DLCV HW1 Report

R11943004 黃子青

Problem 1

1. Paper reading(3%)

Please read the paper " Visual Instruction Tuning" and briefly describe the important components (modules or techniques) of LLaVA.

The paper "Visual Instruction Tuning" introduces LLaVA (Large Language and Vision Assistant), which integrates multimodal vision and language understanding.
Here are the important components (modules or techniques) of LLaVA:

1.Vision Encoder:

A pre-trained CLIP (Contrastive Language–Image Pre-training) Vision Transformer is used as the vision encoder.
It extracts visual features from input images, transforming them into a high-dimensional feature space aligned with text representations.

2.Language Model:

The system employs a large pre-trained language model (LLM), such as LLaMA (Large Language Model Meta AI), to handle text-based reasoning and response generation.
The LLM serves as the backbone for the language processing tasks.

3.Visual Projection Layer:

A learnable linear projection layer maps the visual features from the vision encoder into the embedding space of the language model. This step ensures compatibility between the modalities.

4.Instruction Tuning:

The core of LLaVA is visual instruction tuning, where the system is fine-tuned on image-text-instruction-response datasets.
These datasets simulate user instructions and model responses, helping align the system to follow multimodal instructions effectively.

5.Multimodal Interaction:

The projected visual features are concatenated with the text embeddings, enabling the language model to reason jointly over visual and textual inputs.

2. **Prompt-text analysis (6%)**
   Please come up with two settings (different instructions or generation config). Compare and discuss their performances

I used two types of prompts for testing, which are as follows:

```
prompt1 = "USER: <image>\nProvide a brief caption of the given image ASSISTANT:"
prompt2 = "USER: <image>\nDescribe the image caption in a sentence ASSISTANT:"
```

Prompt1 result:

```
CIDEr: 1.1236827541154448 | CLIPScore: 0.7881488037109375
```

Prompt2 result:

```
CIDEr: 1.1454801865395143 | CLIPScore: 0.789749755859375
```

I think the reason why Prompt 2 produces better results is that it emphasizes "in a sentence," which encourages more focused and concise outputs.
If this emphasis is missing, the model often generates responses exceeding the max_new_tokens limit, which negatively impacts performance.

Problem 2:

1. Report your best setting and its corresponding CIDEr & CLIPScore on the validation data. Briefly introduce your method. (TA will reproduce this result) (5%)

Encoder Configuration:

Using the vit_giant_patch14_224_clip_laion2b model from timm as the encoder:

[1:] skips the first token (class token) and retains the rest.

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.model = timm.create_model('vit_giant_patch14_clip_224.laion2b', pretrained=True)

    def forward(self, x):
        out = self.model.forward_features(x)
        out = out[:, 1:, :]
        return out
```

Decoder Configuration:

```
        # Projection layer for visual features
        self.visual_projection = nn.Linear(1408, 1088)

        self.visual_projection2 = nn.Linear(1088, cfg.n_embd)

        # load checkpoint
        if self.cfg.checkpoint is not None:
            state_dict = torch.load(self.cfg.checkpoint)
            transposed = [ '.c_attn.weight', '.c_fc.weight', '.c_proj.weight' ]
            for key, value in state_dict.items():
                if any(key.endswith(w) for w in transposed):
                    state_dict[key] = value.t()
            self.transformer.load_state_dict(state_dict, strict=False)
            print('pretrain decoder loaded')


    def forward(self, x: Tensor, image_features: Tensor = None):
        if image_features is not None:
            visual_embed = self.visual_projection(image_features.float())
            visual_embed = self.visual_projection2(visual_embed)

            x = self.transformer.wte(x)
            visual_embed=self.ln(visual_embed)
            x = torch.cat((visual_embed, x), dim=1)

        x = torch.narrow(x, 1, 0, min(x.size(1), self.block_size))
        pos = torch.arange(x.size(1), dtype=torch.long, device=x.device).unsqueeze(0)
        x = x + self.transformer.wpe(pos)
        attn_list=[]
        for idx,block in enumerate(self.transformer.h):

            x = block(x)
            attn_list.append(block.atten)
        # x = self.transformer.h(x)
        x = self.transformer.ln_f(x)
        logits = self.lm_head(x)
```

n_layer = 12

head = 12

n_embd = 768

Use a two-layer MLP to map image features into the text feature space.

