In [ ]:

```
import torch
import os
from torchvision import datasets, transforms
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from PIL import Image
```

In [ ]:

```
# 图像
# 指定文件夹路径
folder_path = '/data/gxq/data/GSoC/wikiart'

# 获取文件夹中所有文件的路径
files = os.listdir(folder_path)

# 筛选出所有图片文件路径
image_files = [os.path.join(folder_path, f) for f in files if f.endswith(('.jpg'))]
for img_path in image_files:
    print(img_path)
transform = transforms.Compose([
   transforms.Resize((224, 224)),  # 调整图像大小
   transforms.ToTensor(),  # 转换为 Tensor 格式
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # 归一化
处理
])
```

In [ ]:

```
# 图像
# dataset = datasets.ImageFolder(root='/data/gxq/data/GSoC/wikiart',
transform=transform)
# img, label = dataset[0]  # 获取第一个样本的图像和标签
```

In [ ]:

```
# 表格
genre_train = pd.read_csv('/data/gxq/data/GSoC/label/genre_train.csv',header =  None)
genre_train
```

Out[ ]:

	0	1
0	Northern_Renaissance/hieronymus-bosch_st-jacqu...	7
1	Post_Impressionism/vincent-van-gogh_ears-of-wh...	4
2	Symbolism/theodor-severin-kittelsen_kvitebj-rn...	3
3	Expressionism/martiros-saryan_mother-of-the-ar...	6

0	**1**

	0	1
4	Early_Renaissance/leonardo-da-vinci_study-for-...	8
...	...	...
45498	Post_Impressionism/pablo-picasso_marin-and-stu...	8
45499	Expressionism/ernst-ludwig-kirchner_windswept-...	4
45500	Color_Field_Painting/gene-davis_two-part-blue-...	0
45501	Abstract_Expressionism/frank-bowling_sacha-jas...	0
45502	Ukiyo_e/utagawa-kuniyoshi_the-station-kambara.jpg	2

45503 rows × 2 columns

In [ ]:

```
# CSV
class CustomDataset(Dataset):
    def __init__(self, data_dir,label_dir, csv_file, transform=None):
        self.label_dir = label_dir
        self.data_dir = data_dir
        self.transform = transform
        self.csv_data = pd.read_csv(os.path.join(self.label_dir, csv_file),
header=None)

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

    def __getitem__(self, idx):
        file_path = os.path.join(self.data_dir, self.csv_data.iloc[idx, 0])
        c_lass = self.csv_data.iloc[idx, 1]
        image_path = file_path
        image = Image.open(image_path)
        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]),
        ])
        img_tensor = preprocess(image)
        # Assuming self.transform is a function that processes the data
        if self.transform:
            data = self.transform(img_tensor, c_lass)
        else:
            data = (img_tensor, c_lass)  # Return the file path and class

        return data
```

In [ ]:

```
data_dir = '/data/gxq/data/GSoC/wikiart'
label_dir = '/data/gxq/data/GSoC/label'
csv_file = 'genre_train.csv'
custom_dataset = CustomDataset(data_dir,label_dir, csv_file)

# Assuming you have a transform function defined
# transform = ...
```

In [ ]:

```
custom_dataset[6][0].shape
```

Out[ ]:

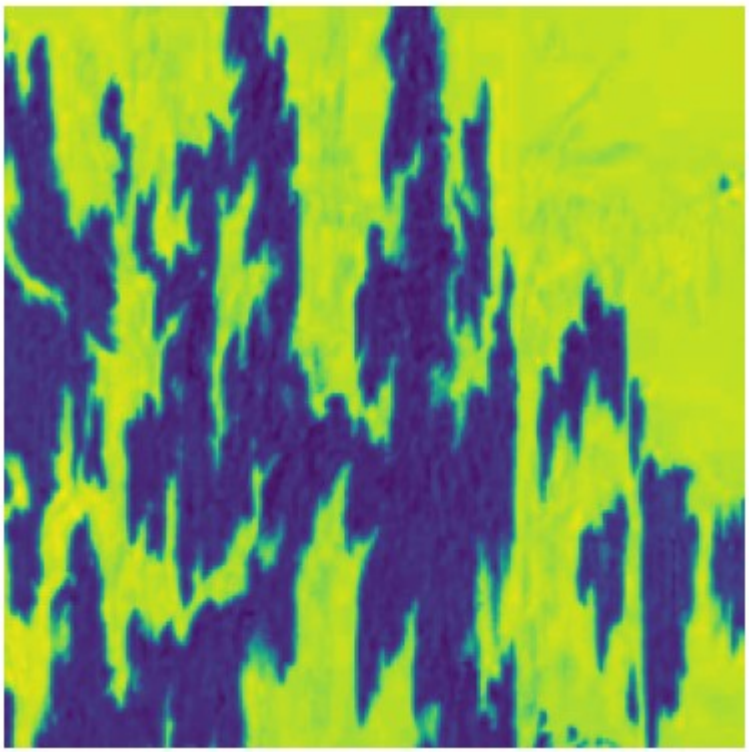
```
torch.Size([3, 224, 224])
```

In [ ]:

```
# 指定已知图片的位置
from matplotlib import pyplot as plt

img_np = custom_dataset[6][0].cpu().numpy()  # 如果在 GPU 上，需要先将张量移回 CPU

# 显示图像
plt.imshow(img_np[2])
plt.axis('off')  # 关闭坐标轴
plt.show()
```



In [ ]:

```
import torch.nn as nn
```

```
import torch.optim as optim
from tqdm.notebook import tqdm
from torch.utils.data import random_split
```

In [ ]:

```
class CNNRNN(nn.Module):
    def __init__(self, rnn_in_channels, rnn_hidden_size, num_layers, num_classes):
        super(CNNRNN, self).__init__()

        # 定义CNN部分
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.hidden = nn.Linear(3136, rnn_in_channels)
        # 定义多层RNN部分
        self.rnn = nn.ModuleList([
            nn.RNN(rnn_in_channels if i == 0 else rnn_hidden_size, rnn_hidden_size,
num_layers=1, batch_first=True)
            for i in range(num_layers)
        ])

        # 全连接层
        self.fc = nn.Linear(rnn_hidden_size, num_classes)

    def forward(self, x):
        # CNN部分
        x = self.cnn(x)

        # 将CNN的输出转换为RNN的输入格式
        x = x.view(x.size(0), x.size(1), -1)
        x = self.hidden(x)
        # 多层RNN部分
        for layer in self.rnn:
            x, _ = layer(x)

        # 全连接层
        out = self.fc(x[:, -1, :])  # 取RNN序列的最后一个输出作为预测

        return out
```

In [ ]:

```
val_dataset[10]
import numpy as np
```

In [ ]:

```
# train_data = ...
# train_labels = ...
dataset_size = len(custom_dataset)
```

```
train_size = int(0.8 * dataset_size)  # 假设80%的数据用于训练，20%用于验证
val_size = dataset_size - train_size

train_dataset, val_dataset = random_split(custom_dataset, [train_size, val_size])
data_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# # 创建数据集实例
# train_dataset = CustomDataset(train_data, train_labels, transform=transform)

device = 'cuda:1'



# 初始化模型
in_channels = 3
rnn_in_channels = 16
rnn_hidden_size = 23
num_layers = 2  # 设置RNN的层数
num_classes = 23

model = CNNRNN(rnn_in_channels, rnn_hidden_size, num_layers, num_classes)

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
# # 创建数据加载器
# train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
model = model.to(device)
# 训练模型
train_loader = data_loader
try:
    for epoch in range(5):  # 假设进行 5 个 epoch 的训练
        running_loss = 0.0
        for inputs, labels in tqdm(train_loader):
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
except Exception as e:
                print(f"Encountered an error in iteration {i}: {e}. Skipping this
iteration.")

print('Finished Training')
torch.save(model.state_dict(), 'cnnrnn_model.pth')
# 模拟多个 epoch 的训练过程
num_epochs = 5
```

```
for epoch in range(num_epochs):
    # 模拟每个 epoch 中多个 batch 的训练过程
    num_batches = 100
    for batch in range(num_batches):
        # 打印当前的 loss 值
        print(f'Epoch [{epoch+1}/{num_epochs}], Batch [{batch+1}/{num_batches}], Loss:
{loss:.4f}')
```

```
  0%|          | 0/1138 [00:00<?, ?it/s]

  0%|          | 0/1138 [00:00<?, ?it/s]

  0%|          | 0/1138 [00:00<?, ?it/s]

  0%|          | 0/1138 [00:00<?, ?it/s]
```