/Flickr/captions.txt")
df_sampled = df.sample(frac=0.22, random_state=config.SEED)
train_df, val_df = train_test_split(df_sampled, test_size=0.1, random_state=config.SEED)
```

Figure 2: Reducing the dataset's size.

Difficulties I encountered and respective solutions.

As we know, Artificial Intelligence is ever changing field. Thus, on a daily basis we get new concepts or updates. Same way, I was required to utilize certain packages like transformers from the torch-vision. In addition, I needed to utilize many other packages. The problem I faced was related to compatibility of transformer package. I tried a lot and did some extensive research on open source community then I came to know about the compatibility issue associated with transformer. From the Hugging face's official site, I came to know that I would require previous version of transformer. The version would be 4.17. After downgrading the version, I restarted the session. In this way I could solve this problem.

```
!pip install datasets
!pip install transformers==4.17
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (9.0.0)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/dist-packages (from datasets) (0.6)
Requirement already satisfied: dill<0.3.8,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.3.7)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from datasets) (3.4.1)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.10/dist-packages (from datasets) (0.70.15)
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.9.1)
Requirement already satisfied: huggingface-hub>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.19.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
```

Figure 3: Downgrading Transformer library.

Final Results.

After struggling a bit, finally I trained the model. The model learned from a set of 8010 examples over three training sessions. I checked how well it did using some measures, like Rouge2 Precision, Recall, and F-measure, which tell us how good the model is at making image captions compared to correct captions. These measures improved as the model practiced more, showing it got better over time. The training loss, a measure of how well the model learned, went down, reaching a final value of 2.635900. A lower training loss usually means the model understood the training data well. The validation loss, which helps us to see how well the model works on new, unseen data, also decreased from 2.8547 to 2.6257 between the first and third training sessions. This suggests the model got better at making accurate captions not just for the examples it saw during training but also for new ones. In conclusion, the model learned and improved during training, making it good at creating high-quality image captions.

```
***** Running training *****
  Num examples = 8010
  Num Epochs = 3
  Instantaneous batch size per device = 8
  Total train batch size (w. parallel, distributed & accumulation) = 8
  Gradient Accumulation steps = 1
  Total optimization steps = 3006
```

[3006/3006 1:42:36, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Rouge2 Precision	Rouge2 Recall	Rouge2 Fmeasure
1	No log	2.854749	0.020600	0.192400	0.035900
2	3.475100	2.651174	0.023700	0.256800	0.042800
3	2.635900	2.625706	0.025200	0.262500	0.045300

```
***** Running Evaluation *****
  Num examples = 890
  Batch size = 8
***** Running Evaluation *****
  Num examples = 890
  Batch size = 8
Saving model checkpoint to VIT_large_gpt2/checkpoint-2048
Configuration saved in VIT_large_gpt2/checkpoint-2048/config.json
Model weights saved in VIT_large_gpt2/checkpoint-2048/pytorch_model.bin
Feature extractor saved in VIT_large_gpt2/checkpoint-2048/preprocessor_config.json
***** Running Evaluation *****
  Num examples = 890
  Batch size = 8
```

Figure 4: Training results.

At the end I checked its performance by passing certain images to generate the caption. I found that models can identify the context of the image successfully. However, it is not able to handle

the detailed scenario of the image. For example, sometimes though there is just one dog, it generates captions with two dogs. This is obvious since I did not utilize full data and trained the model on limited dataset with very limited resources. The performance can be further enhanced by using more data and epochs in the training process. Fine tuning can also improve efficiency. However, these things demand access to decent computing power.

```
img =  Image.open("/content/drive/MyDrive/Module 4/Assignment/Flickr/Images/1001773457_577c3a7d70.jpg").convert("RGB")
img
```

```
[ ] generated_caption = tokenizer.decode(model.generate(feature_extractor(img, return_tensors="pt").pixel_values.to("cuda"))[0])
    print(generated_caption[:85])
```

```
<|endoftext|>A black and white dog is running in the snow... a black and brown dog is
```

Figure 5: Caption for fighting dogs.

As we can see, my model has identified that there are two dogs. One is white and the second one is black. But is that dogs are running on the snow. So here it is failed to recognize the street road.

