[8]:
```
import os
import random
import pickle
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
```

[9]:
```
# ----------------------------------------
# Device Setup
# ----------------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"[Device] Using: {device}")
```

```
[Device] Using: cuda
```

*[10]:
```
# ----------------------------------------
# Paths
# ----------------------------------------
repolyvore_root = "Re-polyvore"            # Root folder for Re-polyvore dataset
dress_folder = os.path.join(repolyvore_root, "dress")   # Folder with dress images
archive_folder = "archive/images"          # Archive containing full outfit sets (organized by set ID)
query_image_path = "query.jpg"             # Path for a single query image demonstration
queries_folder = "query"             # Folder containing multiple query images for evaluation

categories = [
    'bag', 'bracelet', 'brooch', 'dress', 'earrings', 'eyewear', 'gloves',
    'hairwear', 'hats', 'jumpsuit', 'legwear', 'necklace', 'neckwear',
    'outwear', 'pants', 'rings', 'shoes', 'skirt', 'top', 'watches'
]

# ----------------------------------------
# Image Preprocessing Transform
# ----------------------------------------
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

# ----------------------------------------
# Color Histogram Function using PIL / NumPy
# ----------------------------------------
def get_color_histogram(img_path, bins=32):
    # Open image, convert to RGB, and resize to 128x128
    img = Image.open(img_path).convert("RGB").resize((128, 128))
    # Split channels
    r, g, b = img.split()
    r_hist = np.histogram(np.array(r).flatten(), bins=bins, range=(0, 256))[0]
    g_hist = np.histogram(np.array(g).flatten(), bins=bins, range=(0, 256))[0]
    b_hist = np.histogram(np.array(b).flatten(), bins=bins, range=(0, 256))[0]
    hist = np.concatenate([r_hist, g_hist, b_hist]).astype(np.float32)  # Total dim = 3*bins (e.g., 96)
    hist /= (np.linalg.norm(hist) + 1e-8)  # L2 normalization
    return hist

# ----------------------------------------
# Model: Full Compatibility Network
# ----------------------------------------
class FullCompatibilityNet(nn.Module):
    def __init__(self, hist_size=96):
        super().__init__()
        # Siamese (shared) embedding network
        self.embedding_net = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((1, 1))
        )
        self.projection = nn.Linear(128, 128)
```

```
        # Metric sub-network to learn a non-linear compatibility function
        self.metric_net = nn.Sequential(
            nn.Linear(128 + hist_size, 256),
            nn.ReLU(),
            nn.Linear(256, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Sigmoid()  # Compatibility score between 0 and 1
        )

    def extract_embedding(self, x):
        x = self.embedding_net(x).view(x.size(0), -1)  # Flatten CNN output
        return self.projection(x)  # Return 128-d embedding

    def forward(self, x1, x2, h1, h2):
        # x1, x2: image tensors; h1, h2: corresponding color histograms of shape [B, hist_size]
        e1 = self.extract_embedding(x1)  # Embedding for image1
        e2 = self.extract_embedding(x2)  # Embedding for image2
        merged_feat = e1 * e2            # Hadamard (elementwise) product of embeddings
        merged_hist = h1 * h2            # Hadamard product of color histograms
        # Concatenate the two merged vectors
        combined = torch.cat([merged_feat, merged_hist], dim=1)
        # Pass through metric network to get compatibility score
        score = self.metric_net(combined)
        return score

model = FullCompatibilityNet().to(device)

# ----------------------------------------
# Dataset: Compatibility Dataset with Color Histograms
# ----------------------------------------
class CompatibilityDataset(Dataset):
    def __init__(self, root_folder, transform=None):
        self.transform = transform
        self.data = []     # List of image file paths
        self.labels = []   # Their category labels

        # Allowed categories for training (adjust as needed)
        allowed_categories = {'dress', 'shoes', 'pants'}
        for category in os.listdir(root_folder):
            if category not in allowed_categories:
                continue
            cat_path = os.path.join(root_folder, category)
            if os.path.isdir(cat_path):
                for fname in os.listdir(cat_path):
                    if fname.lower().endswith('.jpg'):
                        self.data.append(os.path.join(cat_path, fname))
                        self.labels.append(category)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        img1_path = self.data[index]
        label1 = self.labels[index]

        # Randomly form a positive pair (same category) or negative pair (different category)
        if random.random() < 0.5:
            same_indices = [i for i, lbl in enumerate(self.labels) if lbl == label1 and i != index]
            idx2 = random.choice(same_indices) if same_indices else index
            pair_label = 1.0  # Compatible
        else:
            diff_indices = [i for i, lbl in enumerate(self.labels) if lbl != label1]
            idx2 = random.choice(diff_indices)
            pair_label = 0.0  # Incompatible

        img2_path = self.data[idx2]

        # Load images using PIL
        img1 = Image.open(img1_path).convert("RGB")
        img2 = Image.open(img2_path).convert("RGB")
        if self.transform:
            img1 = self.transform(img1)
            img2 = self.transform(img2)

        # Compute color histograms for the images
        hist1 = torch.tensor(get_color_histogram(img1_path), dtype=torch.float32)
        hist2 = torch.tensor(get_color_histogram(img2_path), dtype=torch.float32)

        return img1, img2, hist1, hist2, torch.tensor([pair_label], dtype=torch.float32)

# ----------------------------------------
# Training Function
# ----------------------------------------
def train_model(model, dataset_folder, epochs=10, batch_size=32):
    dataset = CompatibilityDataset(dataset_folder, transform)
    loader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=0)
