George Washington University

Data Science Capstone

DATS 6501

Spring 2023

Instructor: Dr. Edwin Lo

Image Caption Generator

Author: Nusrat Nawshin

Submission Date: May 5, 2023

Contents

1. Introduction:

People with weak eyesight or no eyesight find it challenging to stay aware of their surroundings. A system that provides a brief synopsis of the visuals around them would be very helpful to them and remove their accessibility constraints. With a goal of an improved assistive system, my caption generator is going to employ image processing and deep learning techniques to identify objects in the images, and then natural language processing (NLP) will be used for transforming that information into a relevant sentence that describes the entire image and the scenario. In this study, I have used Pytorch framework and proposed an Image Captioning Model by implementing image feature extraction using Convolution Neural Network (CNN), caption tokenization through Bidirectional Encoder Representations from Transformers (BERT) model and transformer model with multi-head attention mechanism.

2. Dataset & Data Preparation

The dataset I am using is called "VizWiz-Captions" dataset collected from a competition from vizwiz.org consists of 39,181 images originating from people who are blind that are each paired with 5 captions [1]. The test set doesn't have the annotations provided so I have decided to make the provided validation set as my test set for this project. The annotations files were stored in a json file in the original dataset. I have removed unnecessary keys and created train and validation dataset with the image names and their captions and stored them in CSV files.


```
images = [image]
image = {
    "file_name": "VizWiz_train_00023410.jpg",
    "id": 23410
    "text_detected": true
}
annotations = [annotation]
annotation = {
    "image_id": 23410,
    "id": 117050,
    "caption": "A plastic rewards card lying face down
on the floor."
    "is_rejected": false,
    "is_precanned": false,
    "text_detected": true
}
```


Figure 1: Raw annotation files in json format

Name	Caption
VizWiz_train_00000000.jpg	ITS IS A BASIL LEAVES CONTAINER ITS CONTAINS THE NET WEIGHT TOO.
VizWiz_train_00000000.jpg	A green and white plastic condiment bottle containing Basil leaves.
VizWiz_train_00000000.jpg	Quality issues are too severe to recognize visual content.
VizWiz_train_00000000.jpg	A bottle of spices in a plastic container laying on a surface.
VizWiz_train_00000000.jpg	some basil leaves in a container on a counter

Figure 2: Final csv file's top 5 entries

Further, I have divided my validation set using a 50-50 split and left the provided train set completely for training. The final dataset has the following numbers of images and captions:

Training Set: 23,431 images & 117,155 captions.

Validation Set: 3,875 images & 96,876 captions.

Test Set: 3,875 images & 96,876 captions.

3. Data Preprocessing

Image Resizing: The images in the dataset had varied sizes so primarily I have resized it to make all images uniform as in Neural Network takes vector as an input, the images should have same dimensions. I have resized the images to [3 x 224 x 224] or [3 x 299 x 299] as required by the pretrained model input dimensions. I have used 'torchvision.transforms' package to resize the images.

Image Augmentation: Using "torchvision" package's "transformers.Compose()" method I implemented image augmentations on batches of images and transformed it to tensor type data. It generates images with the following augmentation parameters.

- transforms.RandomRotation(degrees=45),
- transforms.RandomHorizontalFlip(),
- transforms.RandomPerspective(),
- transforms.RandomVerticalFlip(),
- transforms.RandomInvert(0.5),
- transforms.RandomAdjustSharpness(0.5),
- transforms.ToTensor()

4. Model Architecture

The complete model architecture consists of three models. First, a convolution model extracts image features and stores all images features in a pickle file and another model tokenizes the captions and stores that in another pickle file. These two files are input files for the main transformer model which generates

the captions. For image feature extractions I have trained and tested my model with different pretrained models and using forward hook technique retuned the image features from the convolution model. I have used ResNet18, ResNet50, VGG16, InceptionV3, Xception, SSDLite pretrained models. For caption tokenization I have used Bag of Words (BoW) or BERT tokenizers. Here is an overview of the final whole model in the following figure 3.

