Name: Jai Prathyush

Roll No: 20MID0211

Code:

import os

import shutil

import time

from torchvision import datasets

import matplotlib.pyplot as plt

import torch

import torch.nn as nn

import torchvision

from torchvision.transforms import transforms

import torch.optim as optim

src = "/kaggle/input/bird-species-classification/train\_data/"

for path, subdirs, files **in** os.walk(src):

print(f"There are **{**len(subdirs)**}** directories and **{**len(files)**}** images in '**{**path**}**'.")

*# Setup train and testing paths*

train\_dir = "/kaggle/input/bird-species-classification/train\_data/train\_data"

test\_dir = "/kaggle/input/bird-species-classification/test\_data/test\_data/"

print(train\_dir, test\_dir)

batch\_size=4

transform = transforms.Compose(

[transforms.Resize((224, 224)),

transforms.CenterCrop(224),

transforms.ToTensor()])

train\_dataset = datasets.ImageFolder(train\_dir, transform=transform)

train\_dataloader = torch.utils.data.DataLoader(train\_dataset,

batch\_size=batch\_size,

shuffle=True,

drop\_last=True)

test\_dataset = datasets.ImageFolder(test\_dir, transform=transform)

test\_dataloader = torch.utils.data.DataLoader(test\_dataset,

batch\_size=batch\_size,

shuffle=True,

drop\_last=True)

*#data visulization*

classes = sorted(os.listdir(train\_dir))

print("classes", classes)

images, labels = next(iter(train\_dataloader))

print("Image Size: ", images.shape)

print("labels", labels)

images = images.permute(0, 2, 3, 1)

print("Image Size after resizing: ", images.shape)

for i **in** range(batch\_size):

plt.imshow(images[i])

plt.title(labels[i])

plt.show()

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = torchvision.models.vgg16(pretrained=True).to(device)

*#print(model)*

*# change the number of classes*

model.classifier[6].out\_features = 16

*# freeze convolution weights*

for param **in** model.features.parameters():

param.requires\_grad = False

print(model)

*# optimizer*

optimizer = optim.SGD(model.classifier.parameters(), lr=0.001, momentum=0.9)

*# loss function*

criterion = nn.CrossEntropyLoss()

*# training function*

def fit(model, train\_dataloader):

model.train()

train\_running\_loss = 0.0

train\_running\_correct = 0

for i, data **in** enumerate(train\_dataloader):

data, target = data[0].to(device), data[1].to(device)

optimizer.zero\_grad()

output = model(data)

loss = criterion(output, target)

train\_running\_loss += loss.item()

\_, preds = torch.max(output.data, 1)

train\_running\_correct += (preds == target).sum().item()

loss.backward()

optimizer.step()

train\_loss = train\_running\_loss / len(train\_dataloader.dataset)

train\_accuracy = 100. \* (train\_running\_correct / len(train\_dataloader.dataset))

print(f"Train Loss: **{**train\_loss**:**.4f**}**, Train Accuracy: **{**train\_accuracy**:**.2f**}**")

return train\_loss, train\_accuracy

*# validation function*

def validata(model, test\_dataloader):

model.eval()

val\_running\_loss = 0.0

val\_running\_correct = 0

for i, data **in** enumerate(test\_dataloader):

data, target = data[0].to(device), data[1].to(device)

output = model(data)

loss = criterion(output, target)

val\_running\_loss += loss.item()

\_, preds = torch.max(output.data, 1)

val\_running\_correct += (preds == target).sum().item()

val\_loss = val\_running\_loss / len(test\_dataloader.dataset)

val\_accuracy = 100. \* (val\_running\_correct / len(test\_dataloader.dataset))

return val\_loss, val\_accuracy

*#Train and validate*

epochs = 10

train\_loss, train\_accuracy = [], []

val\_loss, val\_accuracy = [], []

start = time.time()

for epoch **in** range(epochs):

train\_epoch\_loss, train\_epoch\_accuracy = fit(model, train\_dataloader)

val\_epoch\_loss, val\_epoch\_accuracy = validata(model, test\_dataloader)

train\_loss.append(train\_epoch\_loss)

train\_accuracy.append(train\_epoch\_accuracy)

val\_loss.append(val\_epoch\_loss)

val\_accuracy.append(val\_epoch\_accuracy)

end = time.time()

print((end-start)/60, 'minutes')

plt.figure(figsize=(10, 7))

plt.plot(train\_accuracy, color='green', label='train accuracy')

plt.plot(val\_accuracy, color='blue', label='validataion accuracy')

plt.legend()

plt.savefig('accuracy.png')

plt.show()

plt.figure(figsize=(10, 7))

plt.plot(train\_loss, color='orange', label='train loss')

plt.plot(val\_loss, color='red', label='validataion loss')

plt.legend()

plt.savefig('loss.png')

plt.show()