```
[ ]  generated_caption = tokenizer.decode(model.generate(feature_extractor(img, return_tensors="pt").pixel_values.to("cuda"))[0])
     print(generated_caption[:150])

     <|endoftext|>A man wearing a white shirt and blue jeans is standing in front of a brick wall. A man wearing sunglasses is standing next to a building.
```

Figure 6: Testing my own image.

When I gave my image to the model. It surprisingly performed well. Just it failed to recognize the color of my outfit. It says I am wearing a white shirt which is wrong. However, it identified a man wearing sunglasses is standing in front of brick wall which is absolutely true.

So, in this way utilizing transfer learning concepts, and combination of computer vision (Vision Transformer) and natural language processing (GPT-2), I could develop a decent caption generator.

Reference

Shreydan. (2023, October 2). 📸🖍️ VisIonGPT2 Image captioning | PyTorch 🔥. Kaggle. https://www.kaggle.com/code/shreydan/visiongpt2-image-captioning-pytorch#Predictions

Flickr 8K dataset. (2020, April 27). Kaggle. https://www.kaggle.com/datasets/adityajn105/flickr8k

Papers with Code - Machine Learning Datasets. (n.d.). Retrieved December 7, 2023, from https://paperswithcode.com/datasets?task=image-captioning

Gupta, S. (2023, September 5). Step by Step Guide to Build Image Caption Generator using Deep Learning. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/

Appendix:

```
!pip install datasets
!pip install transformers==4.17

from IPython.display import clear_output
!pip install rouge_score -q
!pip install deep-phonemizer -q
clear_output()

import os

import datasets
import numpy as np
import pandas as pd
from PIL import Image
from pathlib import Path
from tqdm.auto import tqdm
import multiprocessing as mp
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import io, transforms
from torch.utils.data import Dataset, DataLoader, random_split

from transformers import Seq2SeqTrainer ,Seq2SeqTrainingArguments
from transformers import VisionEncoderDecoderModel , ViTFeatureExtractor
from transformers import AutoTokenizer ,  GPT2Config , default_data_collator


if torch.cuda.is_available():

    device = torch.device("cuda")

    print('There are %d GPU(s) available.' % torch.cuda.device_count())

    print('I will use the GPU:', torch.cuda.get_device_name(0))

else:
    print('No GPU available, Let me go with CPU.')
    device = torch.device("cpu")
```

```
"""Here, I am going to create a class named "config" which will store all
the  variables which will be used later on. This is good idea to write a code in
professional way."""

os.environ["WANDB_DISABLED"] = "true"
class config :
    ENCODER = "google/vit-base-patch16-224"
    DECODER = "gpt2"
    TRAIN_BATCH_SIZE = 8
    VAL_BATCH_SIZE = 8
    VAL_EPOCHS = 1
    LR = 5e-5
    SEED = 42
    MAX_LEN = 128
    SUMMARY_LEN = 20
    WEIGHT_DECAY = 0.01
    MEAN = (0.485, 0.456, 0.406)
    STD = (0.229, 0.224, 0.225)
    TRAIN_PCT = 0.95
    NUM_WORKERS = mp.cpu_count()
    EPOCHS = 3
    IMG_SIZE = (224,224)
    LABEL_MASK = -100
    TOP_K = 1000
    TOP_P = 0.95

"""The below function named "build_inputs_with_special_tokens" will create a
special tokens while tokenize the captions."""

def build_inputs_with_special_tokens(self, token_ids_0, token_ids_1=None):
    outputs = [self.bos_token_id] + token_ids_0 + [self.eos_token_id]
    return outputs
AutoTokenizer.build_inputs_with_special_tokens = build_inputs_with_special_tokens

"""Here I am defining the rouge function.

