IA Jojo

Introdução

Sumário

Instalação de Bibliotecas

Primeiramente, se ainda não foi feito, instalaremos todas as bibliotecas necessárias para esse projeto:

```
%pip install -r requirements.txt
```

Pré-processamento de Imagens

```
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, utils

# Variável para determinar o tamanho das imagens geradas no pré-processamento
image_size = 224  # Você pode ajustar este valor conforme necessário

# Função para mostrar imagens
def imshow(inp):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    plt.axis('off')  # Remove os eixos

# Definindo os caminhos
CHAR_FOLDER = Path("Characters")
TEST_FOLDER = CHAR_FOLDER / "Test"
TRAIN_FOLDER = CHAR_FOLDER / "Train"
```

```
# Transformações para pré-processamento das imagens
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(image_size),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(degrees=30),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(image_size + 32),
        transforms.CenterCrop(image_size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
    ]),
}

# Carregando os datasets
image_datasets = {
    'train': datasets.ImageFolder(TRAIN_FOLDER,
transform=data_transforms['train']),
    'test': datasets.ImageFolder(TEST_FOLDER,
transform=data_transforms['test'])
}

# Criando os data loaders
dataloaders = {
    'train': DataLoader(image_datasets['train'], batch_size=32,
shuffle=True, num_workers=4),
    'test': DataLoader(image_datasets['test'], batch_size=32,
shuffle=False, num_workers=4)
}

# Obtendo o número de classes
class_names = image_datasets['train'].classes
num_classes = len(class_names)

print(f"Classes: {class_names}")

Classes: ['Dio-Brando', 'Iggy', 'Jean-Pierre-Polnareff', 'Joseph-
Joestar', 'Joutarou-Kuujou', 'Muhammad-Avdol', 'Noriaki-Kakyouin']

# Obtendo um batch de imagens de treino
inputs, classes = next(iter(dataloaders['train']))

# Fazendo um grid de imagens
out = utils.make_grid(inputs)
```

```
# Aumentando o tamanho da figura
plt.figure(figsize=(12, 12))

# Mostrando as imagens
imshow(out)
plt.show()  # Adiciona plt.show() para garantir que as imagens sejam
exibidas em alguns ambientes
```

Escolha do Modelo

```
import torch
import torch.nn as nn
from torchvision import models
from torchvision.models import VGG16_Weights

# Carregando o modelo VGG16 pré-treinado
model = models.vgg16(weights=VGG16_Weights.IMAGENET1K_V1)

# Congelando os parâmetros do modelo
for param in model.parameters():
    param.requires_grad = False

# Número de características de entrada para a última camada
num_ftrs = model.classifier[6].in_features
model.classifier[6] = nn.Linear(num_ftrs, num_classes)

# Exibindo o modelo para verificar as mudanças
print(model)
```

