

Q1.1 Manually installed the required dependencies, loaded the model using transformers.CLIPModel and the processor using transformers.CLIPProcessor

```
!pip install ftfy packaging regex tqdm torch torchvision huggingface_hub safetensors timm
Collecting ftfy
+ Code + Markdown
[3]: !pip install open_clip_torch
Requirement already satisfied: open_clip_torch in /usr/local/lib/python3.11/dist-packages (2.32.0)

[4]:
import numpy as np
import matplotlib.pyplot as plt
from transformers import CLIPProcessor, CLIPModel
import torch
from PIL import Image
import requests
from open_clip import create_model_from_pretrained, get_tokenizer
import torch.nn.functional as F
import os

2025-04-20 15:03:59.223501: E_external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Un
```

Q1.2 Loaded the CLIP weights as present in the transformers library

```
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

config.json: 100% 4.19k/4.19k [00:00<00:00, 321kB/s]
pytorch_model.bin: 100% 605M/605M [00:02<00:00, 226MB/s]

Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model 4.52, even if the model was saved with a slow processor. This will result in minor differences or with `use_fast=False`.

model.safetensors: 100% 605M/605M [00:03<00:00, 195MB/s]
preprocessor_config.json: 100% 316/316 [00:00<00:00, 17.8kB/s]
tokenizer_config.json: 100% 592/592 [00:00<00:00, 48.0kB/s]
vocab.json: 100% 862k/862k [00:00<00:00, 5.33MB/s]
merges.txt: 100% 525k/525k [00:00<00:00, 26.6MB/s]
tokenizer.json: 100% 2.22M/2.22M [00:00<00:00, 10.6MB/s]
special_tokens_map.json: 100% 389/389 [00:00<00:00, 25.1kB/s]
```

Q1.3 Chose these sample captions and generated the similarity scores

```
Similarity scores for each text description:  
a person walking with a dog: 27.0259  
a man and his pet: 29.0035  
a human and canine companion: 27.7823  
a person holding a dog like a baby: 27.1600  
a man playing with his dog: 29.0925  
a person and dog resting: 26.1550  
a human holding a pet: 29.5404  
a man and his loyal companion: 27.6204  
a person with their furry friend: 28.8493  
a human and dog enjoying home time: 29.2507
```

Q1.4 Download the dependencies and used the open_clip library for the model and tokenizer

The screenshot shows a Jupyter Notebook interface. Cell [1] contains the command `!pip install ftfy packaging regex tqdm torch torchvision huggingface_hub safetensors timm`. Cell [2] is collapsed, labeled "Collecting ftfy". Cell [3] contains the command `!pip install open_clip_torch`, which is also collapsed. Cell [4] contains Python code for importing numpy, matplotlib.pyplot, CLIPProcessor, CLIPModel, torch, Image, requests, and functions from open_clip. The code uses `create_model_from_pretrained` and `get_tokenizer`. The code block ends with a timestamp: `2025-04-20 15:03:59.223501: External/local_xla/xla/stream_executor/cuda/cuda_fft.cc:4771 Un`.

Q1.5 Loaded the weights as per the github

The screenshot shows a Jupyter Notebook cell with code to load a CLIP model. It uses `create_model_from_pretrained` and `get_tokenizer` from the openclip library. Below the code, a progress bar shows the download of various files: `open_clip_pytorch_model.bin`, `open_clip_config.json`, `tokenizer_config.json`, `config.json`, `vocab.txt`, and `tokenizer.json`. The progress bar indicates speeds ranging from 5.22kB/s to 217MB/s.

Q1.6 Calculated the similarity scores using the CLIPS model

```
CLIPS Similarity scores for each text:  
a person walking with a dog: 8.3078  
a man and his pet: 15.4639  
a human and canine companion: 13.9483  
a person holding a dog like a baby: 11.1285  
a man playing with his dog: 13.9156  
a person and dog resting: 7.2605  
a human holding a pet: 13.1432  
a man and his loyal companion: 13.7897  
a person with their furry friend: 12.7554  
a human and dog enjoying home time: 12.4032
```

Q1.7 The scores generated by the CLIP (ViT-B/32) have very low variance, ranging from 26.1550 to 29.5404. It gives a higher score to a semantically more accurate caption, such as “a human holding a pet”. On the other hand, the CLIPS model (ViT-L/14) shows quite a high variance, ranging from 7.2605 to 15.4639, punishing captions which are quite incorrect with regards to the image, such as “a person and dog resting”, when the human is standing in the image, and gives a high score to simple but correct prompts such as “a man and his pet”.