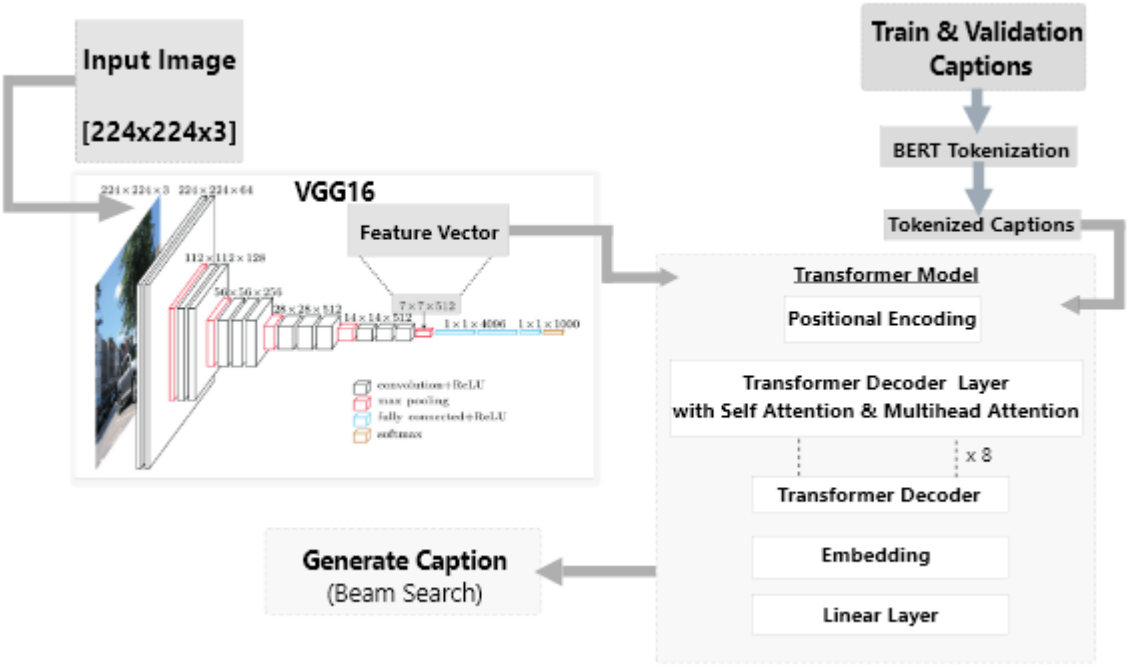

Figure 3: Complete model architecture and flowchart

I. Image Feature Extraction

The first step in creating features from an image is to build a vector with a specific length that effectively captures the information in the image. For image feature extraction I have used different pretrained models and returned the convolution models output as stored it in a pickle file. The models I have used are ResNet18, ResNet50, VGG16, InceptionV3, Xception, SSDLite. For all the models, I have used pretrained weight and using forward hook I returned the convolution layers output as my task does not need to classify the image but get the feature vector only. For instance, VGG16 has three modules: - 'features', 'avgpool', and 'classifier'. The ‘features’ module has the CNN model and ‘avgpool’ returns 2D adaptive average pooling over the CNN modules output. Finally, the ‘classifier’ module uses fully connected layers for classification, but this layer is not required by my project objective. Using the following code, I have implemented forward hook on the models. This example is for VGG16 which returns embedding vector of size [1, 512, 7, 7].

```
vgg16 = torchvision.models.vgg16(weights=VGG16_Weights.IMAGENET1K_V1).to(device)
vgg16.eval()
print(list(vgg16._modules))
vgg16_features = vgg16._modules.get('avgpool').to(device)

def get_vector(t_img):
    t_img = Variable(t_img)
    my_embedding = torch.zeros(1, 512, 7, 7) #VGG16

    def copy_data(model, input, output):
        my_embedding.copy_(output.data)

    handle = vgg16_features.register_forward_hook(copy_data)
    vgg16(t_img)

    handle.remove()
    return my_embedding

extract_img_feature_train = {}
print("Extracting features from Train set:")
for imgs,image_name in tqdm(train_image_dataloader):
    t_img = imgs.to(device)
    embdg = get_vector(t_img)
    extract_img_feature_train[image_name[0]] = embdg #VGG16
```

Figure 4: Forward hooking on VGG16 model

Other models, that return different dimensions of embedded feature vector I had to further implement another CNN layer or permute function to match the vector size of [1, 512, 7, 7] as more than arises CUDA Memory Error.

Model	Total Convolution Layers	Pretrained Weight	Extracted Embedding Size
ResNet18	17	ImageNet-1k	[1, 512, 7, 7]
ResNet50	48	ImageNet-1k	[1, 2048, 7, 7]
VGG16	13	ImageNet-1k	[1, 512, 7, 7]
InceptionV3	94	ImageNet-1k	[1, 8, 8, 2048]
Xception	36	ImageNet-1k	[1, 3, 3, 2048]
SSDLite	6	COCO_V1	[1, 480, 10, 10]

Table 1: Pretrained models

II. Caption Tokenization

The caption tokenization steps included text cleaning and converting words to ids using either bag-of-words (BoW) or BERT method. The text cleaning steps included separating joined words using ‘wordninja’ package from PyPI [2]. Next, I removed single characters, punctuation and lowered down all the alphabets in the captions. After that I converted words to ids using BoW or BERT depending on different model trails. After tokenizing, all the training and validation set captions were stored in pickle files.

Bag of Words

The bag-of-words model is a condensed representation used in information retrieval (IR) and natural language processing (NLP). This approach maintains multiplicity while ignoring syntax and even word order, and it represents a text—for example, a sentence or document—as the collection of its words [3].

For this technique I used Counter package to create a vocabulary of words. There were a total of 19008 tokens in this vocabulary.

BERT

Bidirectional Encoder Representations from Transformers (BERT) employs a tokenizer known as “WordPiece”. It relies on separating words into complete forms i.e., one word becomes one token or into segments of words, where one word may be fragmented into several tokens [4]. All the captions of the dataset are tokenized using pretrained BERT tokenizer and its vocabulary. BERT uses a special token to indicate the sentence initialization, ending, and padding. For tokenization, first [CLS] token as the initialization and [SEP] token as the end of sentence was appended. For padding, the [PAD] token is used. For uniformity and space utilization I have used max sequence length as 45 as only a few captions have more than 45 words. There are a total of 28996 tokens in BERT vocabulary.

For example, the original text is:

“ITS IS A BASIL LEAVES CONTAINER ITS CONTAINS THE NET WEIGHT TOO”

After using BERT tokenizer:

"['[CLS]', 'its', 'is', 'basil', 'leaves', 'container', 'its', 'contains', 'the', 'net', 'weight', 'too', '[SEP]', '[PAD]']"

After converting tokens to IDs:

"[101, 1157, 1110, 171, 17506, 1233, 2972, 12461, 1157, 2515, 1103, 5795, 2841, 1315, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]"

III. Transformer Model

For the main language model, I have used a transformer model which is a sequence-to-sequence architecture with multi-head attention mechanism and transformer decoder layers. The attention mechanism is a complex cognitive ability that human beings possess. People can choose to deliberately overlook part of the primary information while disregarding other secondary information when they obtain information. This ability of the selection process is known as attention. The attention mechanism empowers the neural network to concentrate on a subset of its inputs and choose particular features [5]. The transformer layer I have used is based on the paper “Attention is all you need” [6] an innovative way to construct neural networks without using any CNN decoder layer. The custom transformer model built for this task consists of positional encoding, transformer decoder layer with 8 self-attention and 8 multi-head attention and 8 decoder layers. The positional encoding gives importance to the elements as transformer architecture dropped the recurrence process as a substitute for a multi-head self-attention mechanism. Basically, positional encoding creates a vector for each element representing its position regarding every other element in the sequence. An embedding layer is required before adding the positional encoding so that every element in the sequences is changed into a vector that is able to manipulate (instead of a fixed integer) [7]. The following figure shows the summary of the whole transformer model for the captioning names as ‘ImageCaptionModel’.

```
ImageCaptionModel(
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (TransformerDecoderLayer): TransformerDecoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (multihead_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
    (dropout3): Dropout(p=0.1, inplace=False)
  )
  (TransformerDecoder): TransformerDecoder(
    (layers): ModuleList(
      (0-7): 8 x TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (embedding): Embedding(28996, 512)
  (last_linear_layer): Linear(in_features=512, out_features=28996, bias=True)
)
```

Figure 5: Image Caption Model summary

5. Framework and optimizer

All the coding was done using Python and Pytorch framework as it is more dynamic and uses asynchronous execution to implement data parallelism. For the user interface I used Flask and Bootstrap framework to design and implement the website.

As the optimizer I used Adam, Adamax or SGD. The final model is optimized with the Adam optimization technique with “ReduceLROnPlateau” to accelerate learning by gradually slowing down the learning rate progress. The initialized learning rate was 0.00001.

6. Evaluation Metrices

ROUGE 2 F Measure: ROUGE-2 refers to the bigram overlap between two sentences.

ROUGE L F Measure: ROUGE L uses the longest common sequence to determine the longest matched string of words. Utilizing this method has the benefit of only requiring in-sequence matches that accurately reflect sentence level word order, rather than consecutive matches [8].

Bilingual Evaluation Understudy (BLEU): Counting matching n-grams by measuring their correlation to reference translations. It doesn’t consider semantic meaning [9]. In this study I have used BLEU-1 metric.