Treinamento do Modelo

```
import torch
import torch.nn as nn
import torch.optim as optim
from pathlib import Path

# Definindo o dispositivo (GPU ou CPU)
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
model = model.to(device)

# Definindo a função de perda e o otimizador
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.classifier[6].parameters(), lr=0.001)

# Função para treinar o modelo
def train_model(model, dataloaders, criterion, optimizer,
num_epochs=25, model_path='model.pth'):
    best_model_wts = model.state_dict()
```

```
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # Cada época tem uma fase de treino e uma de validação
        for phase in ['train', 'test']:
            if phase == 'train':
                model.train()  # Definir o modelo para treinamento
            else:
                model.eval()   # Definir o modelo para validação

            running_loss = 0.0
            running_corrects = 0

            # Iterar sobre os dados
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                # Somente faça cálculo de gradiente na fase de
treinamento
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # Backpropagation e otimização na fase de
treinamento
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss /
len(dataloaders[phase].dataset)
            epoch_acc = running_corrects.double() /
len(dataloaders[phase].dataset)

            print(f'{phase} Loss: {epoch_loss:.4f} Acc:
{epoch_acc:.4f}')

            # Salvar o melhor modelo
            if phase == 'test' and epoch_acc > best_acc:
                best_acc = epoch_acc
```

```
                best_model_wts = model.state_dict()

    print(f'Best val Acc: {best_acc:.4f}')

    # Carregar os melhores pesos do modelo
    model.load_state_dict(best_model_wts)

    # Salvar o modelo treinado
    torch.save(model.state_dict(), model_path)

    return model

# Carregar o modelo salvo, se existir
model_path = 'model.pth'
if Path(model_path).exists():
    model.load_state_dict(torch.load(model_path))

# Treinar o modelo
model = train_model(model, dataloaders, criterion, optimizer,
num_epochs=25, model_path=model_path)
```

```
Epoch 0/24
----------
train Loss: 1.9979 Acc: 0.2277
test Loss: 1.7332 Acc: 0.2308
Epoch 1/24
----------
train Loss: 1.8952 Acc: 0.3036
test Loss: 1.6128 Acc: 0.2769
Epoch 2/24
----------
train Loss: 1.6437 Acc: 0.3527
test Loss: 1.6194 Acc: 0.3692
Epoch 3/24
----------
train Loss: 1.5659 Acc: 0.4062
test Loss: 1.5800 Acc: 0.3385
Epoch 4/24
----------
train Loss: 1.4889 Acc: 0.4554
test Loss: 1.5721 Acc: 0.3692
Epoch 5/24
----------
train Loss: 1.4049 Acc: 0.5000
test Loss: 1.4960 Acc: 0.5077
Epoch 6/24
----------
train Loss: 1.3576 Acc: 0.5312
test Loss: 1.6240 Acc: 0.3692
Epoch 7/24
```

```
----------
train Loss: 1.4255 Acc: 0.4554
test Loss: 1.5682 Acc: 0.4462
Epoch 8/24
----------
train Loss: 1.3853 Acc: 0.4732
test Loss: 1.5022 Acc: 0.4615
Epoch 9/24
----------
train Loss: 1.3990 Acc: 0.4955
test Loss: 1.4700 Acc: 0.4308
Epoch 10/24
----------
train Loss: 1.4334 Acc: 0.4241
test Loss: 1.4869 Acc: 0.4615
Epoch 11/24
----------
train Loss: 1.3199 Acc: 0.5268
test Loss: 1.3564 Acc: 0.5077
Epoch 12/24
----------
train Loss: 1.3218 Acc: 0.4866
test Loss: 1.3167 Acc: 0.5538
Epoch 13/24
----------
train Loss: 1.3737 Acc: 0.4732
test Loss: 1.3709 Acc: 0.4923
Epoch 14/24
----------
train Loss: 1.2242 Acc: 0.5446
test Loss: 1.4238 Acc: 0.4923
Epoch 15/24
----------
train Loss: 1.2743 Acc: 0.5268
test Loss: 1.3970 Acc: 0.5385
Epoch 16/24
----------
train Loss: 1.3602 Acc: 0.5446
test Loss: 1.3785 Acc: 0.5385
Epoch 17/24
----------
train Loss: 1.3104 Acc: 0.5089
test Loss: 1.3191 Acc: 0.5538
Epoch 18/24
----------
train Loss: 1.2815 Acc: 0.5089
test Loss: 1.3104 Acc: 0.6000
Epoch 19/24
----------
```

```
train Loss: 1.2090 Acc: 0.5580
test Loss: 1.3858 Acc: 0.5077
Epoch 20/24
----------
train Loss: 1.3702 Acc: 0.4643
test Loss: 1.4596 Acc: 0.4923
Epoch 21/24
----------
train Loss: 1.2327 Acc: 0.5268
test Loss: 1.4049 Acc: 0.5385
Epoch 22/24
----------
train Loss: 1.2588 Acc: 0.5625
test Loss: 1.4779 Acc: 0.5231
Epoch 23/24
----------
train Loss: 1.1828 Acc: 0.5625
test Loss: 1.4283 Acc: 0.5846
Epoch 24/24
----------
train Loss: 1.1635 Acc: 0.5312
test Loss: 1.3947 Acc: 0.5692
Best val Acc: 0.6000
```

Avaliação do Modelo

```
def evaluate_model(model, dataloaders):
    model.eval()
    running_corrects = 0

    with torch.no_grad():
        for inputs, labels in dataloaders['test']:
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            running_corrects += torch.sum(preds == labels.data)

    accuracy = running_corrects.double() /
len(dataloaders['test'].dataset)
    print(f'Test Accuracy: {accuracy:.4f}')

evaluate_model(model, dataloaders)
```

```
Test Accuracy: 0.5692
```