Art_Extract_Test2 (1)

March 30, 2025

[1]:
```
import os
import pandas as pd
import numpy as np
import requests
from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt
from tqdm import tqdm
import torch
from torchvision import models, transforms
import faiss
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import zipfile
import torch.optim as optim
import torch.nn as nn
from PIL import Image
from facenet_pytorch import MTCNN
from sklearn.metrics.pairwise import cosine_similarity
import json
from skimage.metrics import structural_similarity as ssim
from sklearn.metrics import mean_squared_error
from PIL import Image, UnidentifiedImageError
```

[2]:
```
class Config:
    # Paths
    DATA_DIR = "nga_data"
    METADATA_DIR = os.path.join(DATA_DIR, "opendata-main", "data")
    IMAGES_DIR = os.path.join(DATA_DIR, "images")

    data_url = "https://github.com/NationalGalleryOfArt/opendata/archive/refs/
 ↪heads/main.zip"


    DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
    BATCH_SIZE = 16
```

```
    IMAGE_SIZE = 224
    EMBEDDING_SIZE = 512


    TOP_K = 10

os.makedirs(Config.DATA_DIR, exist_ok=True)
os.makedirs(Config.IMAGES_DIR, exist_ok=True)
os.makedirs(Config.METADATA_DIR, exist_ok=True)
```

[3]:
```
def download_and_extract(url, destination_folder):
    """Downloads and extracts a zip file from a given URL."""

    filename = os.path.basename(url)

    download_path = os.path.join(destination_folder, filename)

    response = requests.get(url, stream=True)
    total_size = int(response.headers.get('content-length', 0))
    block_size = 1024

    with open(download_path, 'wb') as f:
        for data in tqdm(response.iter_content(block_size),
                         total=total_size // block_size,
                         unit='KB', unit_scale=True):
            f.write(data)

    print(f"Extracting {download_path} to {destination_folder}")
    with zipfile.ZipFile(download_path, 'r') as zip_ref:
        zip_ref.extractall(destination_folder)

    os.remove(download_path)

destination_folder = 'nga_data'
os.makedirs(destination_folder, exist_ok=True)
download_and_extract(Config.data_url, destination_folder)
```

```
40.3kKB [00:05, 6.86kKB/s]

Extracting nga_data/main.zip to nga_data
```

[4]:
```
BASE_DIR = 'nga_data'
DATA_DIR = os.path.join(BASE_DIR, 'opendata-main', 'data')
IMAGE_DIR = os.path.join(BASE_DIR, 'images')

os.makedirs(IMAGE_DIR, exist_ok=True)

def load_data():
```

```
    objects_df = pd.read_csv(os.path.join(DATA_DIR, 'objects.csv'))
    images_df = pd.read_csv(os.path.join(DATA_DIR, 'published_images.csv'))

    paintings_df = objects_df[
        ((objects_df['classification'].str.contains('painting', case=False,␣
↪na=False)) |
         (objects_df['classification'].str.contains('portrait', case=False,␣
↪na=False)) |
         (objects_df['title'].str.contains('portrait', case=False, na=False))) &
        (objects_df['medium'].notna())]

    if 'style' in objects_df.columns:
        paintings_df['style_category'] = paintings_df['style'].apply(
            lambda x: categorize_style(x) if isinstance(x, str) else 'unknown'
        )

    painting_images = pd.merge(
        paintings_df,
        images_df,
        left_on='objectid',
        right_on='depictstmsobjectid',
        how='inner'
    )

    painting_images = painting_images[
        (painting_images['viewtype'] == 'primary') |
        (painting_images['viewtype'] == 'alternate') &␣
↪(painting_images['iiifurl'].notna())
    ]

    painting_images['has_high_res'] = painting_images['width'] > 1000

    return painting_images
```