Q2.1 Manually installed the dependencies, used transformers.BlipProcessor and transformers.BlipForQuestionAnswering, and imported the pretrained weights of the base model

```
[6]: blip_model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base").to(device)  
blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")  
  
config.json: 100% ██████████ 4.56k/4.56k [00:00<00:00, 372kB/s]  
model.safetensors: 100% ██████████ 1.54G/1.54G [00:05<00:00, 275MB/s]  
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model.  
4.52, even if the model was saved with a slow processor. This will result in minor differences  
or with `use_fast=False`.  
preprocessor_config.json: 100% ██████████ 445/445 [00:00<00:00, 43.6kB/s]  
tokenizer_config.json: 100% ██████████ 592/592 [00:00<00:00, 59.2kB/s]  
vocab.txt: 100% ██████████ 232k/232k [00:00<00:00, 1.43MB/s]  
tokenizer.json: 100% ██████████ 711k/711k [00:00<00:00, 2.90MB/s]  
special_tokens_map.json: 100% ██████████ 125/125 [00:00<00:00, 13.2kB/s]
```

Q2.2 Generated answer using the appropriate model and processor objects for the question

```
[8]: inputs_3 = blip_processor(image_2, question, return_tensors="pt").to(device)

with torch.no_grad():
    output_3 = blip_model.generate(**inputs_3)

answer_3 = blip_processor.decode(output_3[0], skip_special_tokens=True)
print("Question:", question)
print("Reply:", answer_3)
```

Question: Where is the dog present in the image?
 Reply: in man ' s arms

Q2.3 Did the same for the other question to be asked

```
[9]: question_2 = "Where is the man present in the image?"
inputs_3_2 = blip_processor(image_2, question_2, return_tensors="pt").to(device)

with torch.no_grad():
    output_3_2 = blip_model.generate(**inputs_3_2)

answer_3_2 = blip_processor.decode(output_3_2[0], skip_special_tokens=True)
print("Question:", question_2)
print("Reply:", answer_3_2)
```

Question: Where is the man present in the image?
 Reply: living room

Q2.4 The answers of the above questions were accurate and straightforward.

Q3.1 Used the transformers.BlipForConditionalGeneration library and loaded the weights for the base model

```
[8]: blipc_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base").to(device)
blipc_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
```

config.json: 100% ██████████ 4.56k/4.56k [00:00<00:00, 453kB/s]
 pytorch_model.bin: 100% ██████████ 990M/990M [00:11<00:00, 86.0MB/s]
 model.safetensors: 100% ██████████ 990M/990M [00:11<00:00, 82.8MB/s]
 preprocessor_config.json: 100% ██████████ 287/287 [00:00<00:00, 21.9kB/s]
 tokenizer_config.json: 100% ██████████ 506/506 [00:00<00:00, 36.1kB/s]
 vocab.txt: 100% ██████████ 232k/232k [00:00<00:00, 13.0MB/s]
 tokenizer.json: 100% ██████████ 711k/711k [00:00<00:00, 26.6MB/s]
 special_tokens_map.json: 100% ██████████ 125/125 [00:00<00:00, 10.8kB/s]

Q3.2 Generated Caption for each image using the model

Caption: a small dog running across a green field
ILSVRC2012_test_00000004.jpg



Caption: a man riding a bike down a wet street
ILSVRC2012_test_00000023.jpg



Caption: a small dog standing on a stone ledge
ILSVRC2012_test_00000022.jpg



Caption: a man in a suit and tie sitting on a couch
ILSVRC2012_test_00000026.jpg



Caption: a family sitting in a pool with a towel
ILSVRC2012_test_00000018.jpg



Caption: a small bird sitting on a plant
ILSVRC2012_test_00000019.jpg



Caption: a small dog walking on a green carpet
ILSVRC2012_test_00000003.jpg



Caption: a duck drinking water from a pond
ILSVRC2012_test_00000030.jpg



Caption: a coffee machine with two cups on it

ILSVRC2012_test_00000034.jpg



- Mocca Espresso -

Caption: a brown butterfly sitting on a green plant

ILSVRC2012_test_00000025.jpg



Q3.3 Used CLIP to evaluate the semantic accuracy of the Generated Caption, Similarity scores are as follows:

Caption: a small dog running across a green field

CLIP Eval Score: 32.70260238647461

ILSVRC2012_test_00000004.jpg



Caption: a man riding a bike down a wet street

CLIP Eval Score: 30.83452606201172

ILSVRC2012_test_00000023.jpg



Caption: a small dog standing on a stone ledge

CLIP Eval Score: 31.03472137451172

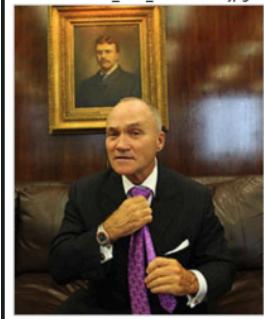
ILSVRC2012_test_00000022.jpg



Caption: a man in a suit and tie sitting on a couch

CLIP Eval Score: 28.895353317260742

ILSVRC2012_test_00000026.jpg



Caption: a family sitting in a pool with a towel

CLIP Eval Score: 31.336458206176758

ILSVRC2012_test_00000018.jpg



Caption: a small bird sitting on a plant

CLIP Eval Score: 28.93827247619629

ILSVRC2012_test_00000019.jpg



Caption: a small dog walking on a green carpet

CLIP Eval Score: 31.566001892089844

ILSVRC2012_test_00000003.jpg

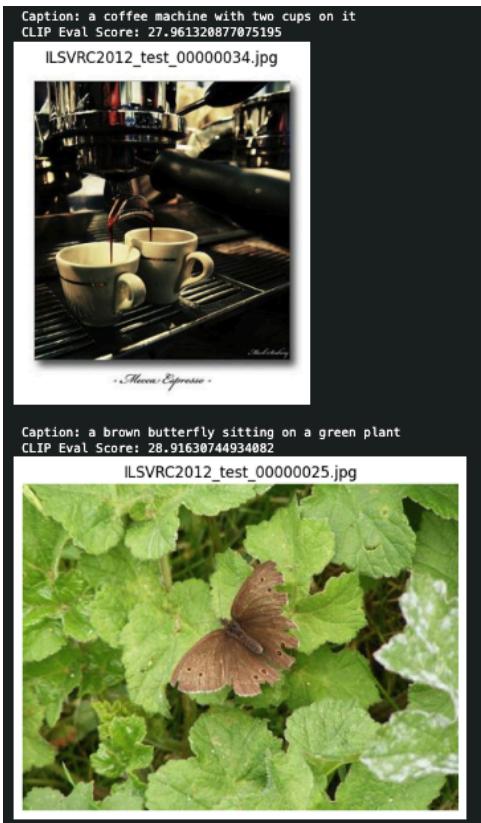