Meteor: Meteor is unigram matching between two sentences, but it also considers additional information such as synonyms, word forms, and sentence structure [9].

7. All Attempted Models

Throughout the course I have attempted several models and their description are noted below:

ResNet18: ResNet18 with Image Augmentation, 8 transformer attention heads, 6 decoder layers, activation function = 'ReLU', BoW caption tokenization, Adam optimizer with learning rate 0.00001.

ResNet50: ResNet50 with Image Augmentation, 8 transformer attention heads, 6 decoder layers, activation function = 'ReLU', BoW caption tokenization, Adam optimizer with learning rate 0.0001.

VGG16 1: VGG16 with Image Augmentation, 16 transformer attention heads, 8 decoder layers, activation function = 'ReLU', BoW caption tokenization, Adam optimizer with learning rate 0.00001.

VGG16 2: VGG16 with Image Augmentation, 16 transformer attention heads, 8 decoder layers, activation function = 'GeLU', BoW caption tokenization, Adam optimizer with learning rate 0.00001.

VGG16 3: VGG16 without Image Augmentation, 16 transformer attention heads, 6 decoder layers, activation function = 'ReLU', BoW caption tokenization, Adam optimizer with learning rate 0.00001.

VGG16 4: VGG16 without Image Augmentation, 16 transformer attention heads, 8 decoder layers, activation function = 'ReLU', BoW caption tokenization, Adam optimizer with learning rate 0.00001.

VGG16 5: VGG16 without Image Augmentation, 16 transformer attention heads, 8 decoder layers, activation function = 'ReLU', BERT caption tokenization, Adam optimizer with learning rate 0.00001.

VGG16 6: VGG16 without Image Augmentation, 16 transformer attention heads, 6 decoder layers, activation function = 'ReLU', BERT caption tokenization, Adam optimizer with learning rate 0.00001.

VGG16 7: VGG16 without Image Augmentation, 16 transformer attention heads, 6 decoder layers, activation function = 'ReLU', BERT caption tokenization, SGD optimizer with learning rate 0.00001.

Inception 1: InceptionV3 without Image Augmentation, 16 transformer attention heads, 6 decoder layers, activation function = 'ReLU', BERT caption tokenization, Adam optimizer with learning rate 0.00001.

Inception 2: InceptionV3 without Image Augmentation, 16 transformer attention heads, 8 decoder layers, activation function = 'ReLU', BERT caption tokenization, Adam optimizer with learning rate 0.00001.

Xception: Xception with Image Augmentation, 16 transformer attention heads, 8 decoder layers, activation function = 'ReLU', BoW caption tokenization, Adam optimizer with learning rate 0.00001.

SSDLite: SSDLite without Image Augmentation, 16 transformer attention heads, 8 decoder layers, activation function = 'ReLU', BERT caption tokenization, Adam optimizer with learning rate 0.00001.

8. Results

Model	Epoch	Best Model's Loss	ROUGE-2 F Measure		ROUGE-L F Measure		BLEU		Meteor	
			Valid	Test	Valid	Test	Valid	Test	Valid	Test
ResNet18	15	4.602	0.195	0.292	0.351	0.355	0.001	0.0011	0.023	0.023
ResNet50	18	3.285	0.278	0.304	0.413	0.439	0.001	0.0011	0.023	0.023
VGG16 1	22	3.237	0.2492	0.2902	0.38	0.4173	0.0012	0.0010	0.0234	0.0235
VGG16 2	22	3.235	0.2526	0.2923	0.381	0.4182	0.0012	0.0009	0.0234	0.0235
VGG16 3	22	3.152	0.2667	0.3016	0.4055	0.4368	0.0012	0.0010	0.0234	0.0235
VGG16 4	23	3.147	0.2647	0.2954	0.4065	0.4340	0.0011	0.0009	0.0234	0.0235
VGG16 5	**22**	**2.988**	**0.2705**	**0.3011**	**0.4102**	**0.4342**	**0.0013**	**0.0010**	**0.0234**	**0.0235**
VGG16 6	24	2.986	0.2680	0.2997	0.4016	0.4312	0.0012	0.0010	0.0234	0.0235
VGG16 7	7	3.077	0.2616	0.2963	0.3747	0.4085	0.0012	0.0010	0.0234	0.0235
Inception 1	28	2.947	0.2658	0.3011	0.3983	0.4304	0.0012	0.0010	0.0234	0.0235
Inception 2	32	2.994	0.2619	0.2970	0.3962	0.4307	0.0012	0.0010	0.0234	0.0235
Xception	24	3.107	0.268	0.302	0.401	0.432	0.011	0.001	0.023	0.023
SSDLite	39	3.015	0.206	--	0.332	--	0.0011	--	0.023	--

Table 2: All model's results

Among all the models, VGG16-5 model has overall better performance also that is the only model that could improve BLUE score at the validation set.