[5]:
```
def categorize_style(style_text):
    if not isinstance(style_text, str):
        return 'unknown'

    style_text = style_text.lower()
    if any(term in style_text for term in ['renaissance', 'baroque', 'rococo']):
        return 'classical'
    elif any(term in style_text for term in ['impressionist', 'impressionism']):
        return 'impressionism'
    elif any(term in style_text for term in ['modern', 'contemporary',␣
↪'abstract']):
        return 'modern'
    else:
```

```
        return 'other'

def download_images(image_data, limit=10000):
    """
    Download images from IIIF URLs with improved quality control
    """
    images = []
    object_ids = []
    metadata = []

    existing_images = [f for f in os.listdir(IMAGE_DIR) if f.endswith('.jpg')]
    if len(existing_images) >= limit:
        existing_ids = [f.split('.')[0] for f in existing_images]
        sample_data = image_data[image_data['objectid'].astype(str).
 ↪isin(existing_ids)]
        sample_data = sample_data.head(limit)
        print(f"Using {len(sample_data)} existing images")
    else:
        remaining = limit - len(existing_images)
        existing_ids = [f.split('.')[0] for f in existing_images]
        existing_data = image_data[image_data['objectid'].astype(str).
 ↪isin(existing_ids)]
        new_sample = image_data[~image_data['objectid'].astype(str).
 ↪isin(existing_ids)]

        if 'has_high_res' in new_sample.columns and␣
 ↪len(new_sample[new_sample['has_high_res']]) > 0:
            high_res_data = new_sample[new_sample['has_high_res']].sample(
                min(remaining, len(new_sample[new_sample['has_high_res']]))
            )
            remaining_count = remaining - len(high_res_data)
            if remaining_count > 0 and len(new_sample) > len(high_res_data):
                remaining_data = new_sample[~new_sample['has_high_res']].sample(
                    min(remaining_count,␣
 ↪len(new_sample[~new_sample['has_high_res']]))
                )
                new_sample_data = pd.concat([high_res_data, remaining_data])
            else:
                new_sample_data = high_res_data
        else:
            new_sample_data = new_sample.sample(min(remaining, len(new_sample)))

        sample_data = pd.concat([existing_data, new_sample_data])
        print(f"Using {len(existing_data)} existing images and downloading␣
 ↪{len(new_sample_data)} new images")
```

```
    if 'has_high_res' in image_data.columns:
        high_res_data = image_data[image_data['has_high_res']].
↪sample(min(limit, len(image_data[image_data['has_high_res']])))
        remaining_count = limit - len(high_res_data)
        if remaining_count > 0 and len(image_data) > len(high_res_data):
            remaining_data = image_data[~image_data['has_high_res']].
↪sample(min(remaining_count, len(image_data[~image_data['has_high_res']])))
            sample_data = pd.concat([high_res_data, remaining_data])
        else:
            sample_data = high_res_data
    else:
        sample_data = image_data.sample(min(limit, len(image_data)))

    for _, row in tqdm(sample_data.iterrows(), total=len(sample_data)):
        try:
            img_url = f"{row['iiifurl']}/full/1200,/0/default.jpg"

            file_path = os.path.join(IMAGE_DIR, f"{row['objectid']}.jpg")

            if not os.path.exists(file_path):
                response = requests.get(img_url)
                if response.status_code == 200:
                    img = Image.open(BytesIO(response.content))

                    if img.width < 300 or img.height < 300:
                        print(f"Skipping low-quality image {row['objectid']}")
                        continue

                    img.save(file_path)
                    images.append(file_path)
                    object_ids.append(row['objectid'])

                    meta = {
                        'objectid': row['objectid'],
                        'title': row['title'] if 'title' in row else '',
                        'artist': row['attribution'] if 'attribution' in row␣
↪else '',
                        'date': row['displaydate'] if 'displaydate' in row else␣
↪'',
                        'medium': row['medium'] if 'medium' in row else '',
                        'style': row.get('style_category', 'unknown')
                    }
                    metadata.append(meta)
            else:
                img = Image.open(file_path)
                if img.width >= 300 and img.height >= 300:
                    images.append(file_path)
```

```
                    object_ids.append(row['objectid'])

                    meta = {
                        'objectid': row['objectid'],
                        'title': row['title'] if 'title' in row else '',
                        'artist': row['attribution'] if 'attribution' in row␣
↪else '',
                        'date': row['displaydate'] if 'displaydate' in row else␣
↪'',
                        'medium': row['medium'] if 'medium' in row else '',
                        'style': row.get('style_category', 'unknown')
                    }
                    metadata.append(meta)

        except Exception as e:
            print(f"Error downloading {row['objectid']}: {e}")

    return images, object_ids, metadata
```