Caption: a duck drinking water from a pond

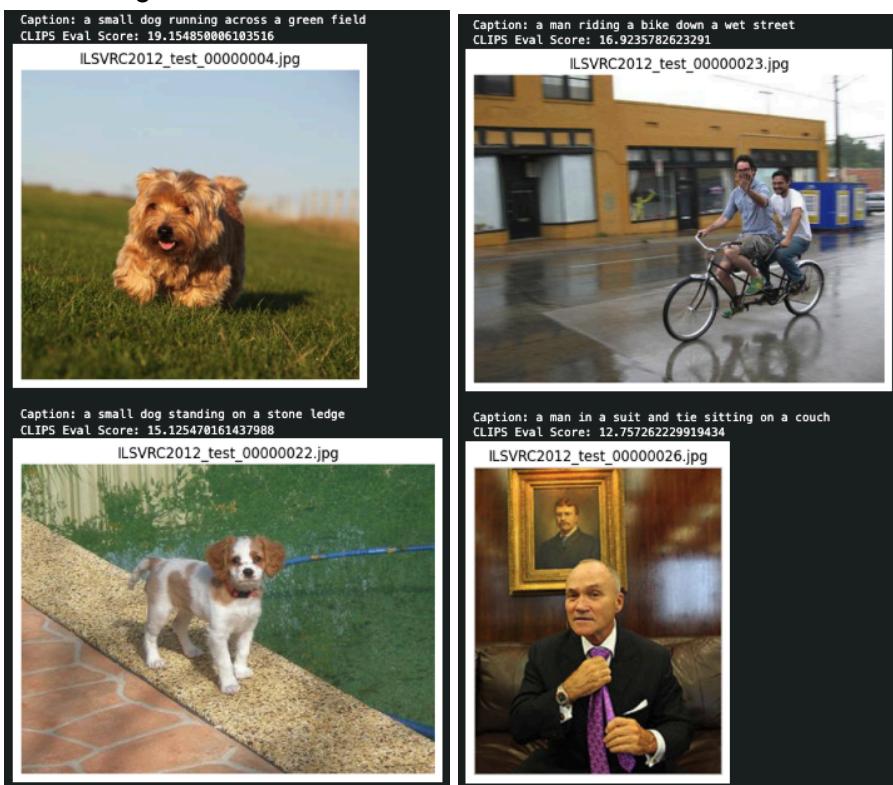
CLIP Eval Score: 30.52522087097168

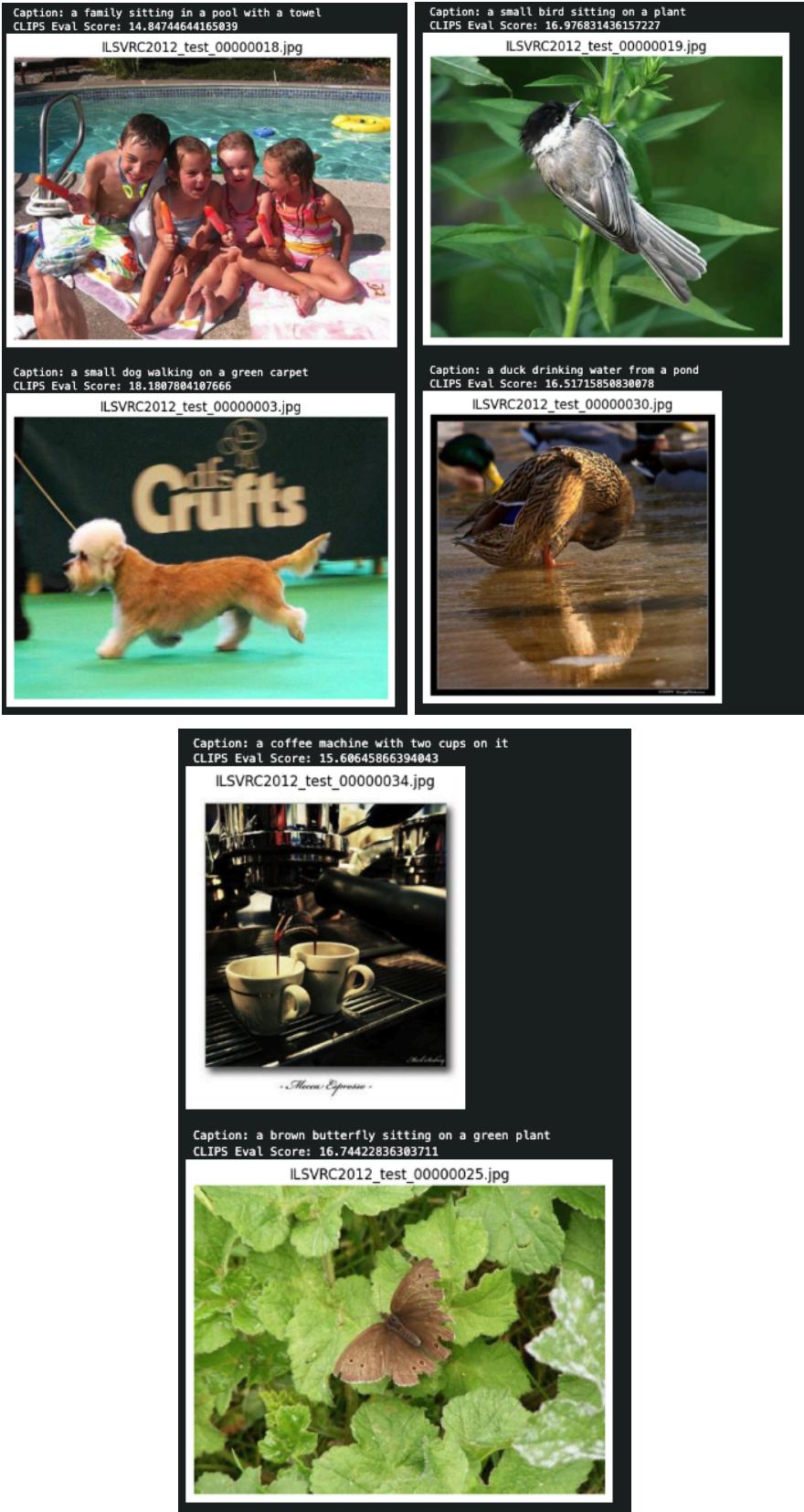
ILSVRC2012_test_00000030.jpg





Q3.4 Did the same using the CLIPS model





Q3.5 The direct **CLIP/CLIPS scores** are cosine similarity values between the textual and image embeddings. They are great if one wants an overall score of the semantic meaning between the text and the image. The **softmaxed version** is great when we want to compare multiple text descriptions of the image. It should be used to get the best description out of many. Another metric is the **CLIP-Score** which is the normalized version of the above scores and is great to compare the similarity across different pairs of text and images and different datasets

Q4.1 Cloned the github and installed the dependencies. Largely used the github (demo_inference.py file)