    optimizer = optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.BCELoss()  # Binary cross-entropy loss on compatibility score
    model.train()
    for epoch in range(epochs):
        total_loss = 0
        for img1, img2, h1, h2, label in tqdm(loader, desc=f"Epoch {epoch+1}/{epochs}"):
            img1, img2 = img1.to(device), img2.to(device)
```

```
            h1, h2 = h1.to(device), h2.to(device)
            label = label.to(device)

            output = model(img1, img2, h1, h2)
            loss = criterion(output, label)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f"Epoch {epoch+1} - Avg Loss: {total_loss/len(loader):.4f}")
    torch.save(model.state_dict(), "compatibility_model.pth")
    print("Training complete. Model saved!")

# ----------------------------------------
# Inference: Compute Compatibility Score between Two Images
# ----------------------------------------
def compute_compatibility_score(img1_path, img2_path):
    try:
        img1 = Image.open(img1_path).convert("RGB")
        img2 = Image.open(img2_path).convert("RGB")
        img1_tensor = transform(img1).unsqueeze(0).to(device)
        img2_tensor = transform(img2).unsqueeze(0).to(device)
        h1 = torch.tensor(get_color_histogram(img1_path), dtype=torch.float32).unsqueeze(0).to(device)
        h2 = torch.tensor(get_color_histogram(img2_path), dtype=torch.float32).unsqueeze(0).to(device)
        model.eval()
        with torch.no_grad():
            score = model(img1_tensor, img2_tensor, h1, h2).item()
        return score
    except Exception as e:
        print(f"Error computing compatibility: {e}")
        return 0.0

# ----------------------------------------
# Retrieval: Embedding Extraction for Similarity Search
# (Uses the embedding branch of our network)
# ----------------------------------------
def extract_embedding(img_path):
    try:
        img = Image.open(img_path).convert("RGB")
        img_tensor = transform(img).unsqueeze(0).to(device)
        model.eval()
        with torch.no_grad():
            # Use the embedding branch (and projection) to extract a 128-dim vector
            embedding = model.extract_embedding(img_tensor).squeeze().cpu().numpy()
        return embedding
    except Exception as e:
        print(f"Error extracting embedding from {img_path}: {e}")
        return None

def load_dress_embeddings(folder):
    embeddings = {}
    for fname in tqdm(os.listdir(folder), desc="Extracting dress embeddings"):
        if fname.lower().endswith(('.jpg', '.jpeg', '.png')):
            path = os.path.join(folder, fname)
            emb = extract_embedding(path)
            if emb is not None:
                embeddings[fname] = emb
    return embeddings

def find_similar_images(query_image, embeddings, top_n=5):
    query_emb = extract_embedding(query_image)
    if query_emb is None:
        print("Query image could not be processed.")
        return []
    names = list(embeddings.keys())
    emb_matrix = np.array([embeddings[n] for n in names])
    sims = cosine_similarity([query_emb], emb_matrix).flatten()
    top_indices = sims.argsort()[::-1][:top_n]
    return [(names[i], sims[i]) for i in top_indices]

# ----------------------------------------
# Display Function: Show Query, Archive, and Repolyvore Results
# ----------------------------------------
def display_results(query_image, similar_results, archive_root, repolyvore_root, categories):
    query_img = Image.open(query_image).convert('RGB')
    plt.figure(figsize=(4, 5))
    plt.imshow(query_img)
    plt.axis('off')
    plt.title(f"Query: {os.path.basename(query_image)}")
    plt.show()

    for filename, similarity in similar_results:
        set_id = filename.split('_')[0]

        # ARCHIVE DISPLAY
        archive_set = os.path.join(archive_root, set_id)
        if os.path.isdir(archive_set):
            files = sorted([f for f in os.listdir(archive_set) if f.endswith('.jpg')])
            fig, axes = plt.subplots(1, len(files), figsize=(4 * len(files), 5))
            if len(files) == 1:
                axes = [axes]
            for ax, f in zip(axes, files):
                img = Image.open(os.path.join(archive_set, f)).convert('RGB')
```

```
                ax.imshow(img)
                ax.set_title(f, fontsize=9)
                ax.axis('off')
            plt.suptitle(f"Archive Set: {set_id} | Similarity: {similarity:.2f}")
            plt.show()

        # REPOLYVORE COMPLEMENTARY SET DISPLAY
        fig, axes = plt.subplots(4, 5, figsize=(15, 10))
        axes = axes.flatten()
        idx = 0
        for cat in categories:
            cat_path = os.path.join(repolyvore_root, cat)
            if not os.path.exists(cat_path):
                continue
            match = next((f for f in os.listdir(cat_path) if f.startswith(set_id + '_')), None)
            if match:
                img = Image.open(os.path.join(cat_path, match)).convert('RGB')
                axes[idx].imshow(img)
                axes[idx].set_title(cat)
                axes[idx].axis('off')
                idx += 1
        for i in range(idx, len(axes)):
            axes[i].axis('off')
        plt.suptitle(f"Repolyvore Set: {set_id}")
        plt.show()

# ----------------------------------------
# Evaluation: Top-N Accuracy on a Query Folder
# ----------------------------------------
def get_set_id(filename):
    return filename.split('_')[0]

def evaluate_on_query_folder(query_folder, embeddings, top_n=5):
    queries = [f for f in os.listdir(query_folder) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
    total = 0
    correct = 0
    for query_file in tqdm(queries, desc="Evaluating Queries"):
        query_path = os.path.join(query_folder, query_file)
        query_set_id = get_set_id(query_file)
        similar_results = find_similar_images(query_path, embeddings, top_n=top_n)
        top_set_ids = [get_set_id(fname) for fname, _ in similar_results]
        if query_set_id in top_set_ids:
            correct += 1
        total += 1
        print(f"\nQuery: {query_file} | Correct Set ID: {query_set_id}")
        print("Top Matches Set IDs:", top_set_ids)
    accuracy = correct / total if total > 0 else 0
    print(f"\n Top-{top_n} Accuracy on {total} queries: {accuracy * 100:.2f}%")
```

[10]:

```
# ----------------------------------------
# Main Execution Block
# ----------------------------------------
if __name__ == "__main__":
    print("[Training the Model]")

    # # Optional: Load saved model weights if available
    # if os.path.exists("compatibility_model.pth"):
    #     model.load_state_dict(torch.load("compatibility_model.pth", map_location=device))
    #     print("[Model Loaded from compatibility_model.pth]")
    # else:
    #     print("[Warning] Trained model not found. Please run training first.")
    #     # Uncomment below to run training:
        # train_model(model, repolyvore_root, epochs=10, batch_size=32)
    train_model(model, repolyvore_root, epochs=10, batch_size=32)

    # For retrieval, extract embeddings from the dress folder
    print("\nExtracting dress embeddings for retrieval...")
    dress_embeddings = load_dress_embeddings(dress_folder)

    # Evaluate retrieval accuracy on all query images in queries_folder
    print("\nEvaluating on query folder...")
    evaluate_on_query_folder(queries_folder, dress_embeddings, top_n=5)

    # Single query demonstration
    print("\nRunning compatibility search for a single query image...")
    similar_results = find_similar_images(query_image_path, dress_embeddings, top_n=5)
    print(f"Top {len(similar_results)} similar items:")
    for fname, sim in similar_results:
        print(f"{fname}: Cosine Similarity = {sim:.4f}")

    display_results(query_image_path, similar_results, archive_folder, repolyvore_root, categories)
```

```
[Training the Model]
Epoch 1/10: 100%|██████████| 1143/1143 [09:35<00:00,  1.98it/s]
Epoch 1 - Avg Loss: 0.5176
Epoch 2/10: 100%|██████████| 1143/1143 [09:15<00:00,  2.06it/s]
Epoch 2 - Avg Loss: 0.4071
Epoch 3/10: 100%|██████████| 1143/1143 [09:21<00:00,  2.04it/s]
```

```
Epoch 3 - Avg Loss: 0.3061
Epoch 4/10: 100%|██████████| 1143/1143 [09:40<00:00,  1.97it/s]
Epoch 4 - Avg Loss: 0.2289
```

[11]:
```python
import torch
import matplotlib.pyplot as plt
import seaborn as sns

def plot_similarity_score_distribution(model_path, dataset_folder, num_samples=1000):
    # Load model
    model = FullCompatibilityNet().to(device)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.eval()

    # Dataset for sampling pairs
    dataset = CompatibilityDataset(dataset_folder, transform=transform)

    positive_scores = []
    negative_scores = []

    # Sampling random pairs
    for _ in tqdm(range(num_samples), desc="Generating pairs for SSD"):
        idx1 = random.randint(0, len(dataset)-1)
        img1, _, h1, _, _ = dataset[idx1]
        label1 = dataset.labels[idx1]

        # Positive pair
        same_indices = [i for i, lbl in enumerate(dataset.labels) if lbl == label1 and i != idx1]
        if same_indices:
            idx2 = random.choice(same_indices)
            img2, _, h2, _, _ = dataset[idx2]
            with torch.no_grad():
                score = model(
                    img1.unsqueeze(0).to(device),
                    img2.unsqueeze(0).to(device),
                    h1.unsqueeze(0).to(device),
                    h2.unsqueeze(0).to(device)
                ).item()
                positive_scores.append(score)

        # Negative pair
        diff_indices = [i for i, lbl in enumerate(dataset.labels) if lbl != label1]
        if diff_indices:
            idx2 = random.choice(diff_indices)
            img2, _, h2, _, _ = dataset[idx2]
            with torch.no_grad():
                score = model(
                    img1.unsqueeze(0).to(device),
                    img2.unsqueeze(0).to(device),
                    h1.unsqueeze(0).to(device),
                    h2.unsqueeze(0).to(device)
                ).item()
                negative_scores.append(score)

    # Plotting the histograms
    plt.figure(figsize=(10, 6))
    sns.histplot(positive_scores, bins=50, color='green', label='Positive Pairs', kde=True)
    sns.histplot(negative_scores, bins=50, color='red', label='Negative Pairs', kde=True)
    plt.title("Similarity Score Distribution")
    plt.xlabel("Compatibility Score")
    plt.ylabel("Frequency")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

[13]:
```python
def evaluate_on_query_folder2(query_folder, embeddings, archive_root, repolyvore_root, categories, top_n=5):
    queries = [f for f in os.listdir(query_folder) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
    total = 0
    correct = 0
    for query_file in tqdm(queries, desc="Evaluating Queries"):
        query_path = os.path.join(query_folder, query_file)
        query_set_id = get_set_id(query_file)
        similar_results = find_similar_images(query_path, embeddings, top_n=top_n)
        top_set_ids = [get_set_id(fname) for fname, _ in similar_results]

        # Display query image
        query_img = Image.open(query_path).convert('RGB')
        plt.figure(figsize=(4, 5))
        plt.imshow(query_img)
        plt.axis('off')
        plt.title(f"Query: {query_file}")
        plt.show()

        for set_id in top_set_ids:
            # ARCHIVE DISPLAY
            archive_set = os.path.join(archive_root, set_id)
            if os.path.isdir(archive_set):
                files = sorted([f for f in os.listdir(archive_set) if f.endswith('.jpg')])
                fig, axes = plt.subplots(1, len(files), figsize=(4 * len(files), 5))
```

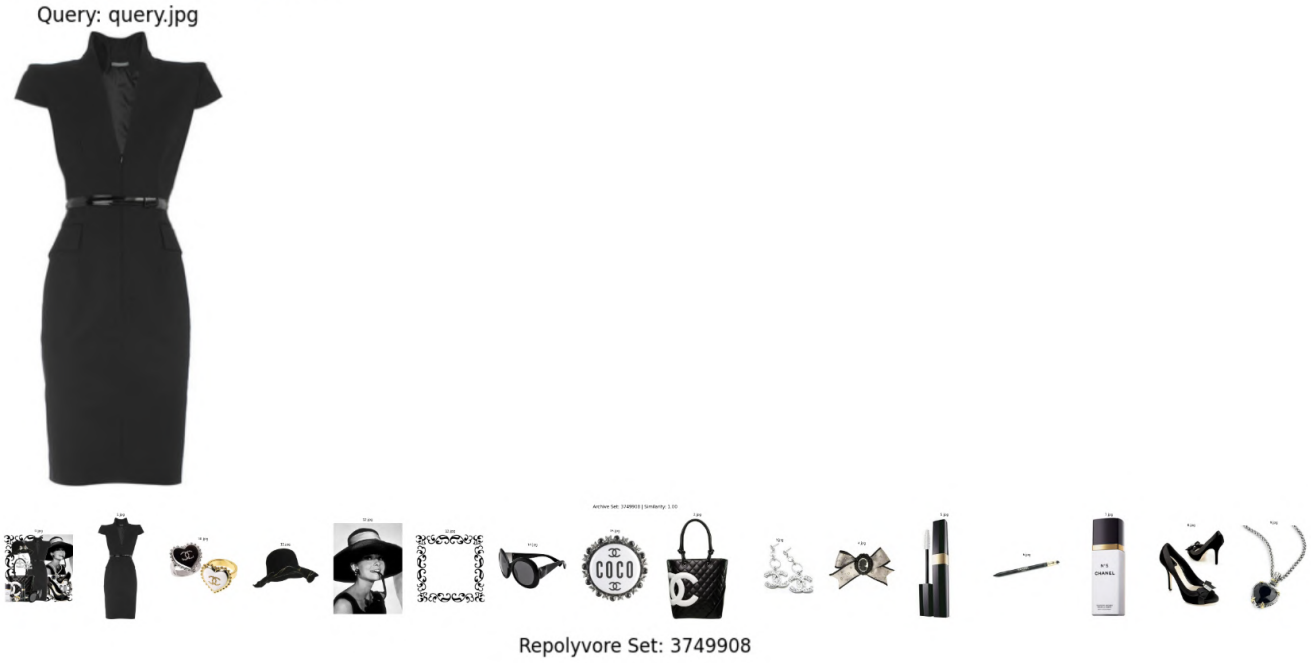
```
                if len(files) == 1:
                    axes = [axes]
                for ax, f in zip(axes, files):
                    img = Image.open(os.path.join(archive_set, f)).convert('RGB')
                    ax.imshow(img)
                    ax.set_title(f, fontsize=9)
                    ax.axis('off')
                plt.suptitle(f"Archive Set: {set_id}")
                plt.show()

            # REPOLYVORE COMPLEMENTARY SET DISPLAY
            fig, axes = plt.subplots(4, 5, figsize=(15, 10))
            axes = axes.flatten()
            idx = 0
            for cat in categories:
                cat_path = os.path.join(repolyvore_root, cat)
                if not os.path.exists(cat_path):
                    continue
                match = next((f for f in os.listdir(cat_path) if f.startswith(set_id + '_')), None)
                if match:
                    img = Image.open(os.path.join(cat_path, match)).convert('RGB')
                    axes[idx].imshow(img)
                    axes[idx].set_title(cat)
                    axes[idx].axis('off')
                    idx += 1
            for i in range(idx, len(axes)):
                axes[i].axis('off')
            plt.suptitle(f"Repolyvore Set: {set_id}")
            plt.show()

        if query_set_id in top_set_ids:
            correct += 1
        total += 1

    accuracy = correct / total if total > 0 else 0
    print(f"\n✅ Top-{top_n} Accuracy on {total} queries: {accuracy * 100:.2f}%")

# # Example usage:
# evaluate_on_query_folder2(queries_folder, dress_embeddings, archive_folder, repolyvore_root, categories, top_n=5)
```

[14]:

```
# ----------------------------------------
# Main Execution Block
# ----------------------------------------
if __name__ == "__main__":
    print("[Training the Model]")

    # Optional: Load saved model weights if available
    if os.path.exists("compatibility_model.pth"):
        model.load_state_dict(torch.load("compatibility_model.pth", map_location=device))
        print("[Model Loaded from compatibility_model.pth]")
    else:
        print("[Warning] Trained model not found. Please run training first.")
        # Uncomment below to run training:
        train_model(model, repolyvore_root, epochs=10, batch_size=32)

    # train_model(model, repolyvore_root, epochs=10, batch_size=32)

    # For retrieval, extract embeddings from the dress folder
    print("\nExtracting dress embeddings for retrieval...")
    dress_embeddings = load_dress_embeddings(dress_folder)

    # Evaluate retrieval accuracy on all query images in queries_folder
    print("\nEvaluating on query folder...")
    evaluate_on_query_folder(queries_folder, dress_embeddings, top_n=5)

    # Single query demonstration
    print("\nRunning compatibility search for a single query image...")
    similar_results = find_similar_images(query_image_path, dress_embeddings, top_n=5)
    print(f"Top {len(similar_results)} similar items:")
    for fname, sim in similar_results:
        print(f"{fname}: Cosine Similarity = {sim:.4f}")

    display_results(query_image_path, similar_results, archive_folder, repolyvore_root, categories)
    # Example usage:
    evaluate_on_query_folder2(queries_folder, dress_embeddings, archive_folder, repolyvore_root, categories, top_n=5)
```

```
[Training the Model]
[Model Loaded from compatibility_model.pth]

Extracting dress embeddings for retrieval...
Extracting dress embeddings: 100%|████████████████████████████████████████████████████████████████████████████████
████████████████| 7481/7481 [00:37<00:00, 201.86it/s]

Evaluating on query folder...
Evaluating Queries:  40%|████████████████████████████████████████████▌
| 4/10 [00:00<00:00, 36.99it/s]

Query: 10356637_1.jpg | Correct Set ID: 10356637
Top Matches Set IDs: ['10356637', '7756878', '11043457', '13674353', '16130570']

Query: 18834057_1.jpg | Correct Set ID: 18834057
Top Matches Set IDs: ['18834057', '74494247', '203954483', '14859946', '8357482']
```

```
Query: 27151658_7.jpg | Correct Set ID: 27151658
Top Matches Set IDs: ['27151658', '211138451', '141661298', '214874087', '109224400']

Query: 31914066_1.jpg | Correct Set ID: 31914066
Top Matches Set IDs: ['31914066', '181730027', '138894795', '137031841', '194951112']

Query: 4578741_5.jpg | Correct Set ID: 4578741
Top Matches Set IDs: ['4578741', '24029734', '196410810', '187720068', '184860264']

Query: 4701279_1.jpg | Correct Set ID: 4701279
Top Matches Set IDs: ['4701279', '163250936', '114098277', '14878014', '210445554']

Query: 5237351_1.jpg | Correct Set ID: 5237351
Top Matches Set IDs: ['5237351', '75633318', '94898865', '215452093', '215918638']