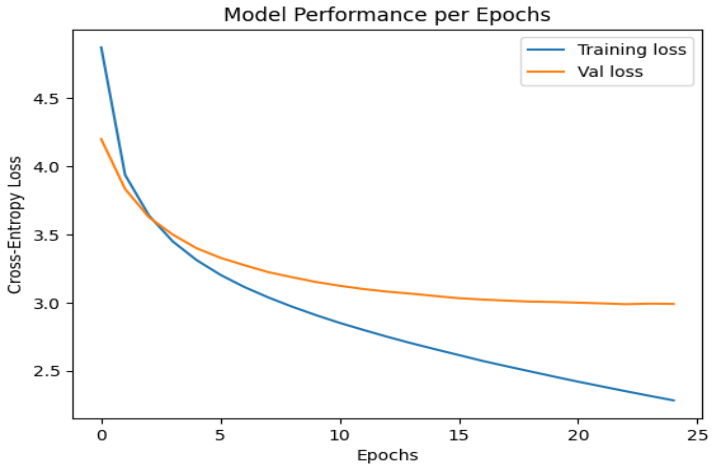

Figure 6: Model VGG16-5 performance over the epochs

9. Predictions

Using the best model, I have generated captions for the validation set and the test set. Here are a few samples of the generated captions.

Actual Captions:

- A laptop screen with system dialog on the screen
- The corner of a black computer with type of the screen and white type on the keyboard
- Partial view of laptop showing upper right portion of keyboard starting with F 8 key all the way to the delete key and the screen is the right hand portion showing approximately half of the screen
- A black laptop with the computer's specs on the screen
- Part of the keyboard and screen on a laptop

Predicted Caption:

laptop screen showing the keyboard and screen

Actual Captions:

- A plastic of Germ-x hand sanitizer with 20% more bonus
- A bottle of germ-x is on a counter
- a plastic bottle of germ-X hand sanitizer with 20% more bonus
- A bottle of germ-x hand sanitizer that has a bonus of 20% more
- Large pump style bottle of Germ-x brand hand sanitizer

Predicted Caption:

bottle of hand sanitizer is on the table

Actual Captions:

- imagine how you would describe this image on the phone to a friend
- A small white jar with a lid on top.
- A white ceramic pot or sugar bowl with a lid on it, standing on a counter top.
- a granite counter top with a tea cup on it
- A ceramic container with lid, on the counter top, from above.

Predicted Caption:

white crock pot is on marble counter top

Actual Captions:

- a white towel hanging on the door of a shower stall
- a white towel hanging on a shower rod
- Long off white towel hanging over a towel bar connected to shower door.
- A towel hangs on the rack in front of a shower door.
- A white towel hanging on the towel rack on the glass shower door.

Predicted Caption:

white towel hanging on the wall in room

Actual Captions:

- A frozen cottage pie sitting on a counter top.
- A microwaveable dinner of something called Cottage Pie.
- A Cottage Pie frozen meal sitting on a wood surface.
- A frozen TV dinner box of cottage pie with the nutritional information and preparing instructions visible on the front.
- A boxed meal called cottage pie and on the front cover is a picture of what it should look like.

Predicted Caption:

frozen cottage pie sitting on wood grained surface

Figure 7: Model generated captions

10. User Interface

I have deployed my caption model in a website as the user interface. In the webpage, user can upload any picture and get the image captioned. User can also hear the caption and by clicking the 'say it again' button user can hear the caption again and again. The following figures have the screenshot of the webpage.