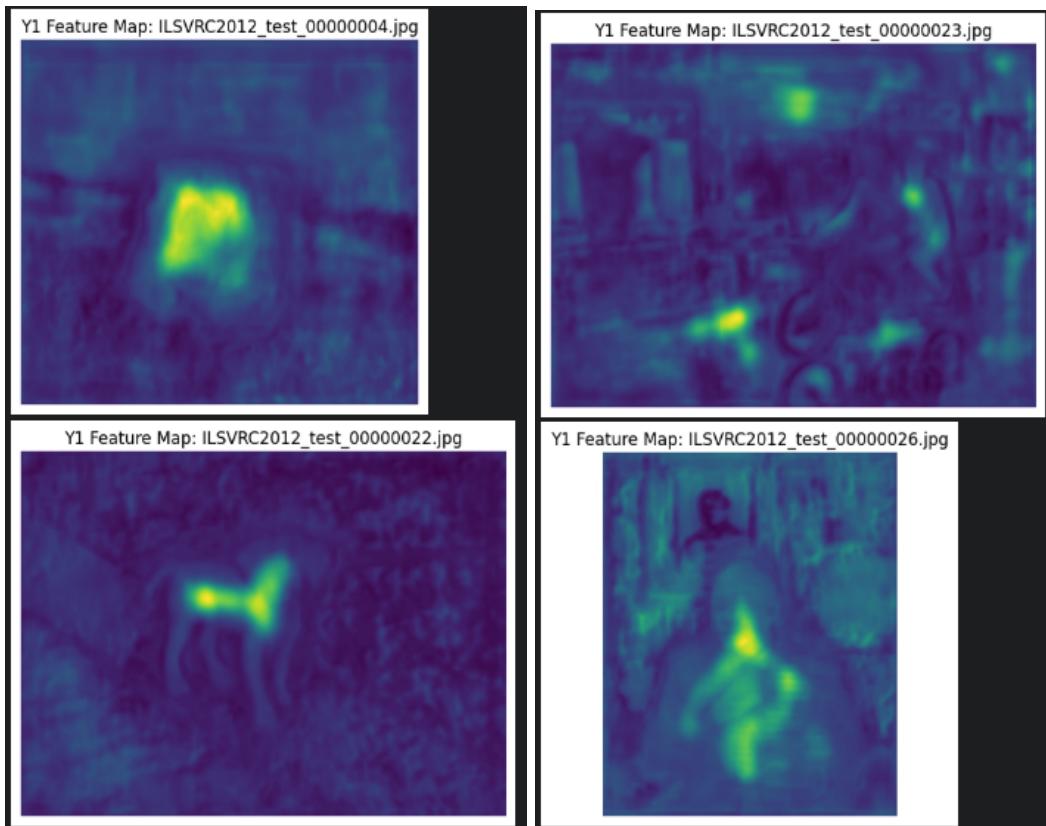
```
[15]:  
from bert.modeling_bert import BertModel  
from lib import segmentation  
  
weights = '/kaggle/input/refcoco.pth'  
  
class args:  
    swin_type = 'base'  
    window12 = True  
    mha = ''  
    fusion_drop = 0.0  
  
    single_model = segmentation.__dict__['lavt'](pretrained='', args=args)  
    single_model.to(device)  
    model_class = BertModel  
    single_bert_model = model_class.from_pretrained('bert-base-uncased')  
    single_bert_model.pooler = None  
  
    checkpoint = torch.load(weights, map_location='cpu')  
    single_bert_model.load_state_dict(checkpoint['bert_model'])  
    single_model.load_state_dict(checkpoint['model'])  
    model = single_model.to(device)  
    bert_model = single_bert_model.to(device)  
  
    ^ /usr/local/lib/python3.11/dist-packages/timm/models/layers/_init_.py:48: FutureWarning
```