```
    # Projection layer for visual features
    self.visual_projection = nn.Linear(1408, 1088)

    self.visual_projection2 = nn.Linear(1088, cfg.n_embd)
```

Lora setting:

I added LoRA to the lm_head in the decoder and the MLP of each self-attention layer

```
self.mlp = nn.Sequential(collections.OrderedDict([
    ('c_fc', lora.Linear(cfg.n_embd, 4 * cfg.n_embd,r=16)),
    ('act', nn.GELU(approximate='tanh')),
    ('c_proj', lora.Linear(4 * cfg.n_embd, cfg.n_embd,r=16))
]))
```

```
self.lm_head = lora.Linear(cfg.n_embd, cfg.vocab_size, bias=False,r=16)
```

Training setting:

During training, only lora components and two-layer MLP were trained.

```
lora.mark_only_lora_as_trainable(model)
model.decoder.visual_projection.requires_grad_(True)
model.decoder.visual_projection2.requires_grad_(True)
```

optimizer = torch.optim.AdamW(trainable_params, lr=1e-4, weight_decay=0.01)
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, 0.9, last_epoch=-1, verbose='deprecated')

Inference setting:

After the fifth epoch, I perform inference on the validation data at the end of each training session using a greedy algorithm to autoregressively generate text, and I save the checkpoint of the trainable parameters that pass the baseline.

```
if epoch<5:
    continue
model.eval()
predictions = {}
print('predicting:')
with torch.no_grad():
    for idx, (image, filename) in enumerate(val_dataloader):
        image = image.to(device)
        encoder_out = model.encoder(image)  # 提取圖像特徵

        # 初始化生成的序列，並設置長度 50，以 50256 作為起始和填充 token
        current_caption = torch.full((1, 50), end_token, dtype=torch.int64, device=device)  # (1, 50)

        word = ""  # 用於存儲生成的文本
        for current_idx in range(49):  # 生成最多 49 個 token
            # 解碼器前向傳播
            output,attn_list = model.decoder(current_caption, encoder_out)  # output shape: (1, 50, vocab_size)

            # 在當前時間步的最後一個位置上獲取最可能的 token
            next_token = output.max(2, keepdim=False)[1][0, output.size(1)-gt.size(1)+current_idx].item()

            # 終止條件：遇到終止 token 或達到最大生成長度
            if next_token == end_token:
                break


            # 更新 `caption_test` 中的下一個位置的 token
            current_caption[0, current_idx + 1] = next_token

            # 將當前 token 解碼成文字，並拼接到 `word`
            word += tokenizer.decode([next_token])

        # 儲存生成的結果
        predictions[filename[0]] = word


        # print(f"word:{word}")

# 保存生成的結果到 JSON 文件
output_path = "pred8.json"
with open(output_path, "w") as f:
```

Best CIDEr & CLIPScore on the validation data:

```
CIDEr: 0.9550542427817708 | CLIPScore: 0.7310794067382812
```

2. Report 2 different attempts of LoRA setting (e.g. initialization, alpha, rank…) and their corresponding CIDEr & CLIPScore. (5%, each setting for 2.5%)

First setting(best setting): added LoRA to the lm_head in the decoder and the MLP of each self-attention layer ,and r=16

```
self.mlp = nn.Sequential(collections.OrderedDict([
    ('c_fc', lora.Linear(cfg.n_embd, 4 * cfg.n_embd,r=16)),
    ('act', nn.GELU(approximate='tanh')),
    ('c_proj', lora.Linear(4 * cfg.n_embd, cfg.n_embd,r=16))
]))
```

```
self.lm_head = lora.Linear(cfg.n_embd, cfg.vocab_size, bias=False,r=16)
```

Below is its training process

```
Epoch [8/10]
CIDEr: 0.9523092493630136 | CLIPScore: 0.7309225463867187
New best model saved with CLIP Score: 0.7309
 80%|                                                  | 8/10 [4:01:53<1:15:37, 2268.78s/it]P2_tra
ining_settingB.py:258: FutureWarning: `torch.cuda.amp.autocast(args ... )` is deprecated. Please use `torch.amp.autoc
ast('cuda', args ... )` instead.
  with autocast():
Epoch [9/10], Average Loss: 2.0502
```

Second setting: Only added LoRA to the MLP of each self-attention layer, and r remains 16