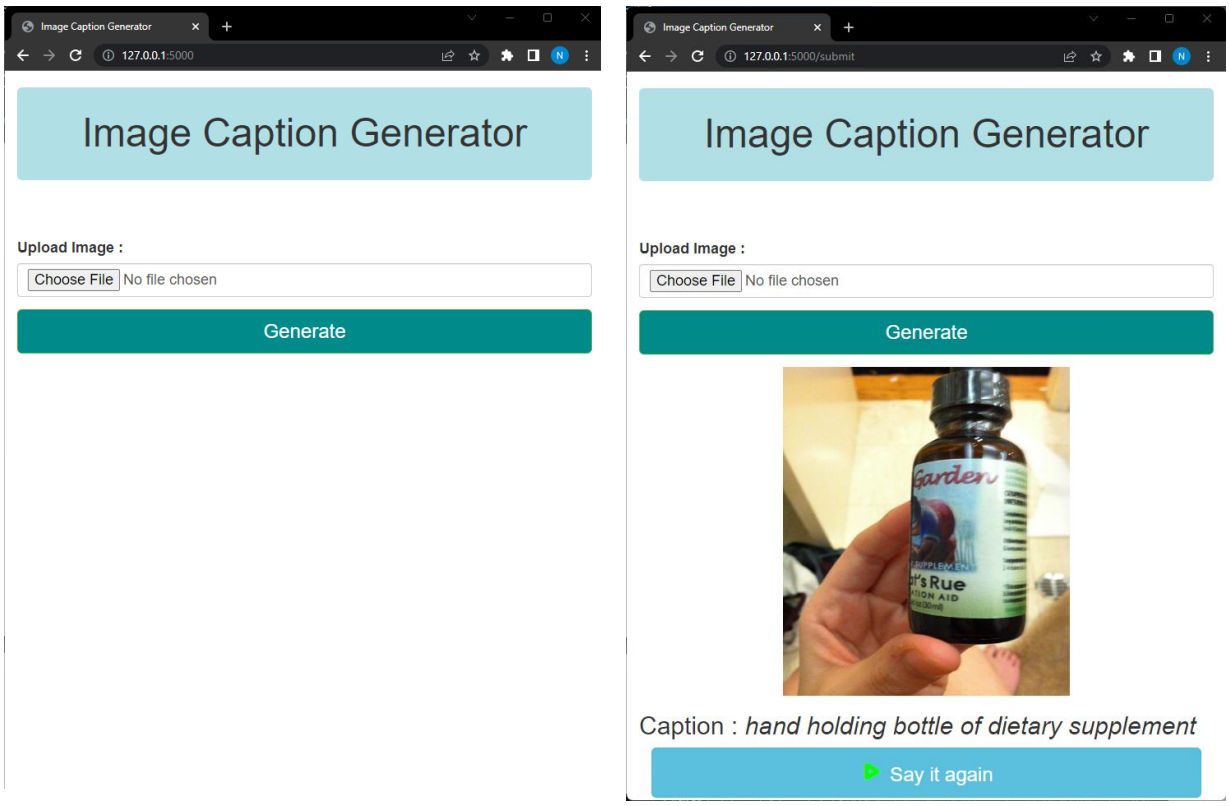

Figure 8: Screenshot of webpage before uploading image (left) and after uploading image with generated caption (right)

11. Conclusion

In this work, I developed a multi-model neural network that consequently learns to generate captions from the content of images. At first, I extracted features from the images using VGG16's convolution layers output and the captions were tokenized using BERT tokenizer. After that these two inputs were passed in a Transformer model with 16 multi-head attention and 8 transformer decoder layers. Model was trained till 22nd epochs with Adam optimizer with learning rate 0.00001. Finally, through beam search the model generates the captions. A scene graph or object detection mechanism may be added to the model to further improve it, however advanced techniques could not be implemented due to computational limitations. In addition, I developed a user interface that allows users to upload photos and generate captions along with audio speech.

12. References

[1] https://vizwiz.org/tasks-and-datasets/image-captioning/

[2] https://pypi.org/project/wordninja/

[3] https://en.wikipedia.org/wiki/Bag-of-words_model

[4] https://towardsdatascience.com/how-to-build-a-wordpiece-tokenizer-for-bert-f505d97dddbb#:~:text=BERT%20uses%20what%20is%20called,have%20multiple%20forms%20of%20words.

[5] https://www.analyticsvidhya.com/blog/2020/11/attention-mechanism-for-caption-generation/

[6] https://arxiv.org/abs/1706.03762

[7] https://towardsdatascience.com/a-detailed-guide-to-pytorchs-nn-transformer-module-c80afbc9ffb1

[8] Lin, Chin-Yew. “Rouge: A package for automatic evaluation of summaries”. Text summarization branches out. 2004

[9] https://ai.stackexchange.com/questions/35296/what-are-the-differences-between-bleu-and-meteor#:~:text=BLEU%20score%20is%20an%20average,the%20hypothesis%20and%20are%20correct.

Code GitHub Link: https://github.com/NusratNawshin/GWU_DS_Capstone