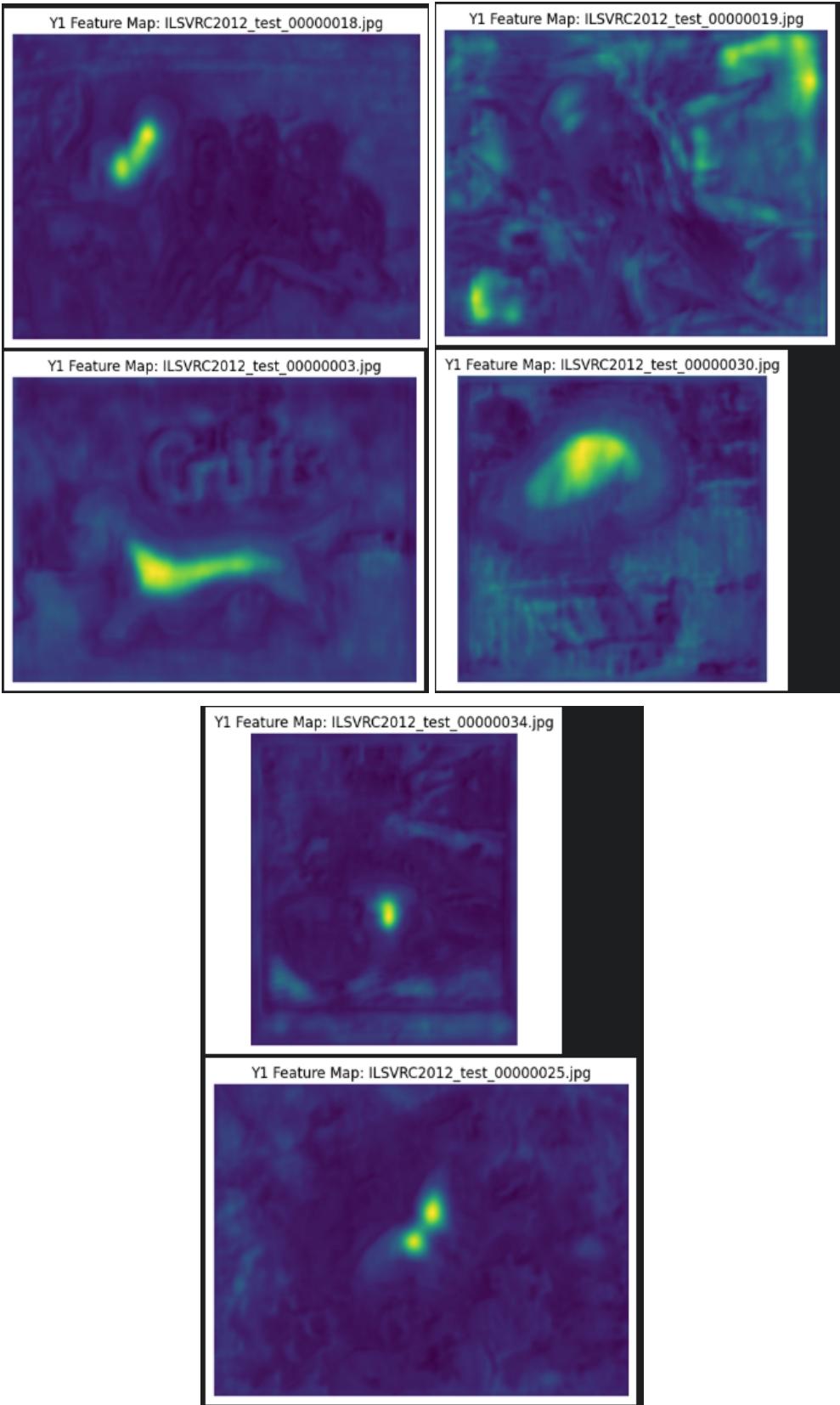
Q4.2 Performed segmentation as per the given prompts