[6]:
```
def get_preprocess_transform(train=True):
    """
    Returns a preprocessing transform with data augmentation for training
    and standard preprocessing for validation/testing.
    """
    if train:
        return transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.
↪2, hue=0.1),
            transforms.RandomRotation(20),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,␣
↪0.225]),
        ])
    else:
        return transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,␣
↪0.225]),
        ])
```

[7]:
```
mtcnn = MTCNN(keep_all=True, device='cuda' if torch.cuda.is_available() else␣
↪'cpu')
```

```
def extract_features(image_paths, train=True, num_epochs=2):
    """
    Extract features using multiple models for better representation.
    If train=True, fine-tune the models for the given number of epochs.
    """
    models_list = {
        'resnet101': models.resnet101(pretrained=True),
        'efficientnet': models.efficientnet_b3(pretrained=True),
        'vit': models.vit_b_16(pretrained=True)
    }

    for name, model in models_list.items():
        if name == 'vit':
            models_list[name].heads = torch.nn.Identity()
        else:
            models_list[name] = torch.nn.Sequential(*list(model.children())[:
 ↪-1])

        models_list[name].eval()

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    for name in models_list:
        models_list[name] = models_list[name].to(device)

    preprocess = get_preprocess_transform(train=train)

    all_features = []

    scaler = StandardScaler()

    if train:
        feature_size = 4352
        num_classes = 2
        classifier = nn.Linear(feature_size, num_classes).to(device)

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(classifier.parameters(), lr=0.001)

        for epoch in range(num_epochs):
            print(f"Epoch {epoch + 1}/{num_epochs}")
            for img_path in tqdm(image_paths):
                try:
                    img = Image.open(img_path).convert('RGB')
                    boxes, _ = mtcnn.detect(img)
                    if boxes is not None:
                        x_min, y_min = np.min(boxes[:, 0]), np.min(boxes[:, 1])
```

```
                        x_max, y_max = np.max(boxes[:, 2]), np.max(boxes[:, 3])
                        img = img.crop((x_min, y_min, x_max, y_max))

                    img_tensor = preprocess(img).unsqueeze(0).to(device)

                    features_combined = []
                    with torch.no_grad():
                        for name, model in models_list.items():
                            feature = model(img_tensor)
                            if len(feature.shape) > 2:
                                feature = feature.reshape(feature.size(0), -1)
                            feature = feature.cpu().numpy()
                            features_combined.append(feature.squeeze())

                    combined = np.concatenate(features_combined)
                    combined = scaler.fit_transform(combined.reshape(-1, 1)).
↪flatten()
                    all_features.append(combined)

                    combined_tensor = torch.tensor(combined).unsqueeze(0).
↪to(device)

                    outputs = classifier(combined_tensor)
                    labels = torch.tensor([0]).to(device)  # Replace with␣
↪actual labels
                    loss = criterion(outputs, labels)

                    optimizer.zero_grad()
                    loss.backward()
                    optimizer.step()

                except Exception as e:
                    print(f"Error processing {img_path}: {e}")
                    if all_features:
                        zero_dim = all_features[0].shape[0]
                        all_features.append(np.zeros(zero_dim))
                    else:
                        print("Failed to process first image, cannot determine␣
↪feature dimension")
                        raise

    else:
        for img_path in tqdm(image_paths):
            try:
                img = Image.open(img_path).convert('RGB')
                boxes, _ = mtcnn.detect(img)
                if boxes is not None:
```

```
                    x_min, y_min = np.min(boxes[:, 0]), np.min(boxes[:, 1])
                    x_max, y_max = np.max(boxes[:, 2]), np.max(boxes[:, 3])
                    img = img.crop((x_min, y_min, x_max, y_max))

                img_tensor = preprocess(img).unsqueeze(0).to(device)

                features_combined = []
                with torch.no_grad():
                    for name, model in models_list.items():
                        feature = model(img_tensor)
                        if len(feature.shape) > 2:
                            feature = feature.reshape(feature.size(0), -1)
                        feature = feature.cpu().numpy()
                        features_combined.append(feature.squeeze())

                combined = np.concatenate(features_combined)
                combined = scaler.fit_transform(combined.reshape(-1, 1)).
↪flatten()
                all_features.append(combined)

            except Exception as e:
                print(f"Error processing {img_path}: {e}")
                if all_features:
                    zero_dim = all_features[0].shape[0]
                    all_features.append(np.zeros(zero_dim))
                else:
                    print("Failed to process first image, cannot determine␣
↪feature dimension")
                    raise

    features_array = np.array(all_features)
    return features_array
```

[8]:
```
def optimize_features(features, n_components=512):
    """
    Apply PCA and normalization to optimize features
    """
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(features)

    pca = PCA(n_components=n_components)
    features_pca = pca.fit_transform(features_scaled)

    print(f"Explained variance ratio sum: {sum(pca.explained_variance_ratio_):.
↪4f}")

    return features_pca, scaler, pca
```

[9]:
```python
def build_similarity_index(features, metadata=None):
    """
    Build a more advanced FAISS index with metadata support
    """
    features = features.astype('float32')
    faiss.normalize_L2(features)

    d = features.shape[1]
    nlist = min(100, features.shape[0] // 10)

    kmeans = faiss.Kmeans(d, nlist, niter=20, verbose=True)
    kmeans.train(features)

    quantizer = faiss.IndexFlatIP(d)
    index = faiss.IndexIVFFlat(quantizer, d, nlist, faiss.METRIC_INNER_PRODUCT)

    index.train(features)
    index.add(features)

    index_with_meta = {
        'index': index,
        'metadata': metadata
    }

    return index_with_meta
```

[10]:
```python
def find_similar_images(index_with_meta, features, query_idx, metadata=None,␣
↪k=5):
    """
    Find similar images with metadata-enhanced similarity using cosine␣
↪similarity.
    """
    index = index_with_meta['index']
    meta_list = index_with_meta['metadata']

    query_vector = features[query_idx].reshape(1, -1).astype('float32')
    faiss.normalize_L2(query_vector)

    k_search = min(k * 3, features.shape[0])
    D, I = index.search(query_vector, k_search)

    candidate_vectors = features[I[0]]
    cos_sim = cosine_similarity(query_vector, candidate_vectors)[0]

    if meta_list and metadata:
        query_meta = meta_list[query_idx]
```

```
        combined_scores = []
        for i, idx in enumerate(I[0]):
            if idx < len(meta_list):
                candidate_meta = meta_list[idx]

                meta_sim = calculate_metadata_similarity(query_meta,␣
↪candidate_meta)

                combined_score = 0.7 * cos_sim[i] + 0.3 * meta_sim
                combined_scores.append((idx, combined_score))

        combined_scores.sort(key=lambda x: x[1], reverse=True)

        top_indices = [idx for idx, _ in combined_scores[:k]]
        top_scores = [score for _, score in combined_scores[:k]]
    else:
        top_indices = np.argsort(-cos_sim)[:k]
        top_scores = cos_sim[top_indices]

    return top_scores, top_indices

def calculate_metadata_similarity(meta1, meta2):
    """
    Calculate similarity between two metadata entries
    """
    similarity = 0.0

    if 'style' in meta1 and 'style' in meta2 and meta1['style'] ==␣
↪meta2['style']:
        similarity += 0.4

    if 'artist' in meta1 and 'artist' in meta2 and meta1['artist'] ==␣
↪meta2['artist']:
        similarity += 0.3

    if 'medium' in meta1 and 'medium' in meta2:
        if meta1['medium'] == meta2['medium']:
            similarity += 0.2
        elif isinstance(meta1['medium'], str) and isinstance(meta2['medium'],␣
↪str):
            # Partial match
            if any(term in meta2['medium'].lower() for term in meta1['medium'].
↪lower().split()):
                similarity += 0.1

    if 'date' in meta1 and 'date' in meta2:
        meta1_period = extract_period(meta1['date'])
```

```
        meta2_period = extract_period(meta2['date'])
        if meta1_period and meta2_period and abs(meta1_period - meta2_period) <␣
 ↪50:
            similarity += 0.1

    return similarity

def extract_period(date_str):
    """
    Extract a numeric year from a date string
    """
    if not isinstance(date_str, str):
        return None

    import re
    years = re.findall(r'\b(1[4-9]\d\d|20[0-2]\d)\b', date_str)
    if years:
        return int(years[0])
    return None
```

[11]:
```
def visualize_similar_images(query_idx, similar_indices, image_paths,␣
 ↪object_ids, metadata=None):
    """
    Visualize the query image and its similar images with metadata
    """
    plt.figure(figsize=(20, 15))

    plt.subplot(2, 3, 1)
    query_img = Image.open(image_paths[query_idx])
    plt.imshow(query_img)
    title = f"Query: {object_ids[query_idx]}"
    if metadata and query_idx < len(metadata):
        meta = metadata[query_idx]
        if 'title' in meta and meta['title']:
            title += f"\n{meta['title']}"
        if 'artist' in meta and meta['artist']:
            title += f"\nArtist: {meta['artist']}"
        if 'style' in meta and meta['style']:
            title += f"\nStyle: {meta['style']}"

    plt.title(title, fontsize=10)
    plt.axis('off')

    for i, idx in enumerate(similar_indices):
        plt.subplot(2, 3, i+2)
        similar_img = Image.open(image_paths[idx])
        plt.imshow(similar_img)
```

```
        title = f"Similar: {object_ids[idx]}"
        if metadata and idx < len(metadata):
            meta = metadata[idx]
            if 'title' in meta and meta['title']:
                title += f"\n{meta['title']}"
            if 'artist' in meta and meta['artist']:
                title += f"\nArtist: {meta['artist']}"
            if 'style' in meta and meta['style']:
                title += f"\nStyle: {meta['style']}"

        plt.title(title, fontsize=10)
        plt.axis('off')

    plt.tight_layout()
    plt.savefig(os.path.join(BASE_DIR, 'similar_images.png'))
    plt.show()
```

[12]:
```
def convert_np_types(obj):
    if isinstance(obj, np.ndarray):
        return obj.tolist()
    elif isinstance(obj, np.float32):
        return float(obj)
    elif isinstance(obj, dict):
        return {key: convert_np_types(value) for key, value in obj.items()}
    elif isinstance(obj, list):
        return [convert_np_types(item) for item in obj]
    return obj

def evaluate_model(index_with_meta, features, metadata=None, k=5,␣
 ↪save_dir='results'):
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)

    precision_at_k = []
    recall_at_k = []
    average_precision = []
    rmse_scores = []
    ssim_scores = []

    if metadata:
        clusters = {}
        for i, meta in enumerate(metadata):
            key = meta.get('artist') or meta.get('style') or meta.get('medium')␣
 ↪or 'unknown'
            if key not in clusters:
                clusters[key] = []
```

```
            clusters[key].append(i)

        valid_clusters = {key: indices for key, indices in clusters.items() if␣
↪len(indices) >= 2}

        import random
        eval_indices = []
        for key, indices in valid_clusters.items():
            if len(indices) > 5:
                eval_indices.extend(random.sample(indices, min(5,␣
↪len(indices))))
        if not eval_indices:
            eval_indices = random.sample(range(len(features)), min(100,␣
↪len(features)))
    else:
        import random
        eval_indices = random.sample(range(len(features)), min(100,␣
↪len(features)))

    for i in eval_indices:
        D, I = find_similar_images(index_with_meta, features, i, metadata, k)

        if metadata and i in [idx for cluster in valid_clusters.values() for␣
↪idx in cluster]:
            relevant_cluster = next((indices for indices in valid_clusters.
↪values() if i in indices), None)
            relevant = set(relevant_cluster) if relevant_cluster else set()
            relevant.discard(i)
        else:
            relevant = set(range(max(0, i - k // 2), min(len(features), i + k //
↪ 2 + 1)))
            relevant.discard(i)

        retrieved = set(I)

        precision_at_k.append(len(relevant.intersection(retrieved)) /␣
↪len(retrieved) if retrieved else 0)

        recall_at_k.append(len(relevant.intersection(retrieved)) /␣
↪len(relevant) if relevant else 0)

        ap = 0
        relevant_count = 0
        for j in range(len(I)):
            if I[j] in relevant:
                relevant_count += 1
```

```
                ap += relevant_count / (j + 1)
        average_precision.append(ap / relevant_count if relevant_count else 0)

        query_image = features[i].reshape(-1)  # Flatten the query image for␣
↪RMSE
        rmse_vals = []
        ssim_vals = []

        for idx in I:
            retrieved_image = features[idx].reshape(-1)
            rmse_vals.append(mean_squared_error(query_image, retrieved_image,␣
↪squared=False))  # RMSE
            ssim_vals.append(ssim(features[i], features[idx],␣
↪data_range=features[idx].max() - features[idx].min()))  # SSIM

        rmse_scores.append(np.mean(rmse_vals))
        ssim_scores.append(np.mean(ssim_vals))

    mean_precision_at_k = np.mean(precision_at_k)
    mean_recall_at_k = np.mean(recall_at_k)
    mean_average_precision = np.mean(average_precision)
    mean_rmse = np.mean(rmse_scores)
    mean_ssim = np.mean(ssim_scores)

    print(f"Precision@{k}: {mean_precision_at_k:.4f}")
    print(f"Recall@{k}: {mean_recall_at_k:.4f}")
    print(f"Mean Average Precision: {mean_average_precision:.4f}")
    print(f"Mean RMSE: {mean_rmse:.4f}")
    print(f"Mean SSIM: {mean_ssim:.4f}")

    results = {
        'precision_at_k': mean_precision_at_k,
        'recall_at_k': mean_recall_at_k,
        'mean_average_precision': mean_average_precision,
        'mean_rmse': mean_rmse,
        'mean_ssim': mean_ssim
    }

    results_converted = convert_np_types(results)

    with open(os.path.join(save_dir, 'evaluation_results.json'), 'w') as f:
        json.dump(results_converted, f, indent=4)

    plt.figure(figsize=(8, 6))
    plt.plot(range(len(precision_at_k)), precision_at_k, label='Precision@K',␣
↪marker='o')
    plt.xlabel('Sample Index')
```

```
    plt.ylabel('Precision@K')
    plt.title(f'Precision@{k}')
    plt.legend()
    plt.savefig(os.path.join(save_dir, f'precision_at_{k}.png'))
    plt.close()

    plt.figure(figsize=(8, 6))
    plt.plot(range(len(recall_at_k)), recall_at_k, label='Recall@K',␣
↪marker='o', color='orange')
    plt.xlabel('Sample Index')
    plt.ylabel('Recall@K')
    plt.title(f'Recall@{k}')
    plt.legend()
    plt.savefig(os.path.join(save_dir, f'recall_at_{k}.png'))
    plt.close()

    plt.figure(figsize=(8, 6))
    plt.plot(range(len(average_precision)), average_precision, label='Average␣
↪Precision', marker='o', color='green')
    plt.xlabel('Sample Index')
    plt.ylabel('Average Precision')
    plt.title(f'Mean Average Precision (mAP)')
    plt.legend()
    plt.savefig(os.path.join(save_dir, f'map_{k}.png'))
    plt.close()

    plt.figure(figsize=(8, 6))
    plt.plot(range(len(rmse_scores)), rmse_scores, label='RMSE', marker='o',␣
↪color='red')
    plt.xlabel('Sample Index')
    plt.ylabel('RMSE')
    plt.title('Root Mean Square Error (RMSE)')
    plt.legend()
    plt.savefig(os.path.join(save_dir, 'rmse.png'))
    plt.close()

    plt.figure(figsize=(8, 6))
    plt.plot(range(len(ssim_scores)), ssim_scores, label='SSIM', marker='o',␣
↪color='purple')
    plt.xlabel('Sample Index')
    plt.ylabel('SSIM')
    plt.title('Structural Similarity Index (SSIM)')
    plt.legend()
    plt.savefig(os.path.join(save_dir, 'ssim.png'))
    plt.close()
```

```
    return mean_precision_at_k, mean_recall_at_k, mean_average_precision,␣
↪mean_rmse, mean_ssim
```

[13]:
```
face_detector = MTCNN(keep_all=True, device='cuda' if torch.cuda.is_available()␣
↪else 'cpu')

def find_similar_images_for_user_input(index_with_meta, features, image_paths,␣
↪object_ids, metadata, user_image_path, pca=None, scaler=None,␣
↪face_crop=False):
    """
    Find similar images for a user-provided image with similarity assessment.
    """
    models_list = {
        'resnet101': models.resnet101(pretrained=True),
        'efficientnet': models.efficientnet_b3(pretrained=True),
        'vit': models.vit_b_16(pretrained=True)
    }

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    for name, model in models_list.items():
        if name == 'vit':
            models_list[name].head = torch.nn.Sequential()
        else:
            models_list[name] = torch.nn.Sequential(*list(model.children())[:
↪-1])
        models_list[name].eval().to(device)

    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225]),
    ])

    try:
        img = Image.open(user_image_path).convert('RGB')
        if face_crop:
            boxes, _ = face_detector.detect(img)
            if boxes is not None:
                x, y, x2, y2 = boxes[0]
                img = img.crop((x, y, x2, y2))

        img_tensor = preprocess(img).unsqueeze(0).to(device)

        features_combined = []
        with torch.no_grad():
```

```
            for name, model in models_list.items():
                feature = model(img_tensor)
                if len(feature.shape) > 2:
                    feature = feature.view(feature.size(0), -1)
                features_combined.append(feature.cpu().numpy().squeeze())

        user_feature = np.concatenate(features_combined).reshape(1, -1)

        if scaler and pca:
            user_feature = scaler.transform(user_feature)
            user_feature = pca.transform(user_feature)

        faiss.normalize_L2(user_feature)

        index = index_with_meta['index']
        D, I = index.search(user_feature, 5)  # K = 5 (top 5 neighbors)

        user_meta = {
            'objectid': 'User',
            'title': 'User Image',
            'artist': '',
            'date': '',
            'medium': '',
            'style': ''
        }

        paths_to_show = [user_image_path]
        ids_to_show = ['User Image']
        meta_to_show = [user_meta]

        for idx in I[0]:
            paths_to_show.append(image_paths[idx])
            ids_to_show.append(object_ids[idx])
            if metadata and idx < len(metadata):
                meta_to_show.append(metadata[idx])
            else:
                meta_to_show.append(None)

        plt.figure(figsize=(20, 15))

        plt.subplot(2, 3, 1)
        plt.imshow(img)
        plt.title("User Query Image", fontsize=10)
        plt.axis('off')

        for i, idx in enumerate(I[0]):
            plt.subplot(2, 3, i + 2)
```

```
            similar_img = Image.open(image_paths[idx])
            plt.imshow(similar_img)

            title = f"Similar: {object_ids[idx]}"
            if metadata and idx < len(metadata):
                meta = metadata[idx]
                if 'title' in meta and meta['title']:
                    title += f"\n{meta['title']}"
                if 'artist' in meta and meta['artist']:
                    title += f"\nArtist: {meta['artist']}"
                if 'style' in meta and meta['style']:
                    title += f"\nStyle: {meta['style']}"

            plt.title(title, fontsize=10)
            plt.axis('off')

        plt.tight_layout()
        plt.savefig(os.path.join(BASE_DIR, 'user_query_results.png'))
        plt.show()

    except UnidentifiedImageError:
        print(f"Error: Cannot open image file {user_image_path}")
    except Exception as e:
        print(f"Error processing user image: {e}")
```