```
self.mlp = nn.Sequential(collections.OrderedDict([
    ('c_fc', lora.Linear(cfg.n_embd, 4 * cfg.n_embd,r=16)),
    ('act', nn.GELU(approximate='tanh')),
    ('c_proj', lora.Linear(4 * cfg.n_embd, cfg.n_embd,r=16))
]))
```

Below is its training process

```
Epoch [8/10], Average Loss: 2.1520
predicting:
Keys in predictions but not in gts: set()
Keys in gts but not in predictions: set()
PTBTokenizer tokenized 123146 tokens at 1114652.95 tokens per second.
PTBTokenizer tokenized 24149 tokens at 352247.55 tokens per second.
Epoch [8/10]
CIDEr: 0.9225287222075155 | CLIPScore: 0.7267036437988281
 80%|                                                  | 8/10 [4:01:33<1:07:43, 2031.88s/it]
P2_training_settingA.py:256: FutureWarning: `torch.cuda.amp.autocast(args ... )` is deprecated. Please use `torch.amp
.autocast('cuda', args ... )` instead.
  with autocast():
Epoch [9/10], Average Loss: 2.1324
```

We observed that after adding LoRA to the lm_head, the training converged faster, and the performance also improved significantly.
I believe the reason is that, compared to directly freezing the lm_head, adding LoRA to the lm_head allows for fine-tuning its parameters with minimal computation, enabling the model to learn more effectively.

Problem 3:

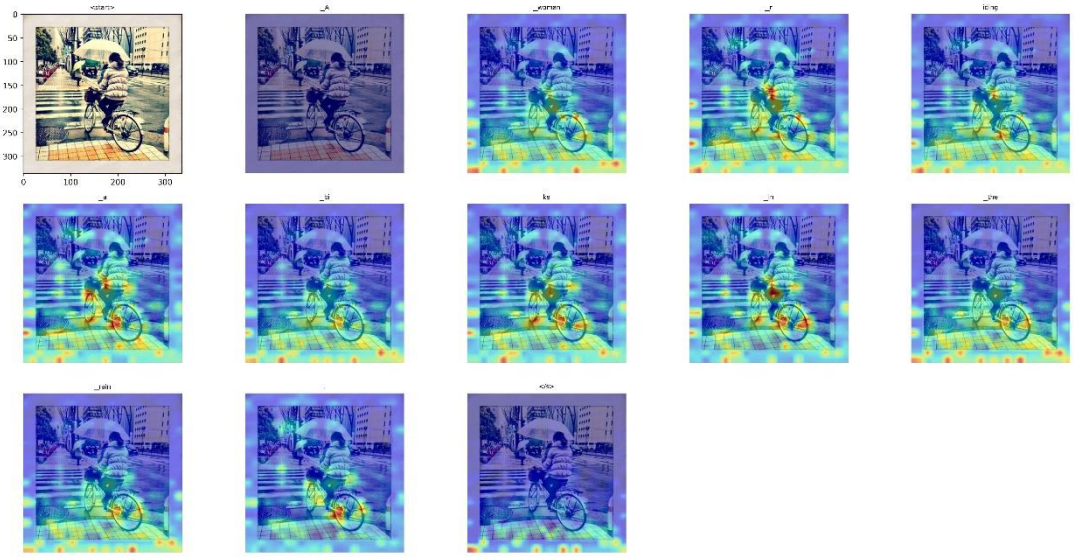
1. Given five test images ([p3_data/images/]), and please visualize the predicted caption and the corresponding series of attention maps in your report with the following template: (20%, each image for 2%, you need to visualize 5 images for both problem 1 & 2)

Problem1:

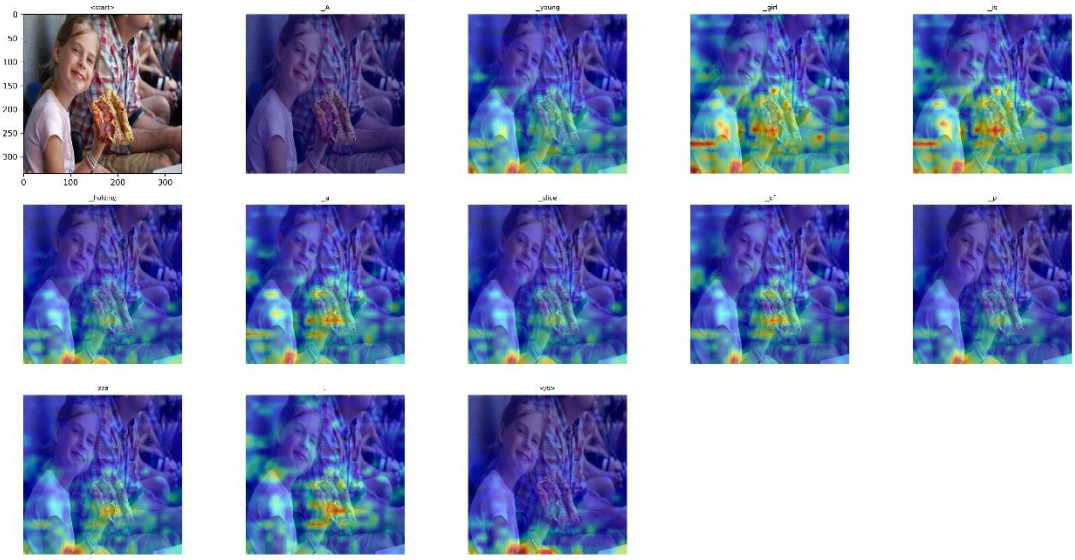
I observed that the visualization results of the last attention layer were not very distinct. Later, through observation, I found that the visualization results of the earlier layers were more noticeable. Therefore, I chose the first layer and averaged the results across 32 heads.

The predicted caption and the corresponding series of attention maps:

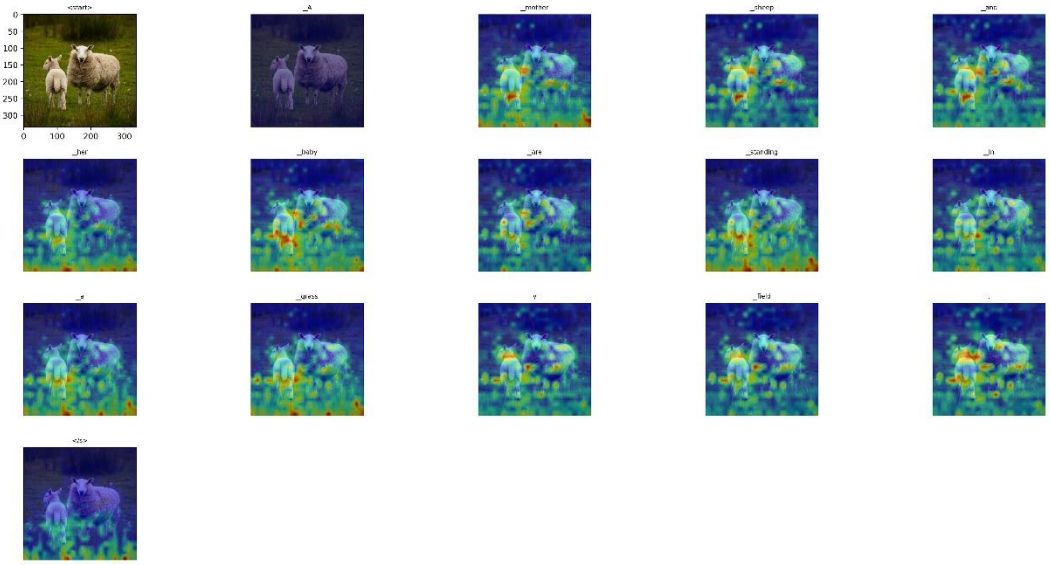
bike.jpg



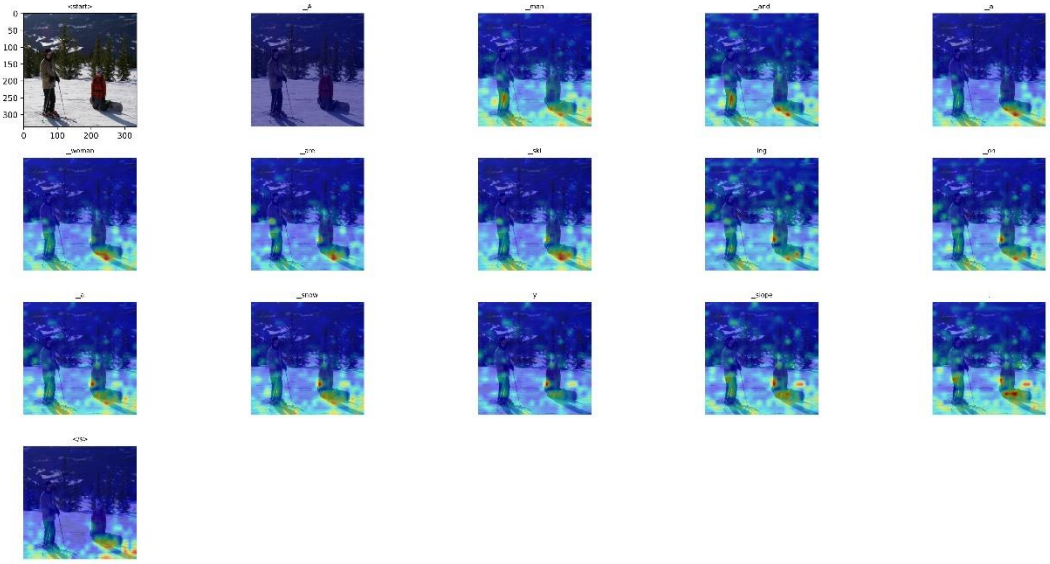
girl.jpg



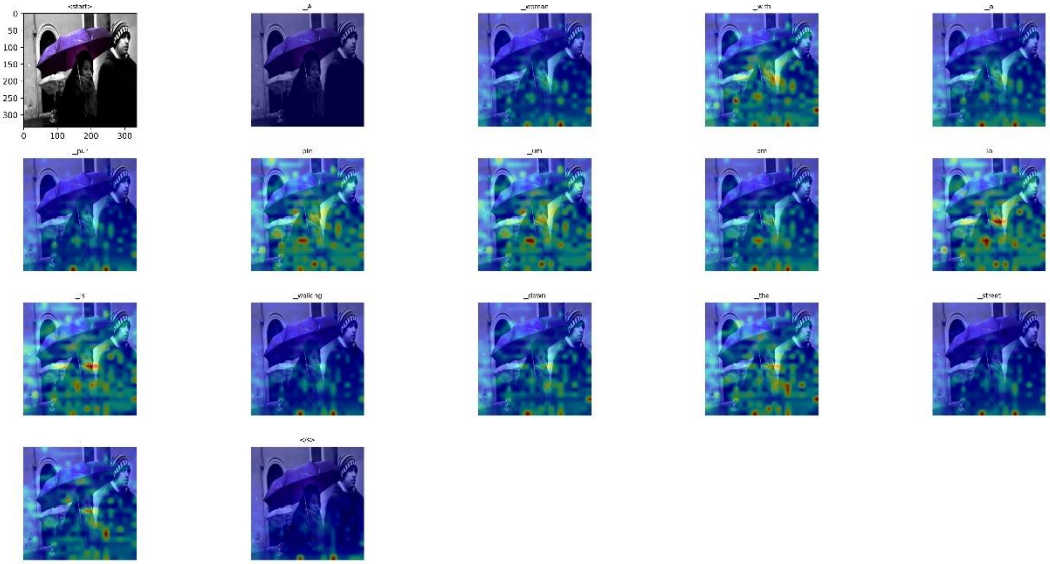
sheep.jpg



ski.jpg



umbrella.jpg

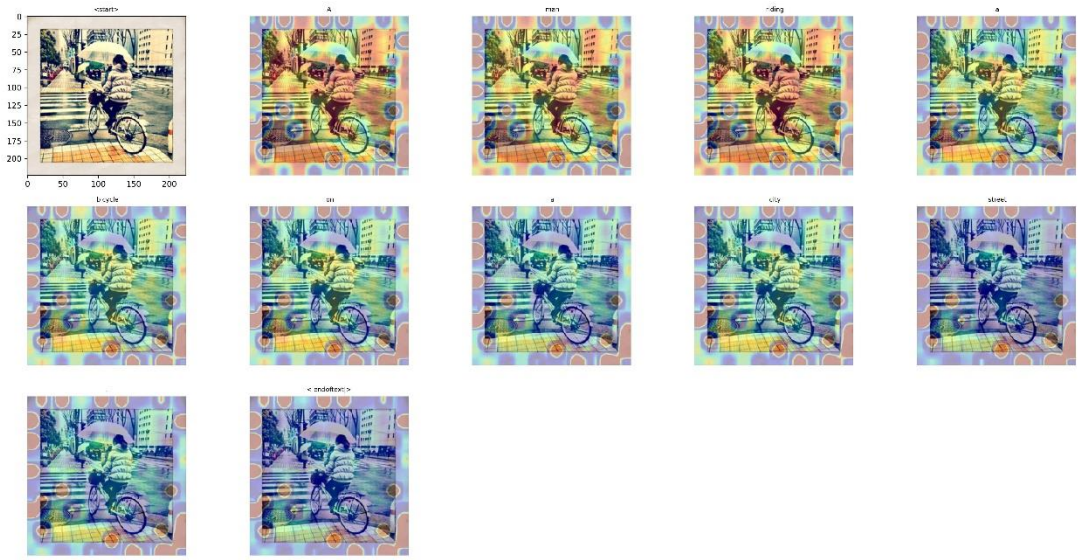


Problem2:

In Problem 2, I similarly observed that the visualization results of the earlier layers were more noticeable. For this task, I found that the second layer provided better visualization results. Similarly, I averaged the results across 12 heads.

The predicted caption and the corresponding series of attention maps:

bike.jpg



girl.jpg

sheep.jpg

ski.jpg

umbrella.jpg

2. According to CLIPScore, you need to:

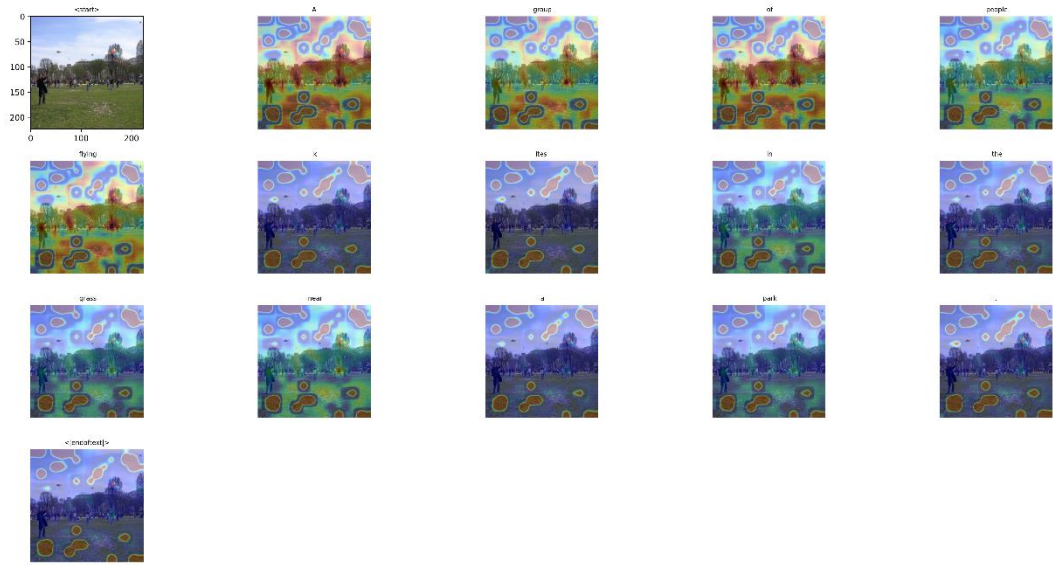
i.visualize top-1 and last-1 image-caption pairs

ii.report its corresponding CLIPScore in the validation dataset of problem 2. (3%)

top-1 image-caption pairs:000000001086.jpg

its corresponding CLIPScore:1.0083

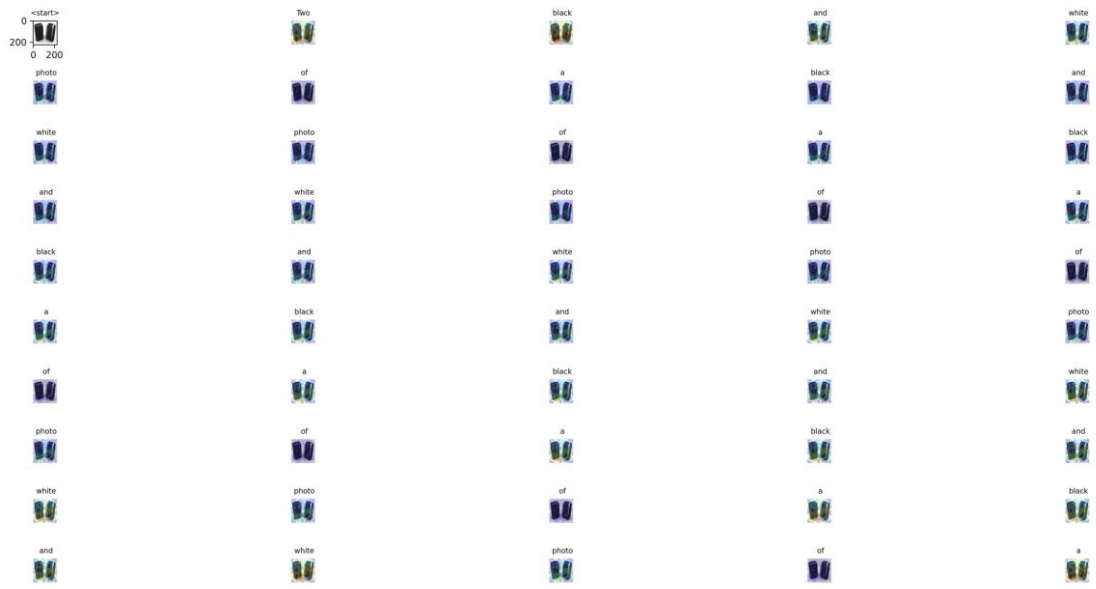
```
$ python3 P3_clipscore.py
calculating:
top-1  image-caption pairs {'filename': '000000001086', 'caption': 'A group of people flying kites in the gra
ss near a park.', 'score': 1.00830078125}
```



last -1 image-caption pairs:000000001976.jpg

its corresponding CLIPScore:0.4211