**What is rouge matrix?**

ROUGE is an evaluation metric commonly used in natural language processing (NLP)
and specifically in the context of text summarization and machine translation.
ROUGE stands for "Recall-Oriented Understudy for Gisting Evaluation." It is
designed to measure the quality of summaries by comparing them to reference
summaries
"""
```

```
rouge = datasets.load_metric("rouge")

def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions

    # all unnecessary tokens are removed
    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = tokenizer.pad_token_id
    label_str = tokenizer.batch_decode(labels_ids, skip_special_tokens=True)

    rouge_output = rouge.compute(predictions=pred_str, references=label_str,
rouge_types=["rouge2"])["rouge2"].mid

    return {
        "rouge2_precision": round(rouge_output.precision, 4),
        "rouge2_recall": round(rouge_output.recall, 4),
        "rouge2_fmeasure": round(rouge_output.fmeasure, 4),
    }

"""**Now** as I need to generate the Captions, It will require NLP for training
purpose. Thus below is the function for feature extracttor. Here, I will use
ViTFeatureExtractor for tokenizing the captions."""

feature_extractor = ViTFeatureExtractor.from_pretrained(config.ENCODER)
tokenizer = AutoTokenizer.from_pretrained(config.DECODER)
tokenizer.pad_token = tokenizer.unk_token

"""Here, I will define the transformer for resizing the image. It will also
normalize and convert image into tensor.

Apart from it, As Flickr8k has around 8k images, My laptop is not capable enough
to handle such a large dataset. Even Google colab also provide GPU access for few
hours. After that we can not use GPUs. Thus, I decided to reduce the size of the
dataset. Here I am using df_sample function and getting only 22% of original
dataset.
"""

custom_transform = transforms.Compose(
    [
        transforms.Resize(config.IMG_SIZE),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=0.5,
```

```
            std=0.5
        )
    ]
)


df=  pd.read_csv("/content/drive/MyDrive/Module
4/Assignment/Flickr/captions.txt")
df_sampled = df.sample(frac=0.22, random_state=config.SEED)
train_df, val_df = train_test_split(df_sampled, test_size=0.1,
random_state=config.SEED)

df.head()

print(len(train_df), len(val_df))

"""Below is the function for reading the image from the directory. In addition it
will transform image using above defined tranformer function. Further transformed
image will pass through feature extraction to extract the pixel value from the
image.
Captions will be loaded and tokenized.It will then go for padding for maximum
lenght. After all, images and tokenized captions are returned.

"""

class ImgDataset(Dataset):
    def __init__(self, df,root_dir,tokenizer,feature_extractor, transform =
None):
        self.df = df
        self.transform = custom_transform
        self.root_dir = root_dir
        self.tokenizer= tokenizer
        self.feature_extractor = feature_extractor
        self.max_length = 50
    def __len__(self,):
        return len(self.df)
    def __getitem__(self,idx):
        caption = self.df.caption.iloc[idx]
        image = self.df.image.iloc[idx]
        img_path = os.path.join(self.root_dir , image)
        img = Image.open(img_path).convert("RGB")

        if self.transform is not None:
            img= self.transform(img)
```

```
        pixel_values = self.feature_extractor(img,
return_tensors="pt").pixel_values
        captions = self.tokenizer(caption,
                                  padding='max_length',
                                  max_length=self.max_length).input_ids
        captions = [caption if caption != self.tokenizer.pad_token_id else -100
for caption in captions]
        encoding = {"pixel_values": pixel_values.squeeze(), "labels":
torch.tensor(captions)}
        return encoding

train_dataset = ImgDataset(train_df, root_dir = "/content/drive/MyDrive/Module
4/Assignment/Flickr/Images",tokenizer=tokenizer,feature_extractor =
feature_extractor ,transform = transforms)
val_dataset = ImgDataset(val_df , root_dir = "/content/drive/MyDrive/Module
4/Assignment/Flickr/Images",tokenizer=tokenizer,feature_extractor =
feature_extractor , transform  = transforms)

len(train_dataset)

"""Below is a vision tranformer. The Vision Transformer, or ViT, is a way of
sorting images into different categories. It uses a structure similar to a
Transformer on parts of the image called patches. The image is divided into
patches, and each patch is turned into a sequence of numbers. These sequences,
along with their positions, go through a Transformer-like process. To decide on
the category, a common method is used, which involves adding a special
"classification token" to the sequence."""

model = VisionEncoderDecoderModel.from_encoder_decoder_pretrained(config.ENCODER,
config.DECODER)

"""Now for the NLP, **GPT-2** is a type of transformer model that has been
trained on a large amount of English text in a special way. It learns by
predicting the next word in a sequence of words. The input is a series of words,
and the goal is to predict the same series with each word shifted one position to
the right. The model uses a mask to only consider information from earlier words,
not ones that come later.

This process helps the model understand and represent the structure of the
English language. The knowledge it gains can be useful for our task.
"""

model.config.decoder_start_token_id = tokenizer.cls_token_id
model.config.pad_token_id = tokenizer.pad_token_id
# making sure vocab size is set correctly
```

```
model.config.vocab_size = model.config.decoder.vocab_size
# setting beam search parameters
model.config.eos_token_id = tokenizer.sep_token_id
model.config.decoder_start_token_id = tokenizer.bos_token_id
model.config.max_length = 128
model.config.early_stopping = True
model.config.no_repeat_ngram_size = 3
model.config.length_penalty = 2.0
model.config.num_beams = 4

"""From the below cell, training will be perfromed. Here Seq2SeqTrainer is used
from hugging face."""

training_args = Seq2SeqTrainingArguments(
    output_dir='VIT_large_gpt2',
    per_device_train_batch_size=config.TRAIN_BATCH_SIZE,
    per_device_eval_batch_size=config.VAL_BATCH_SIZE,
    predict_with_generate=True,
    evaluation_strategy="epoch",
    do_train=True,
    do_eval=True,
    logging_steps=1024,
    save_steps=2048,
    warmup_steps=1024,
    learning_rate = 5e-5,
    #max_steps=1500, # delete for full training
    num_train_epochs = config.EPOCHS, #TRAIN_EPOCHS
    overwrite_output_dir=True,
    save_total_limit=1,
)

# instantiating trainer
trainer = Seq2SeqTrainer(
    tokenizer=feature_extractor,
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=default_data_collator,
)
trainer.train()

ainer.save_model('VIT_large_gpt2')
```

```
img =  Image.open("/content/drive/MyDrive/Module
4/Assignment/Flickr/Images/1001773457_577c3a7d70.jpg").convert("RGB")
img

generated_caption = tokenizer.decode(model.generate(feature_extractor(img,
return_tensors="pt").pixel_values.to("cuda"))[0])
print(generated_caption[:85])

img =  Image.open("/content/drive/MyDrive/Module
4/Assignment/Flickr/Images/1000268201_693b08cb0e.jpg").convert("RGB")
img

generated_caption = tokenizer.decode(model.generate(feature_extractor(img,
return_tensors="pt").pixel_values.to("cuda"))[0])
print(generated_caption[:120])

img =  Image.open("/content/drive/MyDrive/Module
4/Assignment/Flickr/Images/1012212859_01547e3f17.jpg").convert("RGB")
img

generated_caption = tokenizer.decode(model.generate(feature_extractor(img,
return_tensors="pt").pixel_values.to("cuda"))[0])
print(generated_caption[:120])

img = Image.open("/content/drive/MyDrive/Module
4/Assignment/Flickr/vdp4.jpg").convert("RGB")
img

generated_caption = tokenizer.decode(model.generate(feature_extractor(img,
return_tensors="pt").pixel_values.to("cuda"))[0])
print(generated_caption[:150])

"""# **Conclusion**

As per the above trials, I can say that my model is not acting perfecltly,
However, with relatively limited resources, the output seems satisfactory. Even I
tried it by uploading my own image, too.
"""
```