Assignment-2

Objective

To design and implement a content-based video retrieval system that retrieves the best matched video (2–3 seconds duration) from a database using either:

1. A single image as input, or
2. A single video as input.

Code:

```
import os
import torch
import open_clip
import faiss
import cv2
import numpy as np
from PIL import Image
from tqdm import tqdm
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Loading CLIP model...")
model, _, preprocess = open_clip.create_model_and_transforms(
  'ViT-B-32', pretrained='openai'
)
model = model.to(device)
model.eval()
def extract_frames(video_path, frame_skip=30):
  cap = cv2.VideoCapture(video_path)
```

```
    frames = []
    count = 0
    if not cap.isOpened():
        print(f"Skipping corrupted video: {video_path}")
        return []
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        if count % frame_skip == 0:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frames.append(Image.fromarray(frame))
        count += 1
    cap.release()
    return frames
def get_image_embedding(image):
    image = preprocess(image).unsqueeze(0).to(device)
    with torch.no_grad():
        embedding = model.encode_image(image)
        embedding = embedding / embedding.norm(dim=-1,
keepdim=True)
    return embedding.cpu().numpy()
def preprocess_images(frames):
    processed = []
```

```
    for image in frames:
        image = preprocess(image).unsqueeze(0)
        processed.append(image)
    return torch.cat(processed, dim=0).to(device)
def get_video_embedding(video_path):
    frames = extract_frames(video_path)
    if len(frames) == 0:
        return None
    inputs = preprocess_images(frames)
    with torch.no_grad():
        image_features = model.encode_image(inputs)
        image_features = image_features /
image_features.norm(dim=-1, keepdim=True)

    image_features = image_features.cpu().numpy()
    return np.mean(image_features, axis=0)

def create_database_embeddings(video_folder):
    embeddings = []
    video_paths = []

    for file in tqdm(os.listdir(video_folder)):
        if file.endswith(".mp4"):
            path = os.path.join(video_folder, file)
```

```
        emb = get_video_embedding(path)

        if emb is not None:
            embeddings.append(emb)
            video_paths.append(path)

    embeddings = np.vstack(embeddings).astype("float32")
    return embeddings, video_paths
def search(query_embedding, index, video_paths, top_k=1):
    D, I = index.search(query_embedding, top_k)
    return [video_paths[i] for i in I[0]]
if __name__ == "__main__":

    db_folder = "database_videos"  # make sure this matches your
folder

    print("Creating database embeddings...")
    embeddings, video_paths =
create_database_embeddings(db_folder)

    dimension = embeddings.shape[1]
    index = faiss.IndexFlatL2(dimension)
    index.add(embeddings)
```

```
print("\n1 → Search by Image")
print("2 → Search by Video")

choice = input("Enter choice: ")

if choice == "1":
    image = Image.open("query/query.jpg").convert("RGB")
    query_emb = get_image_embedding(image)

elif choice == "2":
    query_emb = get_video_embedding("query/query.mp4")

else:
    print("Invalid choice")
    exit()

if query_emb is None:
    print("Query video/image invalid.")
    exit()

query_emb = query_emb.reshape(1, -1).astype("float32")

results = search(query_emb, index, video_paths)
```

```
print("\nBest matched video:")
print(results[0])
```

OUTPUT: