In [1]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, models
from torchvision.io import read_image
from sklearn.metrics import precision_recall_fscore_support
import pandas as pd
import numpy as np
import os

# Set random seed
torch.manual_seed(42)
np.random.seed(42)

# Load Dataset
base_dir = '/Users/Dell/Documents/TASK1/WikiArt'
pairs_csv = os.path.join(base_dir, 'nga_similarity_pairs.csv')
if not os.path.exists(pairs_csv) or os.stat(pairs_csv).st_size == 0:
    raise FileNotFoundError('Run prepare_similarity_dataset.py first to generate nga_similarity_pairs.csv')
pairs_df = pd.read_csv(pairs_csv)
print('First few pairs:')
print(pairs_df.head())

# Dataset Class
class NGAPairDataset(Dataset):
    def __init__(self, pairs_df, transform=None):
        self.pairs_df = pairs_df
        self.transform = transform
        self.valid_pairs = [(i, row) for i, row in pairs_df.iterrows() if os.path.exists(row['image1']) and os.path.exists(row['image2'])]
        if not self.valid_pairs:
            raise ValueError('No valid image pairs found')
        print(f'Valid pairs: {len(self.valid_pairs)}')

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

    def __getitem__(self, idx):
        _, row = self.valid_pairs[idx]
        img1 = read_image(row['image1']).float() / 255.0
        img2 = read_image(row['image2']).float() / 255.0
        label = row['label']
        if self.transform:
            img1 = self.transform(img1)
            img2 = self.transform(img2)
        return img1, img2, torch.tensor(label, dtype=torch.float32)

# Siamese Network
class SiameseNetwork(nn.Module):
    def __init__(self):
        super(SiameseNetwork, self).__init__()
        self.cnn = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
        self.cnn.fc = nn.Identity()
        self.fc = nn.Linear(512, 128)

    def forward_one(self, x):
        x = self.cnn(x)
        x = self.fc(x)
        return x

    def forward(self, x1, x2):
        out1 = self.forward_one(x1)
        out2 = self.forward_one(x2)
        return out1, out2

# Contrastive Loss
class ContrastiveLoss(nn.Module):
    def __init__(self, margin=1.0):
        super(ContrastiveLoss, self).__init__()
        self.margin = margin

    def forward(self, out1, out2, label):
        dist = torch.nn.functional.pairwise_distance(out1, out2)
        loss = torch.mean((label) * torch.pow(dist, 2) +
                          (1 - label) * torch.pow(torch.clamp(self.margin - dist, min=0.0), 2))
        return loss

# Transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Split Dataset
train_df = pairs_df.sample(frac=0.8, random_state=42)
test_df = pairs_df.drop(train_df.index)
train_dataset = NGAPairDataset(train_df, transform)
test_dataset = NGAPairDataset(test_df, transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Model Setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = SiameseNetwork().to(device)
criterion = ContrastiveLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for img1, img2, label in train_loader:
        img1, img2, label = img1.to(device), img2.to(device), label.to(device)
        optimizer.zero_grad()
        out1, out2 = model(img1, img2)
        loss = criterion(out1, out2, label)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}')

# Evaluation
model.eval()
y_true, y_pred, cos_sim = [], [], []
with torch.no_grad():
    for img1, img2, label in test_loader:
        img1, img2, label = img1.to(device), img2.to(device), label.to(device)
        out1, out2 = model(img1, img2)
        dist = torch.nn.functional.pairwise_distance(out1, out2)
        sim = torch.cosine_similarity(out1, out2)
        pred = (dist < 0.5).float()  # Threshold
        y_true.extend(label.cpu().numpy())
        y_pred.extend(pred.cpu().numpy())
        cos_sim.extend(sim.cpu().numpy())

# Metrics
accuracy = np.mean(np.array(y_true) == np.array(y_pred))
precision, recall, _, _ = precision_recall_fscore_support(y_true, y_pred, average='binary')
mse = np.mean((np.array(y_true) - np.array(cos_sim)) ** 2)
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}, Recall: {recall:.4f}')
print(f'Mean Cosine Similarity: {np.mean(cos_sim):.4f}')
print(f'MSE: {mse:.4f}')

# Save Model
torch.save(model.state_dict(), 'siamese_model.pth')
print('Model saved as siamese_model.pth')
```

```
First few pairs:
                                              image1  \
0  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...
1  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...
2  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...
3  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...
4  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...

                                              image2  label
0  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...      0
1  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...      1
2  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...      0
3  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...      1
4  /Users/Dell/Documents/TASK1/WikiArt/nga_portra...      0
Valid pairs: 36
Valid pairs: 9
Epoch [1/5], Loss: 15.4933
Epoch [2/5], Loss: 14.2111
Epoch [3/5], Loss: 6.5776
Epoch [4/5], Loss: 1.7557
Epoch [5/5], Loss: 1.5973
Accuracy: 0.2222
Precision: 0.0000, Recall: 0.0000
Mean Cosine Similarity: 0.4719
MSE: 0.3180
Model saved as siamese_model.pth
```

In [ ]:

```
/Users/Dell/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```