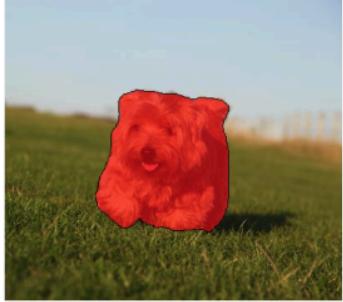
Q4.3 Plotted the Y1 feature map for each image, after saving the feature maps while performing segmentation





Q4.4 Created reference texts as similar in format to the original texts as possible, here are the results:

ILSVRC2012_test_0000004.jpg : the black eyes of the dog
ILSVRC2012_test_0000004.jpg



ILSVRC2012_test_0000023.jpg : the blue dustbin in the picture
ILSVRC2012_test_0000023.jpg



ILSVRC2012_test_0000022.jpg : the brown marble floor in the picture
ILSVRC2012_test_0000022.jpg



ILSVRC2012_test_0000026.jpg : the brown hair in the painting
ILSVRC2012_test_0000026.jpg



ILSVRC2012_test_0000018.jpg : the red icecream in the children's hands
ILSVRC2012_test_0000018.jpg



ILSVRC2012_test_0000019.jpg : the green plant in the picture
ILSVRC2012_test_0000019.jpg

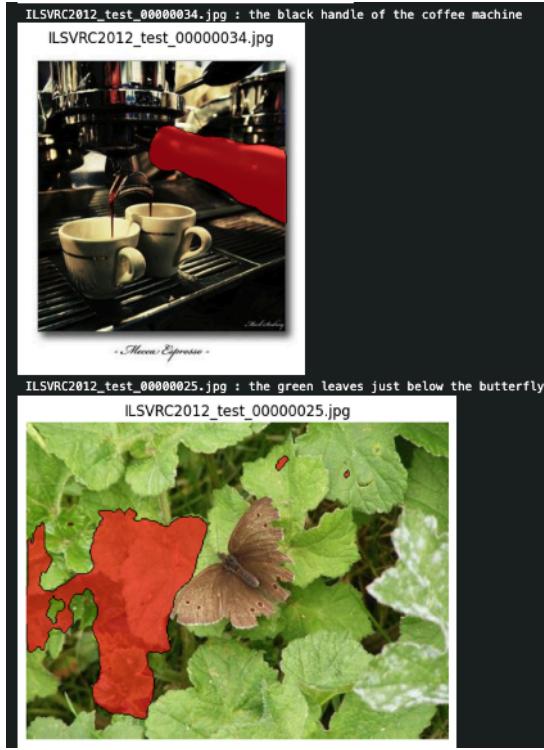


ILSVRC2012_test_0000003.jpg : the golden logo in the picture
ILSVRC2012_test_0000003.jpg



ILSVRC2012_test_0000030.jpg : the yellow beak of the duck
ILSVRC2012_test_0000030.jpg





Q5.1 Cloned the github and installed the required dependencies, used eval code largely from the github

```

images_dir= '/kaggle/input/cv-a3-all/Images-20250418T060633Z-001/Images'
output_dir= 'output_segmentations'
dinov2_weights= '/kaggle/input/cv-a3-all/dinov2_vitl14_pretrain.pth'
sam_weights= '/kaggle/input/cv-a3-all/sam_vit_h_4b8939.pth'

args = argparse.Namespace(
    images_dir=images_dir,
    output_dir=output_dir,
    dinov2_weights=dinov2_weights,
    sam_weights=sam_weights,

    dinov2_size='vit_large',
    sam_size='vit_h',
    num_centers=8,
    use_box=False,
    use_points_or_centers=False,
    sample_range=(4,6),
    max_sample_iterations=30,
    alpha=1.0,
    beta=0.0,
    exp=0.0,
    emd_filter=0.0,
    purity_filter=0.0,
    coverage_filter=0.0,
    use_score_filter=False,
    deep_score_norm_filter=0.1,
    deep_score_filter=0.33,
    topk_scores_threshold=0.7,
    num_merging_mask=10,

    points_per_side=64,
    pred_iou_thresh=0.88,
    stability_score_thresh=0.95,
    sel_stability_score_thresh=0.0,
    iou_filter=0.0,
    box_mms_thresh=1.0,
    output_layer=3,
    dense_multimask_output=0,
    use_dense_mask=0,
    multimask_output=0
)

args.sample_range = eval(args.sample_range)

args.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

matcher = build_matcher_oss(args)

```

Q5.2 Used the matcher paradigm containing DinoV2 and SAM models