```
last-1  image-caption pairs {'filename': '000000001976', 'caption': 'Two black and white photo of a black and
 white photo of a black and white photo of a black and white photo of a black and white photo of a black and
white photo of a black and white photo of a black and white photo of a', 'score': 0.421142578125}
```



2. Analyze the predicted captions and the attention maps for each word according to the previous question. Is the caption reasonable? Does the attended region reflect the corresponding word in the caption? (3%)

For bike.jpg:
The attention seems to focus on the woman and the bike, but when the text refers to the background, such as "street" or "rain," the attention appears less distinct

For girl.jpg:
In both Problem 1 and Problem 2, it is evident that the attention captures the girl's face and the pizza.

For sheep.jpg:
In Problem 1, the results consistently focus on the sheep, with minimal differences between caption-patch associations. However, in Problem 2, words related to the sheep are concentrated on the sheep, while unrelated words tend to focus more on the background.

For ski.jpg:
In Problem 1, the results consistently focus on the skier, with minimal differences between caption-patch associations. However, in Problem 2, words related to the person or actions (standing) are concentrated on the skier, while the rest tend to focus more on the background.

For umbrella.jpg:
In both Problem 1 and Problem 2, the first few words, such as "A," "woman," "with," "a," and "umbrella," have a greater focus on the people.

For 000000001086.jpg:
Words like "group," "people," "grass," and "park" align well with the corresponding image patches, indicating that the attention correctly focuses on the relevant areas. This could be the reason for its high score.

For 000000001976.jpg:
It can be observed that the model repeatedly outputs "white," "photo," "and," and "black" in a loop until it hits the max_new_tokens limit. This explains why its result is so poor.