Query: 5598757_1.jpg | Correct Set ID: 5598757
Top Matches Set IDs: ['5598757', '142376183', '209229240', '187307106', '190127149']

Query: 6583861_1.jpg | Correct Set ID: 6583861
Evaluating Queries: 100%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 10/10 [00:00<00:00, 41.42it/s]

Top Matches Set IDs: ['6583861', '190289157', '158273036', '215184577', '30225138']

Query: 8401543_1.jpg | Correct Set ID: 8401543
Top Matches Set IDs: ['8401543', '6633755', '17155516', '27243035', '58455348']

 Top-5 Accuracy on 10 queries: 100.00%

Running compatibility search for a single query image...
Top 5 similar items:
3749908_1.jpg: Cosine Similarity = 1.0000
49836229_1.jpg: Cosine Similarity = 0.9944
140690022_1.jpg: Cosine Similarity = 0.9923
212796523_1.jpg: Cosine Similarity = 0.9915
60802716_1.jpg: Cosine Similarity = 0.9904
```



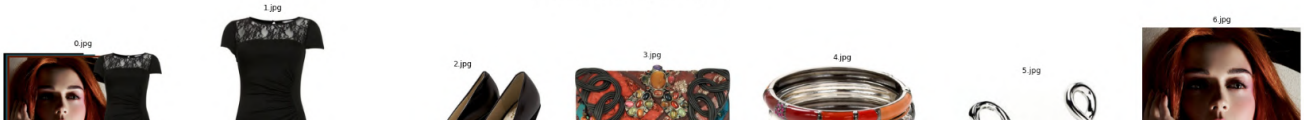
Repolyvore Set: 49836229

Repolyvore Set: 140690022

Archive Set: 212796523 | Similarity: 0.99

Repolyvore Set: 212796523

Archive Set: 60802716 | Similarity: 0.99



Repolyvore Set: 60802716


```
Evaluating Queries:   0%|
| 0/10 [00:00<?, ?it/s]
```

Query: 10356637_1.jpg

Repolyvore Set: 10356637





watches

Repolyvore Set: 7756878

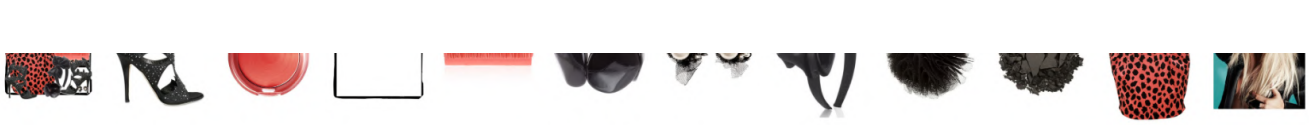
bag

dress

eyewear

pants

shoes



Repolyvore Set: 11043457

dress

earrings

hairwear

shoes

Repolyvore Set: 13674353

bag

brooch

dress

gloves

shoes

Repolyvore Set: 16130570


```
Evaluating Queries:  10%|
| 1/10 [00:08<01:18,  8.74s/it]
```

Query: 18834057_1.jpg

Repolyvore Set: 18834057

bag

dress

shoes

Repolyvore Set: 74494247

bag

dress

eyewear

outwear

rings

shoes

Repolyvore Set: 203954483

bag

dress

Repolyvore Set: 14859946

bag

bracelet

dress

necklace

skirt

Repolyvore Set: 8357482


```
Evaluating Queries:  20%|████████████████████████▌
| 2/10 [00:18<01:13,  9.19s/it]
```

Query: 27151658_7.jpg

Repolyvore Set: 27151658

bag

dress

Repolyvore Set: 211138451

bag

dress

earrings

shoes

Archive Set: 141661298

0.jpg

1.jpg

2.jpg

3.jpg

4.jpg

5.jpg

Repolyvore Set: 141661298

dress

outwear

Archive Set: 214874087

0.jpg

1.jpg

2.jpg

3.jpg

4.jpg

5.jpg

Repolyvore Set: 214874087

Repolyvore Set: 109224400

Evaluating Queries: 30%|
| 3/10 [00:26<01:00, 8.63s/it]

Query: 31914066_1.jpg

Repolyvore Set: 31914066

dress

pants

shoes

Archive Set: 181730027

Repolyvore Set: 181730027

bag

dress

Repolyvore Set: 138894795

bag

bracelet

dress

eyewear

outwear

skirt

Repolyvore Set: 137031841

bag

dress

eyewear

outwear

skirt

watches

Repolyvore Set: 194951112

bag

dress

eyewear

hats

outwear

shoes

Evaluating Queries: 40%|██
| 4/10 [00:33<00:48, 8.14s/it]

Query: 4578741_5.jpg

Archive Set: 4578741

Repolyvore Set: 4578741

bag

dress

earrings

shoes

Repolyvore Set: 24029734

bag

bracelet

dress

earrings

shoes

Repolyvore Set: 196410810

dress

shoes

Repolyvore Set: 187720068

Archive Set: 184860264

Repolyvore Set: 184860264


```
Evaluating Queries:  50%|
| 5/10 [00:39<00:37,  7.51s/it]
```

Query: 4701279_1.jpg

Repolyvore Set: 4701279

bag

dress

shoes

Repolyvore Set: 163250936

bag

dress

eyewear

rings

shoes

watches

Repolyvore Set: 114098277

dress

hats

shoes

Repolyvore Set: 14878014

bag

bracelet

dress

earrings

outwear

rings

Repolyvore Set: 210445554

dress

skirt

```
Evaluating Queries:  60%|
| 6/10 [00:47<00:30,  7.63s/it]
```

Query: 5237351_1.jpg

Repolyvore Set: 5237351

Repolyvore Set: 75633318

Repolyvore Set: 94898865

bag

dress

earrings

eyewear

necklace

shoes

Repolyvore Set: 215452093

bag

dress

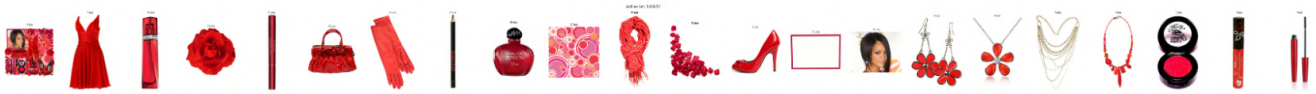
earrings

watches

Repolyvore Set: 215918638

Evaluating Queries: 70%|
| 7/10 [00:55<00:23, 7.76s/it]

Query: 5598757_1.jpg



Repolyvore Set: 5598757

bag

brooch

dress

earrings

gloves

necklace

neckwear

shoes

Archive Set: 142376183

Repolyvore Set: 142376183

bag

dress

outwear

shoes

Repolyvore Set: 209229240

Repolyvore Set: 187307106

dress

outwear

Archive Set: 190127149

Repolyvore Set: 190127149

bag

dress

outwear

shoes

```
Evaluating Queries:  80%|████████████████████████████████████████████████████████████████████████████████████████████████████████████
| 8/10 [01:03<00:15,  7.85s/it]
```

Query: 6583861_1.jpg

Repolyvore Set: 6583861

bag

bracelet

brooch

dress

earrings

outwear

watches

Repolyvore Set: 190289157

bag

dress

outwear

shoes

bag

Repolyvore Set: 158273036

bag

dress

eyewear

top

Repolyvore Set: 215184577

bag

brooch

dress

hairwear

legwear

outwear

Archive Set: 30225138

0.jpg

1.jpg

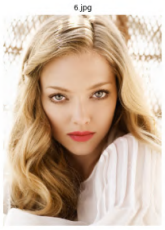
2.jpg

3.jpg

4.jpg

5.jpg

6.jpg



Repolyvore Set: 30225138

bag

dress

earrings

```
Evaluating Queries:  90%|██████████████████████████████████████████████████████████████████████▉        | 9/10 [01:13<00:08,  8.38s/it]
```

Query: 8401543_1.jpg

Repolyvore Set: 8401543

Repolyvore Set: 6633755

Repolyvore Set: 17155516

Repolyvore Set: 27243035

bag

dress

neckwear

outwear

Repolyvore Set: 58455348

bag

bracelet

dress

outwear

shoes

top