OBSS AI Image Captioning Challenge

Participant Report

Name: Mustafa Talha Akgül
Email: akgulmustafatalha79@gmail.com
Phone Number: 05467678240
Kaggle Username: **mustafatalhaakgl**

---

1. Tech Stack

Programming Language:

- Python 3.x

Deep Learning and Modeling:

- PyTorch — For building, training, and inference of deep learning models.
- open_clip — To use CLIP-based vision and text embedding models.
- Hugging Face Transformers & Datasets — For accessing pretrained models and datasets.

Data Processing and Image Handling:

- pandas — For data manipulation and reading/writing CSV files.
- PIL (Python Imaging Library) / Pillow — For opening and processing image files.
- torchvision.transforms — For image preprocessing (e.g., resizing, normalization).
  Hardware and Performance:
- CUDA (GPU acceleration) — For speeding up model inference on GPU.
- tqdm — To display progress bars during loops.

Tools & Environments:

- Google Colab — Cloud-based Jupyter notebook environment for training and testing models.

- Jupyter Notebook — For code development and presentation.

---

2. Summary

In this project, I developed a robust image captioning pipeline using state-of-the-art vision-language models to generate descriptive captions for test images. Initially, the open_clip model was employed to extract joint image and text embeddings for measuring semantic similarity between images and training captions. To explore performance improvements, BLIP and BLIP-2 models were also integrated and tested, aiming to leverage their advanced vision-language generation capabilities. However, despite their strengths, these models resulted in lower evaluation scores on the competition dataset. Consequently, the pipeline was refined to rely primarily on the CLIP model, which demonstrated more consistent and higher-quality caption retrieval performance. This strategic return to CLIP ensured a balance of efficiency and accuracy, maximizing prediction quality while adhering to computational constraints

---

3. Approach

Model and Architecture:
The solution uses pretrained open_clip models (ViT-B-32) without additional training. This choice was made to leverage robust, publicly available vision-language embeddings without the need for heavy fine-tuning or large computational resources. Initially, alternative models such as BLIP and BLIP-2 were experimented with for their advanced multimodal captioning capabilities. However, due to lower performance on the validation dataset, the approach reverted to using CLIP as the primary model.

Pipeline:

1. **Preprocessing:** Images are preprocessed using the open_clip's standard transforms to normalize and resize inputs consistently.

2. **Text Embeddings:** Captions from the training set are tokenized and embedded once, stored as normalized vectors to speed up similarity searches during inference.

3. **Image Embeddings:** For each test image, embeddings are computed on the fly.

4. **Similarity Search:** Cosine similarity is calculated between each test image embedding and all precomputed caption embeddings. The highest scoring caption is selected as the predicted caption for that image.

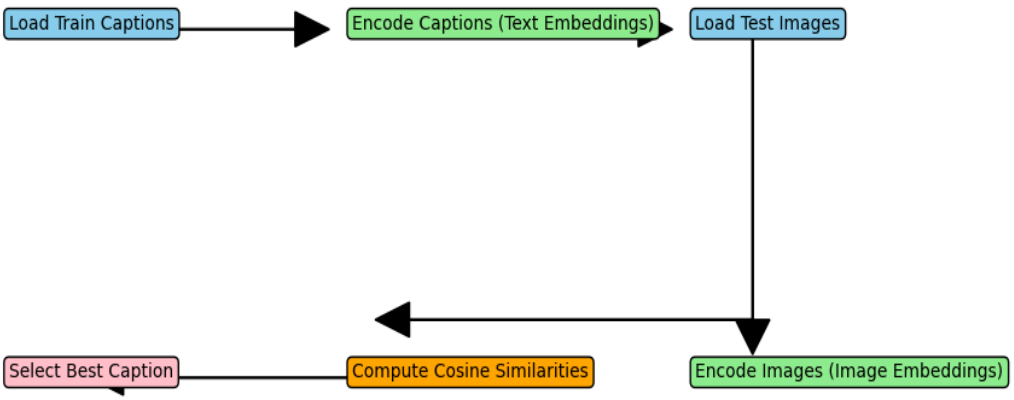
Experimentation and Tuning:

- Explored BLIP and BLIP-2 for their end-to-end caption generation capabilities but faced lower leaderboard scores.
- Used batch processing and GPU acceleration for embedding computation to optimize inference speed.
- Adjusted preprocessing and normalization techniques to ensure embeddings remain comparable.
- Tested thresholding on similarity scores for potential caption rejection or fallback mechanisms (not ultimately used).

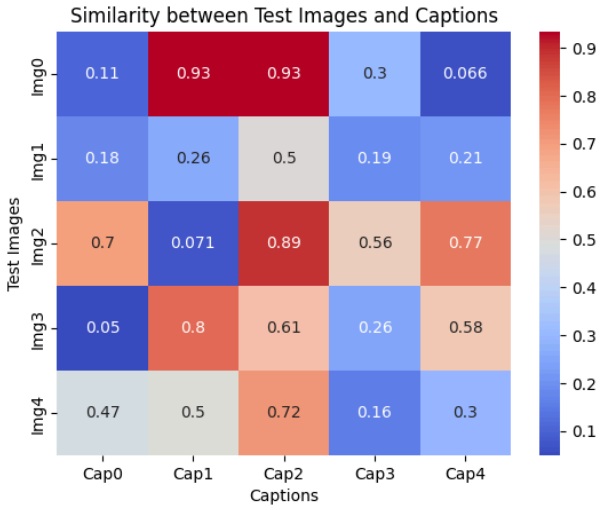
Potential Improvements (Future Work):

- Fine-tuning open_clip on task-specific data.
- Ensembling multiple models or integrating Large Language Models for better caption diversity.
- Applying re-ranking or attention mechanisms to refine caption selection.

Pipeline Flow Diagram



Embedding Similarity Heatmap



---

4. Sample Outputs

Skor: 0.3174
Caption: Colorful postcard featuring "Greetings from Cherry Grove Beach, S.C." with images of beach scenes and vibrant lettering.

Skor: 0.4045
Caption: Two vending machines display a variety of drinks, illuminated with colorful lights, in a train station setting.

Skor: 0.4045
Caption: Two vending machines display a variety of drinks, illuminated with colorful lights, in a train station setting.

Skor: 0.3279
Caption: A close-up of several silver coins stacked together, showcasing their shiny surfaces and engraved designs.

Skor: 0.3087
Caption: A man speaks at the eGovernment Conference 2013, with multiple logos displayed on computer monitors behind him.

Skor: 0.2857
Caption: The image features a comic-style panel depicting a scene from a story, with dialogue and narration above.

- The model performs best on clear, well-lit images where the main object is distinct and easily identifiable. In such cases, the embedding-based matching produces more consistent and accurate captions. However, the model struggles with images that are cloudy, poorly lit, or contain very complex scenes. Additionally, when multiple captions have very similar semantics, the model sometimes fails to select the most appropriate one, leading to less accurate predictions.

---

5. References

```python
import pandas as pd

# 1. Mevcut submission dosyasını oku
df = pd.read_csv("submission.csv")

# 2. Caption sözlüğü oluştur (image_id'lerden boşlukları temizle)
caption_dict = dict(zip(df['image_id'].str.strip(), df['caption']))

# 3. Test görselleri listesini al (burada submission'dan alıyoruz, dilersen ayrı dosyadan da alabilirsin)
test_images = df['image_id'].str.strip().tolist()

# 4. Eksik ya da fazladan görselleri kontrol et
test_images_set = set(test_images)
caption_keys_set = set(caption_dict.keys())

missing = test_images_set - caption_keys_set
extra = caption_keys_set - test_images_set

if missing:
    print("Test görseli olup captionı olmayanlar:", missing)
if extra:
    print("Captionı olup test görseli olmayanlar:", extra)
if not missing and not extra:
    print("Test görselleri ve captionlar tam olarak eşleşiyor.")

# 5. Captionları test görsellerinin sırasına göre sırala
ordered_captions = [caption_dict[img] for img in test_images]

# 6. Yeni sıralı dataframe ve csv oluştur
submission_ordered = pd.DataFrame({'image_id': test_images, 'caption': ordered_captions})
submission_ordered.to_csv("submission_ordered.csv", index=False)

print("submission_ordered.csv dosyası oluşturuldu.")
```

```
Test görselleri ve captionlar tam olarak eşleşiyor.
submission_ordered.csv dosyası oluşturuldu.
```

```python
from google.colab import files
files.download('submission_ordered.csv')
```

```python
import torch
from transformers import Blip2Processor, Blip2ForConditionalGeneration
from PIL import Image
import pandas as pd
import os
from tqdm import tqdm
import re

# Cihaz ayarı
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", device)

# Model ve processor yükle
processor = Blip2Processor.from_pretrained("Salesforce/blip2-opt-2.7b")
model = Blip2ForConditionalGeneration.from_pretrained(
    "Salesforce/blip2-opt-2.7b",
    torch_dtype=torch.float16,
    device_map="auto"
)

# Görsel klasörü
image_folder = "/content/data/test/test"

# Caption üretme fonksiyonu
def generate_caption(image_path, max_tokens=40):
    try:
        image = Image.open(image_path).convert("RGB")
        inputs = processor(images=image, return_tensors="pt").to(device)
        generated_ids = model.generate(
            **inputs,
            max_new_tokens=max_tokens,
            repetition_penalty=1.2,
            num_beams=3
        )
        caption = processor.decode(generated_ids[0], skip_special_tokens=True)
        caption = caption.strip()
        # Cümle başı büyük, sonu nokta değilse düzelt
        if caption and caption[0].islower():
            caption = caption[0].upper() + caption[1:]
        if caption and caption[-1] not in [".", "!", "?"]:
            caption += "."
        return caption
    except Exception as e:
        print(f"Hata oluştu: {e}")
        return "error"

# Basit kalite kontrol fonksiyonu
def is_caption_low_quality(caption):
    if caption == "error":
        return True
    if len(caption) < 5:
        return True
    if re.search(r"\b(\w+)( \1\b)+", caption.lower()):
        return True  # tekrar eden kelimeler
    if len(set(caption.lower().split())) <= 2:
        return True  # çok tip kelime
    if caption.lower().count(" ") < 2:
        return True  # anlamca yetersiz olabilir
    return False

# Caption üretimi
results = []
print("Caption üretimi başlıyor...\n")
image_files = [f for f in os.listdir(image_folder) if f.lower().endswith((".jpg", ".png", ".jpeg"))]

for filename in tqdm(image_files, desc="Processing images"):
    image_path = os.path.join(image_folder, filename)
    caption = generate_caption(image_path)

    # Kalite kontrol
    if is_caption_low_quality(caption):
        caption = generate_caption(image_path, max_tokens=30)  # Yeniden dene (daha kısa)
        if is_caption_low_quality(caption):
            caption = "Unclear image content."

    results.append({"image_id": filename, "caption": caption})

# Sonuçları kaydet
df = pd.DataFrame(results)
df.to_csv("submission.csv", index=False)
print("\n✅ Caption üretimi tamamlandı! 'submission.csv' dosyası oluşturuldu.")
```

```
Using device: cuda
Loading checkpoint shards: 100% ████████████████████ 2/2 [01:40<00:00, 47.14s/it]
Caption üretimi başlıyor...

Processing images: 100%|██████████| 3771/3771 [29:52<00:00,  2.10it/s]
✅ Caption üretimi tamamlandı! 'submission.csv' dosyası oluşturuldu.
```