[ ]:
```
def main():
    print("Loading data...")
    painting_images = load_data()
    print(f"Found {len(painting_images)} paintings with images")

    print("Downloading images...")
    image_paths, object_ids, metadata = download_images(painting_images,␣
 ↪limit=10000)
    print(f"Downloaded {len(image_paths)} images")

    print("Extracting features using multiple models...")
    features = extract_features(image_paths)
    print(f"Extracted raw features with shape {features.shape}")

    print("Optimizing features...")
    features_optimized, scaler, pca = optimize_features(features,␣
 ↪n_components=512)
    print(f"Optimized features to shape {features_optimized.shape}")

    print("Building advanced similarity index...")
    index_with_meta = build_similarity_index(features_optimized, metadata)
```

```
    print("Evaluating improved model...")
    precision, recall, map_score, rmse, ssim = evaluate_model(index_with_meta,␣
 ↪features_optimized, metadata, k=5)

    query_idx = np.random.randint(0, len(image_paths))
    print(f"Finding similar images for query index {query_idx}...")
    D, I = find_similar_images(index_with_meta, features_optimized, query_idx,␣
 ↪metadata, k=5)

    print("Visualizing results...")
    visualize_similar_images(query_idx, I, image_paths, object_ids, metadata)

    user_image_path="r_image.jpg"  # Replace with the actual path
    if os.path.exists(user_image_path):
        print(f"Finding similar images for user-provided image:␣
 ↪{user_image_path}")
        find_similar_images_for_user_input(
            index_with_meta, features_optimized, image_paths,
            object_ids, metadata, user_image_path, pca, scaler
        )
    else:
        print("User image not found. Please provide a valid path.")

    return painting_images, image_paths, object_ids, features_optimized,␣
 ↪index_with_meta, metadata

if __name__ == "__main__":
    main()
```

```
Loading data…

/tmp/ipykernel_2009/3640074436.py:8: DtypeWarning: Columns (23,29) have mixed
types. Specify dtype option on import or set low_memory=False.
  objects_df = pd.read_csv(os.path.join(DATA_DIR, 'objects.csv'))

Found 6259 paintings with images
Downloading images…
Using 6254 existing images and downloading 5 new images

 11%|          | 688/6259 [01:02<06:56, 13.38it/s]

Skipping low-quality image 57679

 36%|        | 2282/6259 [02:17<01:29, 44.29it/s]

Skipping low-quality image 46169

 39%|        | 2425/6259 [02:20<01:23, 45.84it/s]

Skipping low-quality image 41589

 66%|       | 4159/6259 [02:37<00:13, 151.70it/s]
```

```
Skipping low-quality image 107683

 89%|      | 5556/6259 [02:41<00:00, 786.70it/s]

Skipping low-quality image 60330

100%|     | 6259/6259 [02:41<00:00, 38.65it/s]
/home/zeus/miniconda3/envs/cloudspace/lib/python3.10/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/zeus/miniconda3/envs/cloudspace/lib/python3.10/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet101_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet101_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)

Downloaded 6098 images
Extracting features using multiple models…

/home/zeus/miniconda3/envs/cloudspace/lib/python3.10/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=EfficientNet_B3_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B3_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
/home/zeus/miniconda3/envs/cloudspace/lib/python3.10/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ViT_B_16_Weights.IMAGENET1K_V1`. You can also use
`weights=ViT_B_16_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)

Epoch 1/2

100%|     | 6098/6098 [36:29<00:00,  2.79it/s]

Epoch 2/2

100%|     | 6098/6098 [33:40<00:00,  3.02it/s]

Extracted raw features with shape (12196, 4352)
Optimizing features…
Explained variance ratio sum: 0.7878
Optimized features to shape (12196, 512)
Building advanced similarity index…
Clustering 12196 points in 512D to 100 clusters, redo 1 times, 20 iterations
  Preprocessing in 0.00 s
```

```
  Iteration 19 (0.53 s, search 0.48 s): objective=8058.33 imbalance=1.097
nsplit=0
Evaluating improved model…
Precision@5: 0.1223
Recall@5: 0.0447
Mean Average Precision: 0.1715
Mean RMSE: 1.9859
Mean SSIM: 0.3414
Finding similar images for query index 3683…
Visualizing results…
```

[ ]:

[ ]:

[ ]:

[ ]: