图像分类问题

首先考虑残差网络，这里选择 ResNet50

```
**import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset, random_split
from torchvision import transforms, models
from PIL import Image
import pandas as pd
from torch.amp import GradScaler, autocast
import random
import numpy as np

torch.cuda.empty_cache()
torch.cuda.ipc_collect()
```

由于前面处理的结果效果不好，鲁棒性不强，这里选择对于图像随机截取，并加上噪点来提升。

```
class FlowerDataset(Dataset):
    def init(self, image_dir, labels_csv=None, transform=None):
        self.image_dir = image_dir
        self.transform = transform
        self.labels = None
        if labels_csv:
            self.labels = pd.read_csv(labels_csv)
            self.image_ids = self.labels['file_name'].values
            self.targets = self.labels['label'].values
        else:
            self.image_ids = os.listdir(image_dir)
    def len(self):
        return len(self.image_ids)
    def getitem(self, idx):
        image_id = self.image_ids[idx]
        image_path = os.path.join(self.image_dir, image_id)
        image = Image.open(image_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        if self.labels is not None:
            label = self.targets[idx]
            return image, label
        return image, image_id
```

```
class AddGaussianNoise(object):
    def init(self, mean=0., std=0.05):
        self.mean = mean
        self.std = std
    def call(self, tensor):
        if random.random() < 0.2:
            return tensor + torch.randn(tensor.size()) * self.std + self.mean
        return tensor
    def repr(self):
        return self.class.name + f'(mean={self.mean}, std={self.std})'
transform = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.7, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.15),
    transforms.ToTensor(),
    AddGaussianNoise(0., 0.07),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

数据加载

```
train_dataset = FlowerDataset(
    image_dir="/root/data/input_data/train_images",
    labels_csv="/root/data/input_data/train_labels.csv",
    transform=transform
)
```

划分训练集和验证集

```
train_size = int(0.8 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=8, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=8, pin_memory=True)
test_dataset = FlowerDataset(
    image_dir="/root/data/input_data/test_images",
    transform=transform
)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=8, pin_memory=True)
```

定义模型

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)
num_classes = len(set(pd.read_csv("/root/data/input_data/train_labels.csv")['label']))
model.fc = nn.Sequential(
    nn.Dropout(0.32),
    nn.Linear(model.fc.in_features, num_classes)
)

model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=2e-3)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=15)
scaler = GradScaler()
```

训练 func

```
def train_model(model, train_loader, val_loader, criterion, optimizer, scheduler, epochs=20):
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        for images, labels in train_loader:
            images, labels = images.to(device, non_blocking=True), labels.to(device, non_blocking=True)
            optimizer.zero_grad()
            with autocast(device_type='cuda'):
                outputs = model(images)
                loss = criterion(outputs, labels)
            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()
            running_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)
        scheduler.step()
        epoch_loss = running_loss / len(train_loader)
        epoch_acc = correct / total
        print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}, Accuracy: {epoch_acc:.4f}")
        validate_model(model, val_loader, criterion)
```

验证

```
def validate_model(model, val_loader, criterion):
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device, non_blocking=True), labels.to(device, non_blocking=True)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)
    val_acc = correct / total
    print(f"Validation Loss: {val_loss/len(val_loader):.4f}, Validation Accuracy: {val_acc:.4f}")
```

测试

```
def test_model(model, test_loader):
    model.eval()
    predictions = []
    with torch.no_grad():
        for images, image_ids in test_loader:
            images = images.to(device, non_blocking=True)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            predictions.extend(zip(image_ids, preds.cpu().numpy()))
    return predictions
```

运行训练和测试

这里我们选择多个模型运行结果投票表决以提升准确率

```
ENSEMBLE_NUM = 3
all_predictions = []
for seed in range(ENSEMBLE_NUM):
    print(f"\n= Training model {seed+1}/{ENSEMBLE_NUM} =")
    torch.manual_seed(seed)
    random.seed(seed)
    # 重新初始化模型
    model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)
    model.fc = nn.Sequential(
        nn.Dropout(0.6),
        nn.Linear(model.fc.in_features, num_classes)
    )
    model = model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=2e-3)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=15)
    scaler = GradScaler()
    # 训练
    train_model(model, train_loader, val_loader, criterion, optimizer, scheduler, epochs=15)
    # 测试，收集概率
    model.eval()
    preds = []
    with torch.no_grad():
        for images, image_ids in test_loader:
            images = images.to(device, non_blocking=True)
            outputs = model(images)
            prob = torch.softmax(outputs, dim=1).cpu().numpy()
            preds.append(prob)
    all_predictions.append(np.concatenate(preds, axis=0))

ensemble_probs = np.mean(all_predictions, axis=0)
ensemble_labels = np.argmax(ensemble_probs, axis=1)

保存预测结果
image_ids = []
for batch in test_loader:
    _, batch_image_ids = batch
    image_ids.extend(batch_image_ids)
with open("/root/ans/results.csv", "w") as f:
    f.write("file_name,label\n")
    for image_id, label in zip(image_ids, ensemble_labels):
        f.write(f"{image_id},{label}\n")
```