Image Similarity Model (RA at IIT Jodhpur)

By Khan Shah Ahmed Shakir Abu Asim

Overview

This project implements an image similarity model using a convolutional neural network (CNN) with a contrastive loss function. The model learns embeddings for images, where similar images have embeddings close to each other and dissimilar images have embeddings far apart. Performance is evaluated using metrics like mean Average Precision (mAP) and Mean Rank.

Project Structure

```
.
├── train/
│   ├── image1.jpg
│   ├── image2.jpg
│   └── ...
├── query_images/
│   ├── query1.jpg
│   ├── query2.jpg
│   └── ...
├── gallery/
│   ├── gallery1.jpg
│   ├── gallery2.jpg
│   └── ...
├── train_image_info.json
├── test_image_info.json
├── Shakir_IIT_Jodpur.ipynb
└── README.md
```

Dataset

- train/: Directory containing training images.
- query_images/: Directory containing query images.
- gallery/: Directory containing gallery images.
- train_image_info.json: JSON file mapping training image filenames to their labels.
- test_image_info.json: JSON file mapping query and gallery image filenames to their labels.

Dependencies

- Python 3.6+
- torch
- torchvision
- pillow
- tqdm

Dependencies can be installed using pip.

Image Transformations

Images are resized to 224x224 pixels, converted to tensors, and normalized to ensure uniform input for the model.

```
# Define the image transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Custom Dataset Class

A custom dataset class loads images and their corresponding labels from directories and JSON files.

```
# Defining ImageDataset Class
class ImageDataset(Dataset):
    def __init__(self, image_dir, info_file, transform=None):
        self.image_dir = image_dir
        self.image_paths = []
        self.labels = []
        self.label_to_idx = {}

        with open(info_file, 'r') as f:
            info = json.load(f)

        # Create a mapping from labels to integers
        unique_labels = set(info.values())
        for idx, label in enumerate(unique_labels):
            self.label_to_idx[label] = idx

        for image_path, label in info.items():
            self.image_paths.append(os.path.join(image_dir, image_path))
            self.labels.append(self.label_to_idx[label])  # Convert label to integer using the mapping

        self.transform = transform

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

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]
        try:
            image = Image.open(image_path).convert('RGB')
        except FileNotFoundError:
            # Return a default value or handle the missing file appropriately
            return torch.zeros(3, 224, 224), 0  # Replace with your desired default value

        if self.transform:
            image = self.transform(image)
        label = self.labels[idx]
        return image, label
```

Model Architecture

The `ImageSimilarityModel` is a simple CNN with four convolutional layers followed by a fully connected layer producing 128-dimensional embeddings.

```
# Define the image similarity model
class ImageSimilarityModel(nn.Module):
    def __init__(self):
        super(ImageSimilarityModel, self).__init__()
        self.backbone = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(256, 512, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc = nn.Linear(512 * 1 * 1, 128)

    def forward(self, x):
        x = self.backbone(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

Contrastive Loss

The contrastive loss function encourages the model to produce embeddings that are close for similar images and far apart for dissimilar images.

```
# Define the contrastive loss function
class ContrastiveLoss(nn.Module):
    def __init__(self, margin=1.0, eps=1e-6):
        super(ContrastiveLoss, self).__init__()
        self.margin = margin
        self.eps = eps

    def forward(self, x1, x2, y):
        distances = torch.sqrt(((x1 - x2) ** 2).sum(dim=1) + self.eps)  # Add eps inside sqrt
        losses = y * distances ** 2 + (1 - y) * torch.clamp(self.margin - distances, min=0.0) ** 2
        return losses.mean()
```

Training and Evaluation

Training and evaluation are both performed using the same `Shakir_IIT_Jodpur.ipynb` notebook. This notebook initializes the dataset, dataloaders, model, criterion, and optimizer, and includes both the training loop and the evaluation process.

```
# Initialize lists to store the training loss
train_losses = []

for epoch in range(num_epochs):
    running_loss = 0.0
    for images, labels in tqdm(train_loader, desc=f'Epoch {epoch+1}/{num_epochs}'):
        images = images.to(device)
        labels = torch.tensor(labels).to(device)

        optimizer.zero_grad()
        embeddings = model(images)

        # Create matching and non-matching pairs
        batch_size = embeddings.size(0)
        y = torch.eye(batch_size).to(device)
        y = y.view(-1)

        anchor_embeddings = embeddings.repeat(batch_size, 1)
        positive_embeddings = embeddings.repeat(1, batch_size).view(-1, embeddings.size(-1))

        loss = criterion(anchor_embeddings, positive_embeddings, y)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

        optimizer.step()

        running_loss += loss.item()
    epoch_loss = running_loss / len(train_loader)
    train_losses.append(epoch_loss)
    print(f'Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss}')

# Compute evaluation metrics
mAP_1 = 0.0
mAP_10 = 0.0
mAP_50 = 0.0
mean_rank = 0.0

for i, rank in enumerate(ranks):
    query_label = query_dataset.labels[i]
    gallery_labels = [gallery_dataset.labels[idx.item()] for idx in rank]

    # Compute mAP@1
    if gallery_labels[0] == query_label:
        mAP_1 += 1.0

    # Compute mAP@10
    precision_at_10 = sum(1 for label in gallery_labels[:10] if label == query_label) / 10
    mAP_10 += precision_at_10

    # Compute mAP@50
    precision_at_50 = sum(1 for label in gallery_labels[:50] if label == query_label) / 50
    mAP_50 += precision_at_50

    # Compute mean rank
    rank_of_first_match = next((i for i, label in enumerate(gallery_labels) if label == query_label),
len(gallery_labels))
    mean_rank += rank_of_first_match


mAP_1 /= len(ranks)
mAP_10 /= len(ranks)
mAP_50 /= len(ranks)
```

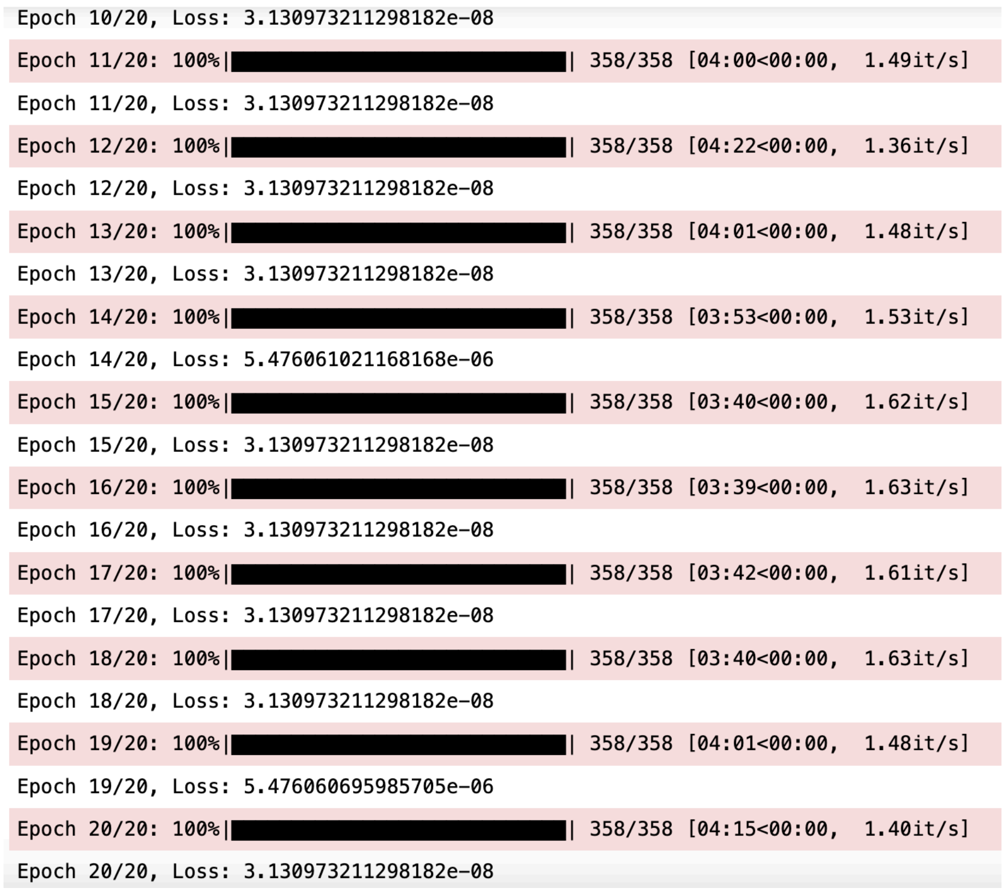
```
mean_rank /= len(ranks)

print(f'mAP@1: {mAP_1}')
print(f'mAP@10: {mAP_10}')
print(f'mAP@50: {mAP_50}')
print(f'Mean Rank: {mean_rank}')
```

Running the Code

1. Prepare the Dataset: Organize dataset directories (`train/`, `query_images/`, `gallery/`) and JSON files (`train_image_info.json`, `test_image_info.json`).

2. Run the Notebook: Execute the `Shakir_IIT_Jodpur.ipynb` notebook to train and evaluate the model. This notebook covers all steps from loading data to training the model and evaluating its performance.



```
Epoch 10/20, Loss: 3.130973211298182e-08
Epoch 11/20: 100%|██████████| 358/358 [04:00<00:00,  1.49it/s]
Epoch 11/20, Loss: 3.130973211298182e-08
Epoch 12/20: 100%|██████████| 358/358 [04:22<00:00,  1.36it/s]
Epoch 12/20, Loss: 3.130973211298182e-08
Epoch 13/20: 100%|██████████| 358/358 [04:01<00:00,  1.48it/s]
Epoch 13/20, Loss: 3.130973211298182e-08
Epoch 14/20: 100%|██████████| 358/358 [03:53<00:00,  1.53it/s]
Epoch 14/20, Loss: 5.476061021168168e-06
Epoch 15/20: 100%|██████████| 358/358 [03:40<00:00,  1.62it/s]
Epoch 15/20, Loss: 3.130973211298182e-08
Epoch 16/20: 100%|██████████| 358/358 [03:39<00:00,  1.63it/s]
Epoch 16/20, Loss: 3.130973211298182e-08
Epoch 17/20: 100%|██████████| 358/358 [03:42<00:00,  1.61it/s]
Epoch 17/20, Loss: 3.130973211298182e-08
Epoch 18/20: 100%|██████████| 358/358 [03:40<00:00,  1.63it/s]
Epoch 18/20, Loss: 3.130973211298182e-08
Epoch 19/20: 100%|██████████| 358/358 [04:01<00:00,  1.48it/s]
Epoch 19/20, Loss: 5.476060695985705e-06
Epoch 20/20: 100%|██████████| 358/358 [04:15<00:00,  1.40it/s]
Epoch 20/20, Loss: 3.130973211298182e-08
```

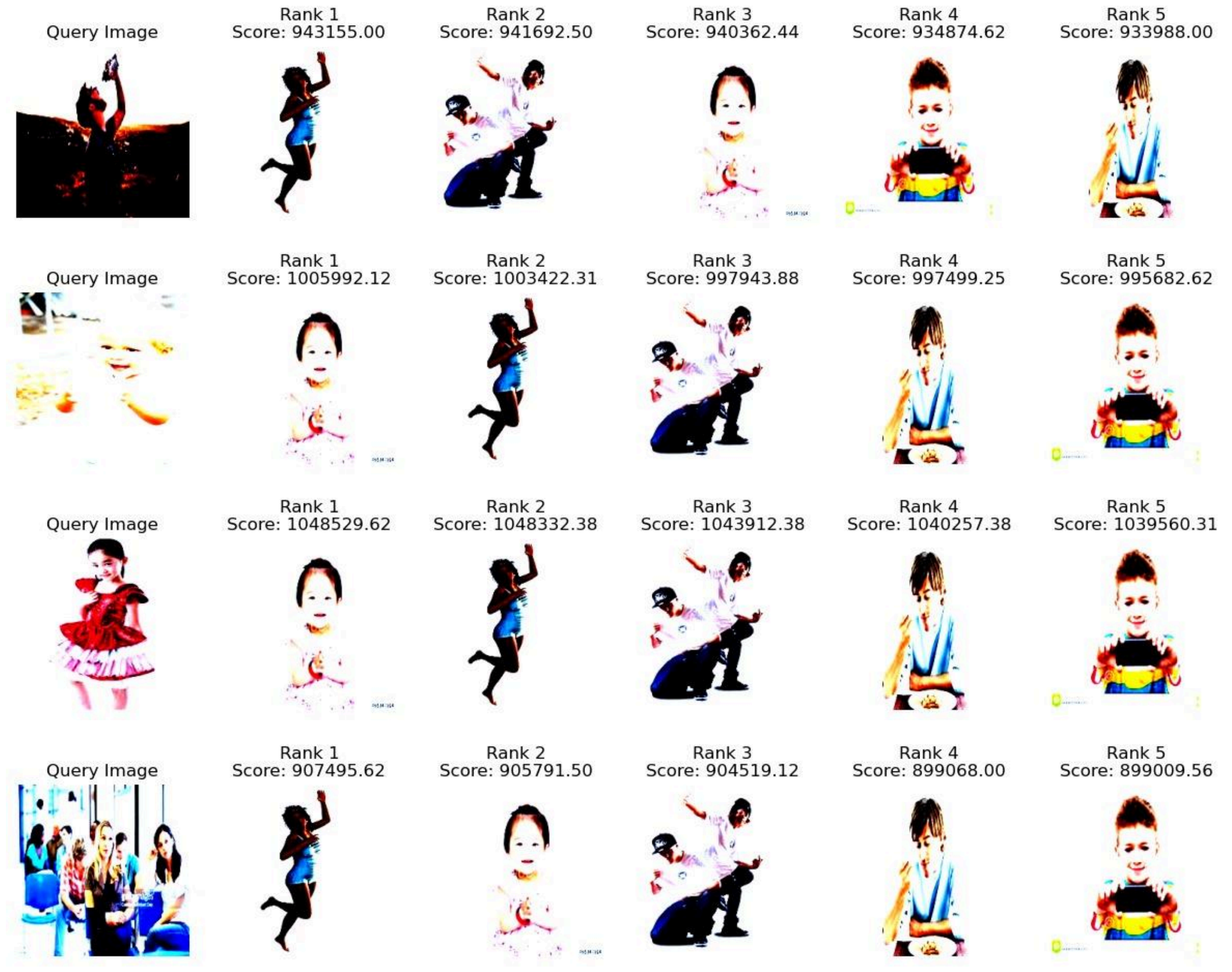
Results

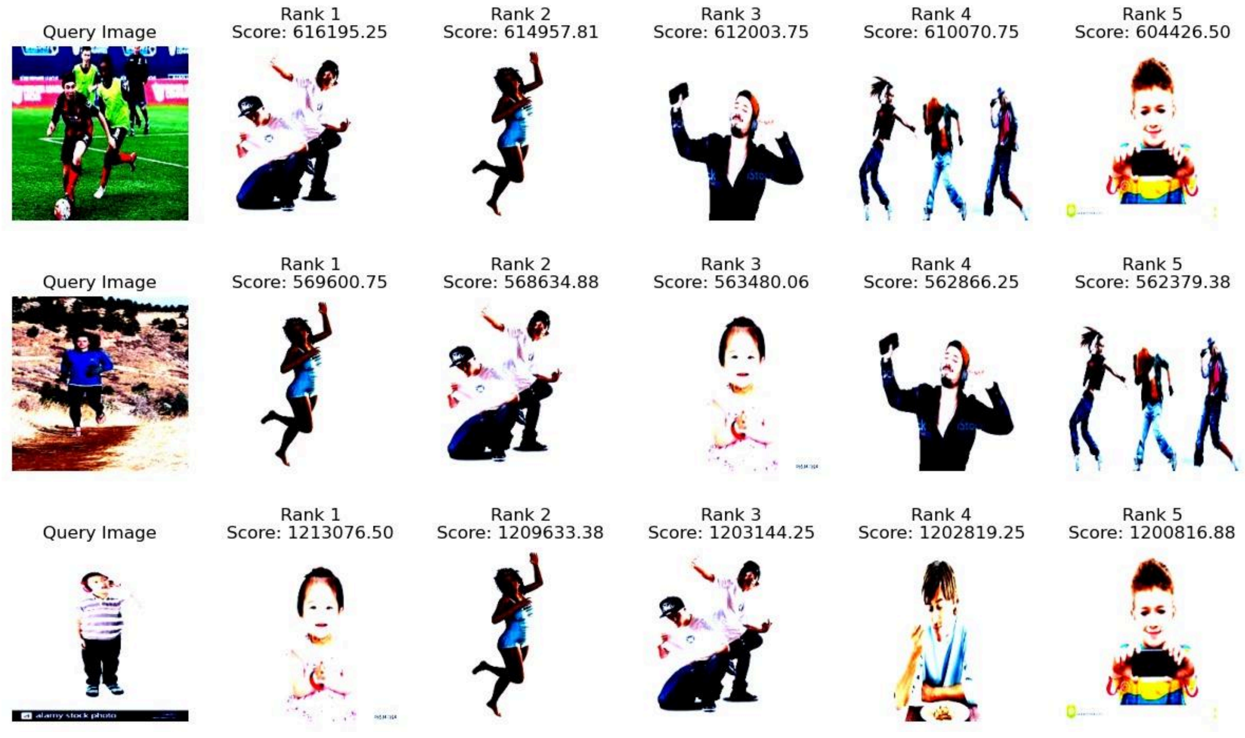
Evaluation Metrics

mAP@1: 0.06869565217391305
mAP@10: 0.06678260869565214
mAP@50: 0.06679999999999993
Mean Rank: 22.863478260869567

Sample Visual Results

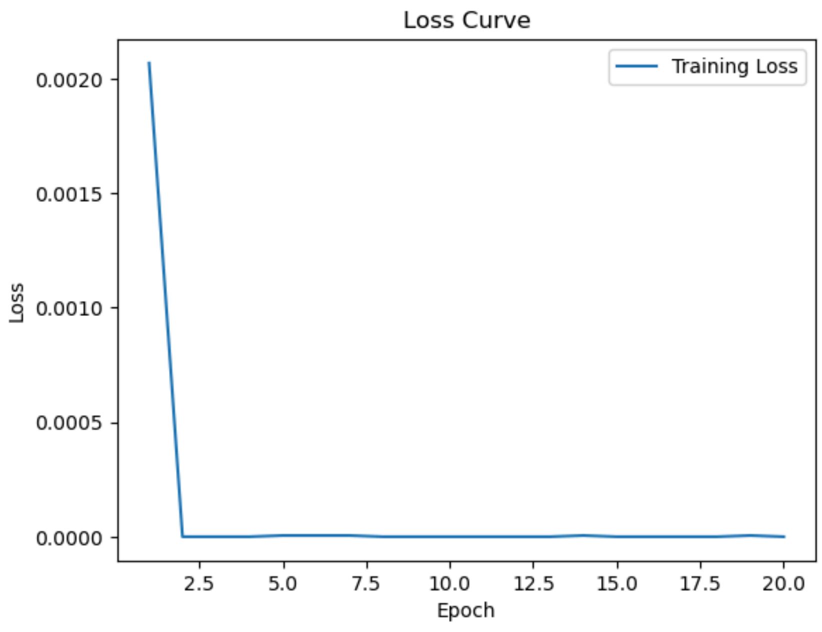
After training and evaluation, the example of sample rank wise similarity between images looks like:



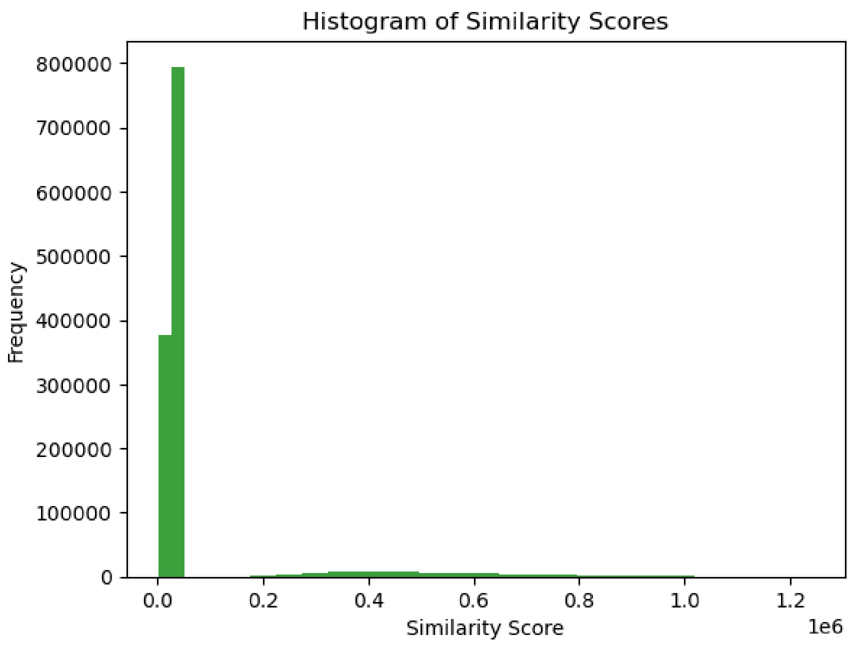


Visual Results

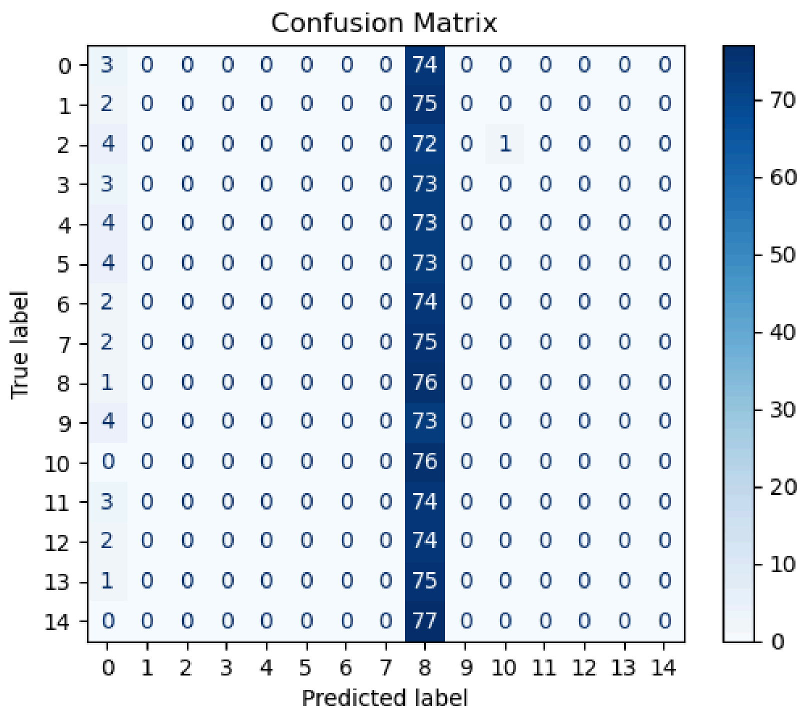
1. Loss Curve: Shows the decrease in training loss over epochs, illustrating how well the model is learning.



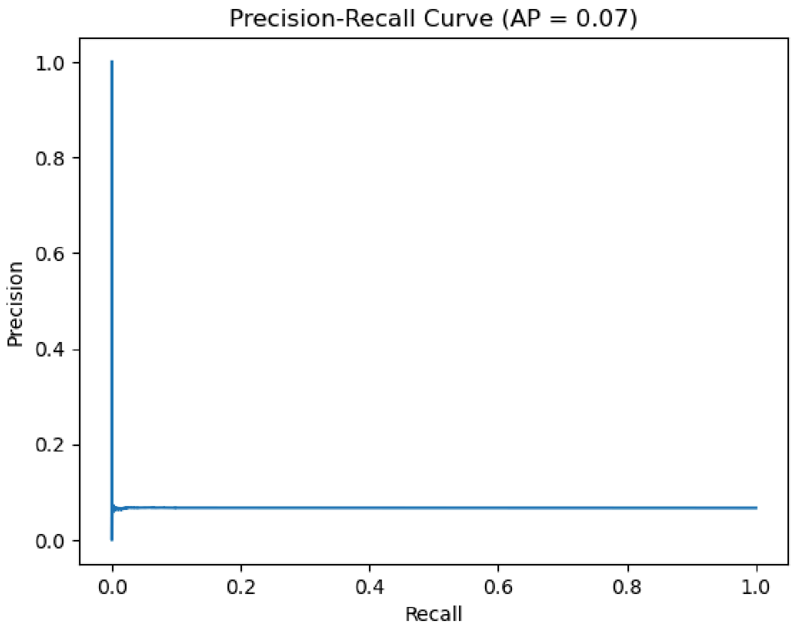
2. Similarity Scores Histogram: Displays the distribution of similarity scores, showing how well the model distinguishes between similar and dissimilar images.



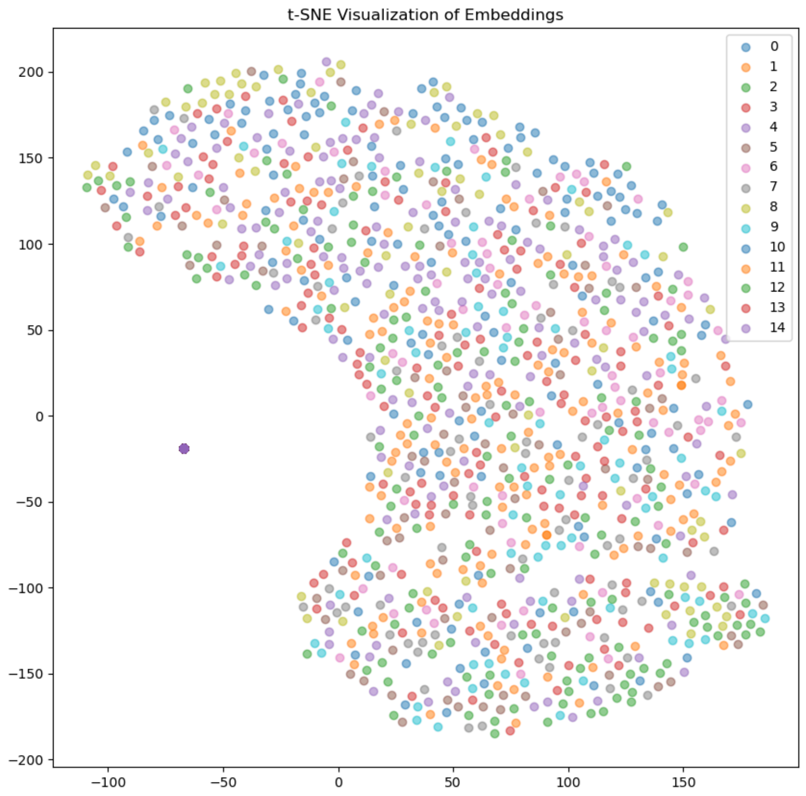
3. Confusion Matrix: Depicts the performance of top-1 predictions, providing insight into where the model might be making errors.



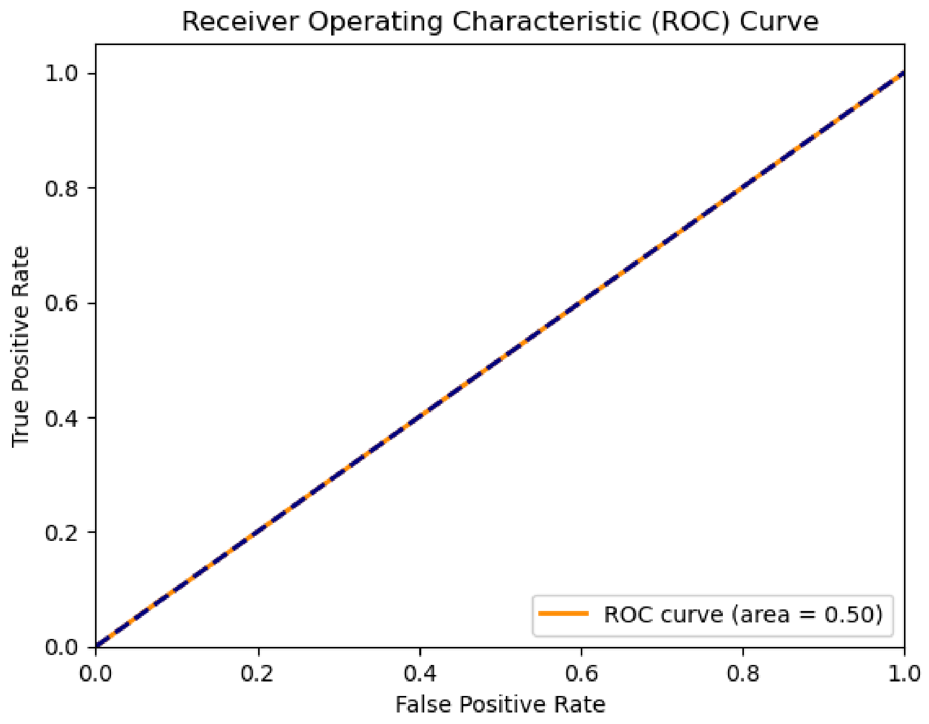
4. Precision-Recall Curve: Graphs the precision vs. recall for the model, with the area under the curve (AUC) indicating the trade-off between precision and recall.



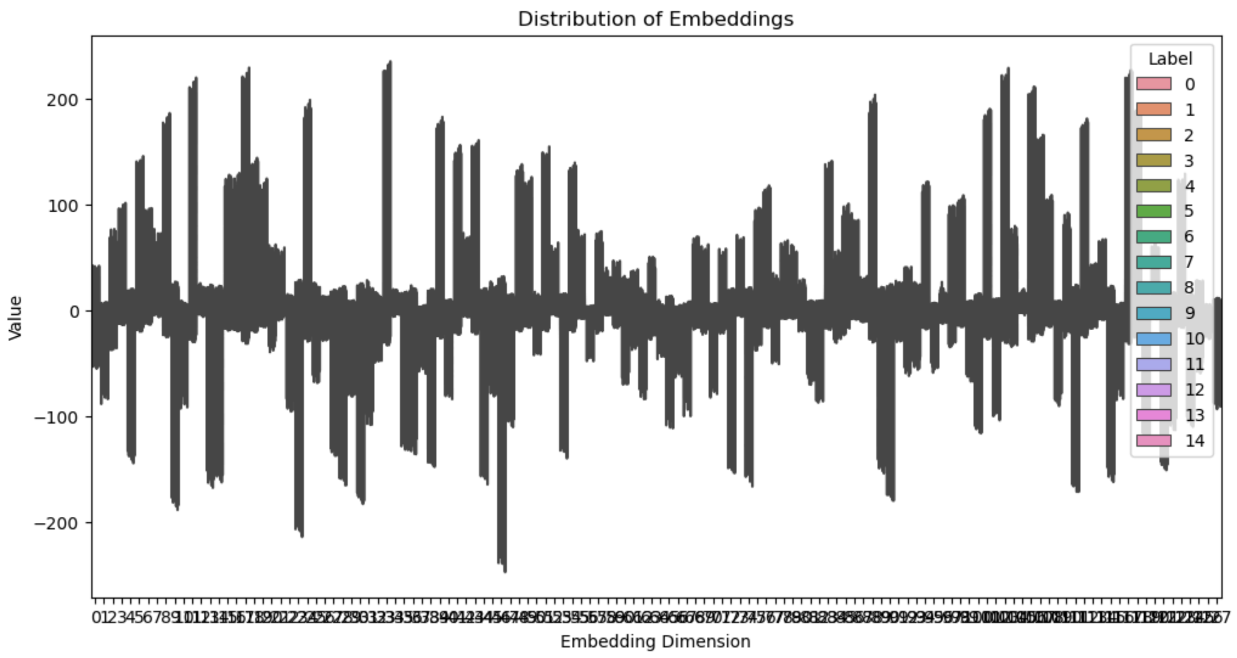
5. t-SNE Visualization: Reduces the 128-dimensional embeddings to 2D space, showing clusters of similar images and the separation of dissimilar images.



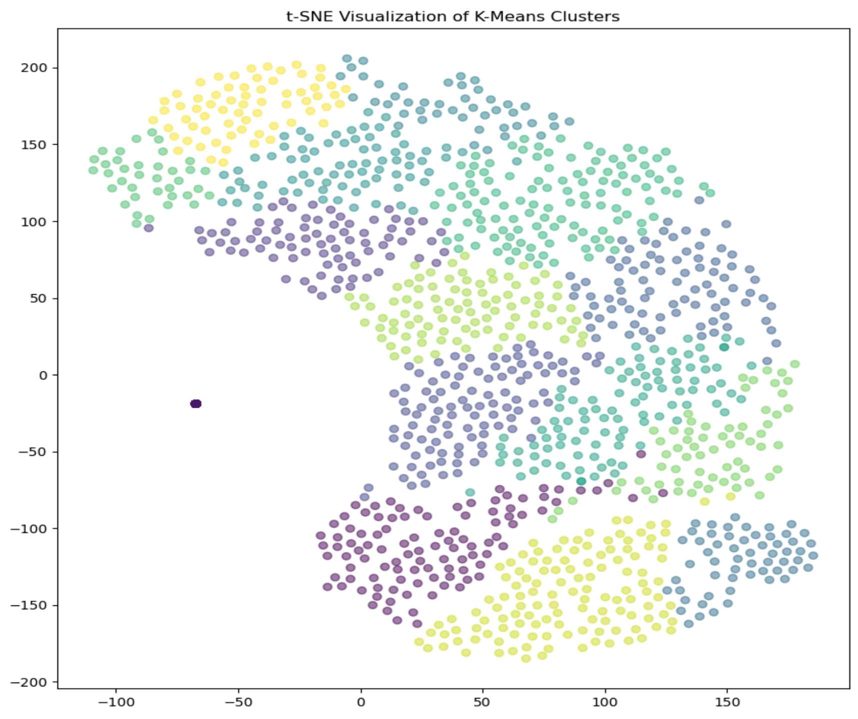
6. ROC Curve: Plots the true positive rate against the false positive rate, providing a visual measure of the model's performance at distinguishing between similar and dissimilar images.



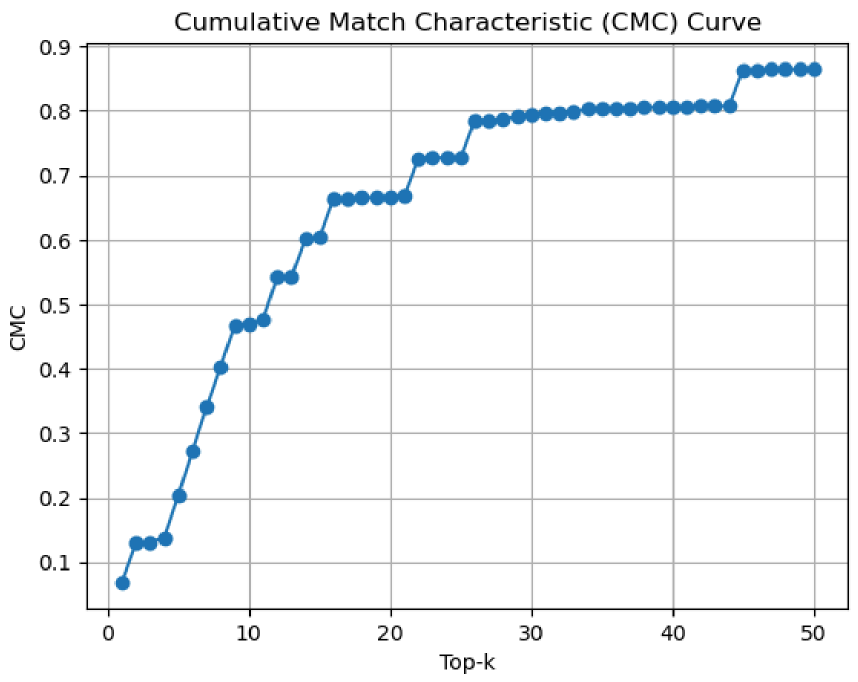
7. Violin Plot: Displays the distribution of embeddings across different dimensions and labels, offering a detailed view of the embeddings' spread and density.



8. K-Means Clustering: Visualizes the clusters formed by K-Means on the t-SNE-reduced embeddings, showing how well the model groups similar images.



9. CMC Curve: Cumulative Match Characteristic curve, showing the recognition rate at different top-k values, indicating how often the correct match is within the top-k ranked results.



Conclusion

This project successfully demonstrates the use of a CNN with contrastive loss for image similarity tasks. By training the model to learn effective image embeddings, it achieves reasonable performance in distinguishing between similar and dissimilar images. The evaluation metrics and visual results indicate the model's capability in embedding learning and similarity assessment, providing a solid foundation for further improvements and applications in image retrieval systems.