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

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

# Step 1: Load Preprocessed Dataset
base_dir = '/Users/Dell/Documents/TASK1/WikiArt'
labels_df = pd.read_csv(os.path.join(base_dir, 'labels.csv'))
print('First few rows of labels.csv:')
print(labels_df.head())

# Step 2: Define Dataset
class WikiArtDataset(Dataset):
    def __init__(self, labels_df, transform=None):
        self.valid_data = [(row['image_path'], row['label'])
                           for _, row in labels_df.iterrows()
                           if os.path.exists(row['image_path'])]
        print(f'Number of valid images: {len(self.valid_data)}')
        if not self.valid_data:
            raise ValueError('No valid images found. Run prepare_dataset.py first and check paths.')
        self.transform = transform

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

    def __getitem__(self, idx):
        img_path, label = self.valid_data[idx]
        image = read_image(img_path).float() / 255.0
        if self.transform:
            image = self.transform(image)
        return image, label

# Step 3: Define Model
class ConvRecurrentModel(nn.Module):
    def __init__(self, num_classes):
        super(ConvRecurrentModel, self).__init__()
        self.cnn = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
        self.cnn.fc = nn.Identity()
        self.rnn = nn.GRU(512, 256, num_layers=2, batch_first=True)
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        batch_size = x.size(0)
        x = self.cnn(x)
        x = x.unsqueeze(1)  # [batch, 1, 512]
        _, h_n = self.rnn(x)
        x = self.fc(h_n[-1])
        return x

# Step 4: Load and Split Data
label_to_idx = {label: idx for idx, label in enumerate(labels_df['label'].unique())}
labels_df['label'] = labels_df['label'].map(label_to_idx)
train_df, test_df = train_test_split(labels_df, test_size=0.2, random_state=42)

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

# Datasets and Loaders
train_dataset = WikiArtDataset(train_df, transform)
test_dataset = WikiArtDataset(test_df, transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Step 5: Model Setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = ConvRecurrentModel(num_classes=len(label_to_idx)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Step 6: Training
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}')

# Step 7: Evaluation
model.eval()
y_true, y_pred, y_probs = [], [], []
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        probs = torch.softmax(outputs, dim=1)
        _, predicted = torch.max(outputs, 1)
        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())
        y_probs.extend(probs.cpu().numpy())

# Metrics
print('Classification Report:')
print(classification_report(y_true, y_pred, target_names=[str(k) for k in label_to_idx.keys()]))
print('Confusion Matrix:')
print(confusion_matrix(y_true, y_pred))

# Outlier Detection
outliers = [(i, y_true[i], y_pred[i], max(y_probs[i]))
            for i in range(len(y_true)) if y_true[i] == y_pred[i] and max(y_probs[i]) < 0.5]
print(f'Potential Outliers (Correct but Low Confidence < 0.5): {len(outliers)}')
for idx, true, pred, prob in outliers[:5]:
    print(f'Index: {idx}, True: {true}, Predicted: {pred}, Confidence: {prob:.4f}')

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

```
First few rows of labels.csv:
                                          image_path  label
0  /Users/Dell/Documents/TASK1/WikiArt/wikiart/Re...     22
1  /Users/Dell/Documents/TASK1/WikiArt/wikiart/Ba...     20
2  /Users/Dell/Documents/TASK1/WikiArt/wikiart/Po...     16
3  /Users/Dell/Documents/TASK1/WikiArt/wikiart/Im...     17
4  /Users/Dell/Documents/TASK1/WikiArt/wikiart/Ro...      9
```

```
Number of valid images: 10676
Number of valid images: 2670
```

In [ ]: