Using device: cuda

Loading JPEG images...

100%|██████████| 50/50 [00:00<00:00, 6197.26it/s]

Found 4479 JPEG images in 50 classes

/usr/local/lib/python3.10/dist-packages/torchvision/models/\_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=ViT\_B\_16\_Weights.IMAGENET1K\_V1`. You can also use `weights=ViT\_B\_16\_Weights.DEFAULT` to get the most up-to-date weights.

warnings.warn(msg)

Downloading: "<https://download.pytorch.org/models/vit_b_16-c867db91.pth>" to /root/.cache/torch/hub/checkpoints/vit\_b\_16-c867db91.pth

100%|██████████| 330M/330M [00:02<00:00, 170MB/s]

Epoch 1/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=1.1031, acc=76.05%]

Validation: 100%|██████████| 28/28 [00:15<00:00, 1.75it/s]

==================================================

Epoch 1/20 Results:

==================================================

Training:

Loss: 1.1031

Accuracy: 76.05%

Precision: 0.7916

Recall: 0.7520

F1 Score: 0.7639

Validation:

Loss: 0.3120

Accuracy: 91.52%

Precision: 0.9115

Recall: 0.9178

F1 Score: 0.9075

Time taken: 168.81 seconds

Epoch 2/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.1775, acc=96.09%]

Validation: 100%|██████████| 28/28 [00:15<00:00, 1.75it/s]

==================================================

Epoch 2/20 Results:

==================================================

Training:

Loss: 0.1775

Accuracy: 96.09%

Precision: 0.9592

Recall: 0.9588

F1 Score: 0.9588

Validation:

Loss: 0.2411

Accuracy: 94.42%

Precision: 0.9496

Recall: 0.9459

F1 Score: 0.9448

Time taken: 167.84 seconds

Epoch 3/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.0716, acc=98.41%]

Validation: 100%|██████████| 28/28 [00:15<00:00, 1.75it/s]

==================================================

Epoch 3/20 Results:

==================================================

Training:

Loss: 0.0716

Accuracy: 98.41%

Precision: 0.9836

Recall: 0.9835

F1 Score: 0.9835

Validation:

Loss: 0.3958

Accuracy: 89.29%

Precision: 0.9214

Recall: 0.8933

F1 Score: 0.8920

Time taken: 167.92 seconds

Epoch 4/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0735, acc=98.38%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.75it/s]

==================================================

Epoch 4/20 Results:

==================================================

Training:

Loss: 0.0735

Accuracy: 98.38%

Precision: 0.9834

Recall: 0.9828

F1 Score: 0.9830

Validation:

Loss: 0.2357

Accuracy: 93.86%

Precision: 0.9437

Recall: 0.9402

F1 Score: 0.9393

Time taken: 168.23 seconds

Epoch 5/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0254, acc=99.67%]

Validation: 100%|██████████| 28/28 [00:15<00:00, 1.76it/s]

==================================================

Epoch 5/20 Results:

==================================================

Training:

Loss: 0.0254

Accuracy: 99.67%

Precision: 0.9969

Recall: 0.9961

F1 Score: 0.9965

Validation:

Loss: 0.1939

Accuracy: 95.31%

Precision: 0.9574

Recall: 0.9541

F1 Score: 0.9549

Time taken: 168.24 seconds

Epoch 6/20: 100%|██████████| 112/112 [02:31<00:00, 1.35s/it, loss=0.0251, acc=99.47%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.74it/s]

==================================================

Epoch 6/20 Results:

==================================================

Training:

Loss: 0.0251

Accuracy: 99.47%

Precision: 0.9943

Recall: 0.9942

F1 Score: 0.9942

Validation:

Loss: 0.1897

Accuracy: 95.87%

Precision: 0.9639

Recall: 0.9600

F1 Score: 0.9607

Time taken: 167.71 seconds

Epoch 7/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0126, acc=99.75%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.74it/s]

==================================================

Epoch 7/20 Results:

==================================================

Training:

Loss: 0.0126

Accuracy: 99.75%

Precision: 0.9973

Recall: 0.9974

F1 Score: 0.9973

Validation:

Loss: 0.2627

Accuracy: 93.08%

Precision: 0.9423

Recall: 0.9332

F1 Score: 0.9328

Time taken: 168.30 seconds

Epoch 8/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0258, acc=99.53%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.67it/s]

==================================================

Epoch 8/20 Results:

==================================================

Training:

Loss: 0.0258

Accuracy: 99.53%

Precision: 0.9958

Recall: 0.9958

F1 Score: 0.9957

Validation:

Loss: 0.2292

Accuracy: 94.42%

Precision: 0.9485

Recall: 0.9477

F1 Score: 0.9465

Time taken: 168.88 seconds

Epoch 9/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.0713, acc=98.46%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.73it/s]

==================================================

Epoch 9/20 Results:

==================================================

Training:

Loss: 0.0713

Accuracy: 98.46%

Precision: 0.9847

Recall: 0.9843

F1 Score: 0.9845

Validation:

Loss: 0.3191

Accuracy: 92.41%

Precision: 0.9344

Recall: 0.9232

F1 Score: 0.9228

Time taken: 168.05 seconds

Epoch 10/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0577, acc=98.41%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.74it/s]

==================================================

Epoch 10/20 Results:

==================================================

Training:

Loss: 0.0577

Accuracy: 98.41%

Precision: 0.9844

Recall: 0.9840

F1 Score: 0.9842

Validation:

Loss: 0.3298

Accuracy: 92.30%

Precision: 0.9356

Recall: 0.9268

F1 Score: 0.9273

Time taken: 168.44 seconds

Epoch 11/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.0140, acc=99.69%]

Validation: 100%|██████████| 28/28 [00:15<00:00, 1.76it/s]

==================================================

Epoch 11/20 Results:

==================================================

Training:

Loss: 0.0140

Accuracy: 99.69%

Precision: 0.9970

Recall: 0.9970

F1 Score: 0.9970

Validation:

Loss: 0.2701

Accuracy: 94.31%

Precision: 0.9489

Recall: 0.9428

F1 Score: 0.9441

Time taken: 167.79 seconds

Epoch 12/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0083, acc=99.92%]

Validation: 100%|██████████| 28/28 [00:15<00:00, 1.75it/s]

==================================================

Epoch 12/20 Results:

==================================================

Training:

Loss: 0.0083

Accuracy: 99.92%

Precision: 0.9992

Recall: 0.9992

F1 Score: 0.9992

Validation:

Loss: 0.2991

Accuracy: 93.64%

Precision: 0.9416

Recall: 0.9401

F1 Score: 0.9380

Time taken: 168.37 seconds

Epoch 13/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0821, acc=98.02%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.74it/s]

==================================================

Epoch 13/20 Results:

==================================================

Training:

Loss: 0.0821

Accuracy: 98.02%

Precision: 0.9794

Recall: 0.9799

F1 Score: 0.9797

Validation:

Loss: 0.3788

Accuracy: 91.29%

Precision: 0.9228

Recall: 0.9120

F1 Score: 0.9089

Time taken: 168.32 seconds

Epoch 14/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.0441, acc=98.72%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.74it/s]

==================================================

Epoch 14/20 Results:

==================================================

Training:

Loss: 0.0441

Accuracy: 98.72%

Precision: 0.9871

Recall: 0.9874

F1 Score: 0.9872

Validation:

Loss: 0.3510

Accuracy: 90.85%

Precision: 0.9237

Recall: 0.9153

F1 Score: 0.9088

Time taken: 167.97 seconds

Epoch 15/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.0459, acc=99.08%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.75it/s]

==================================================

Epoch 15/20 Results:

==================================================

Training:

Loss: 0.0459

Accuracy: 99.08%

Precision: 0.9898

Recall: 0.9900

F1 Score: 0.9899

Validation:

Loss: 0.1990

Accuracy: 95.09%

Precision: 0.9541

Recall: 0.9519

F1 Score: 0.9514

Time taken: 168.05 seconds

Epoch 16/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.0290, acc=99.27%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.74it/s]

==================================================

Epoch 16/20 Results:

==================================================

Training:

Loss: 0.0290

Accuracy: 99.27%

Precision: 0.9924

Recall: 0.9922

F1 Score: 0.9923

Validation:

Loss: 0.2508

Accuracy: 93.42%

Precision: 0.9356

Recall: 0.9381

F1 Score: 0.9336

Time taken: 167.92 seconds

Epoch 17/20: 100%|██████████| 112/112 [02:31<00:00, 1.35s/it, loss=0.0480, acc=98.74%]

Validation: 100%|██████████| 28/28 [00:15<00:00, 1.75it/s]

==================================================

Epoch 17/20 Results:

==================================================

Training:

Loss: 0.0480

Accuracy: 98.74%

Precision: 0.9879

Recall: 0.9878

F1 Score: 0.9878

Validation:

Loss: 0.3835

Accuracy: 90.51%

Precision: 0.9157

Recall: 0.9133

F1 Score: 0.9105

Time taken: 167.69 seconds

Epoch 18/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0457, acc=98.77%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.74it/s]

==================================================

Epoch 18/20 Results:

==================================================

Training:

Loss: 0.0457

Accuracy: 98.77%

Precision: 0.9878

Recall: 0.9873

F1 Score: 0.9875

Validation:

Loss: 0.3088

Accuracy: 92.97%

Precision: 0.9356

Recall: 0.9332

F1 Score: 0.9321

Time taken: 168.29 seconds

Epoch 19/20: 100%|██████████| 112/112 [02:32<00:00, 1.36s/it, loss=0.0184, acc=99.47%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.75it/s]

==================================================

Epoch 19/20 Results:

==================================================

Training:

Loss: 0.0184

Accuracy: 99.47%

Precision: 0.9946

Recall: 0.9947

F1 Score: 0.9946

Validation:

Loss: 0.4100

Accuracy: 90.51%

Precision: 0.9303

Recall: 0.9084

F1 Score: 0.9084

Time taken: 168.04 seconds

Epoch 20/20: 100%|██████████| 112/112 [02:31<00:00, 1.36s/it, loss=0.0153, acc=99.75%]

Validation: 100%|██████████| 28/28 [00:16<00:00, 1.75it/s]

==================================================

Epoch 20/20 Results:

==================================================

Training:

Loss: 0.0153

Accuracy: 99.75%

Precision: 0.9975

Recall: 0.9974

F1 Score: 0.9974

Validation:

Loss: 0.3462

Accuracy: 92.75%

Precision: 0.9353

Recall: 0.9298

F1 Score: 0.9256

Time taken: 168.00 seconds

A graph with orange and blue lines

Description automatically generated

A graph with lines and numbers

Description automatically generated

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 PIL import Image

from tqdm import tqdm

import time

from datetime import datetime

from torchmetrics import Precision, Recall, F1Score

import matplotlib.pyplot as plt

class JPEGImageDataset(Dataset):

    def \_\_init\_\_(self, root\_dir, transform=None):

        self.root\_dir = root\_dir

        self.transform = transform

        self.classes = [d for d in os.listdir(root\_dir) if os.path.isdir(os.path.join(root\_dir, d))]

        self.class\_to\_idx = {cls\_name: i for i, cls\_name in enumerate(self.classes)}

        self.images = []

        self.labels = []

        # Only collect JPEG images

        valid\_extensions = ('.jpg', '.jpeg')

        print("Loading JPEG images...")

        for class\_name in tqdm(self.classes):

            class\_dir = os.path.join(root\_dir, class\_name)

            class\_idx = self.class\_to\_idx[class\_name]

            for img\_name in os.listdir(class\_dir):

                if img\_name.lower().endswith(valid\_extensions):

                    self.images.append(os.path.join(class\_dir, img\_name))

                    self.labels.append(class\_idx)

    def \_\_len\_\_(self):

        return len(self.images)

    def \_\_getitem\_\_(self, idx):

        img\_path = self.images[idx]

        image = Image.open(img\_path).convert('RGB')

        label = self.labels[idx]

        if self.transform:

            image = self.transform(image)

        return image, label

def train\_vit\_base():

    # Create timestamp for logging

    timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')

    log\_file = f'training\_results\_vit\_base\_{timestamp}.txt'

    # Set device

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

    print(f"Using device: {device}")

    # Define transforms

    transform = transforms.Compose([

        transforms.Resize((224, 224)),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

    ])

    # Load dataset

    data\_path = "/content/butterfly-dataset/Train"

    try:

        # Create custom dataset

        dataset = JPEGImageDataset(root\_dir=data\_path, transform=transform)

        print(f"Found {len(dataset)} JPEG images in {len(dataset.classes)} classes")

        # Initialize logging

        with open(log\_file, 'w') as f:

            f.write(f"Training started at: {timestamp}\n")

            f.write(f"Model: ViT-Base\n")

            f.write(f"Total images: {len(dataset)}\n")

            f.write(f"Number of classes: {len(dataset.classes)}\n")

            f.write("="\*50 + "\n\n")

        # Split dataset

        train\_size = int(0.8 \* len(dataset))

        valid\_size = len(dataset) - train\_size

        train\_dataset, valid\_dataset = torch.utils.data.random\_split(

            dataset, [train\_size, valid\_size]

        )

        # Create data loaders

        train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

        valid\_loader = DataLoader(valid\_dataset, batch\_size=32, shuffle=False)

        # Initialize ViT-Base model

        model = models.vit\_b\_16(pretrained=True)

        num\_ftrs = model.heads[0].in\_features

        model.heads = nn.Linear(num\_ftrs, len(dataset.classes))

        model = model.to(device)

        # Initialize metrics

        precision = Precision(task="multiclass", num\_classes=len(dataset.classes), average='macro')

        recall = Recall(task="multiclass", num\_classes=len(dataset.classes), average='macro')

        f1 = F1Score(task="multiclass", num\_classes=len(dataset.classes), average='macro')

        # Training setup

        criterion = nn.CrossEntropyLoss()

        optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight\_decay=0.01)

        # Training loop

        num\_epochs = 20

        best\_valid\_loss = float("inf")

        # Store metrics

        metrics = {

            'train\_loss': [], 'train\_acc': [],

            'valid\_loss': [], 'valid\_acc': [],

            'train\_precision': [], 'train\_recall': [], 'train\_f1': [],

            'valid\_precision': [], 'valid\_recall': [], 'valid\_f1': []

        }

        for epoch in range(num\_epochs):

            epoch\_start\_time = time.time()

            # Training phase

            model.train()

            train\_loss = 0

            correct\_train = 0

            total\_train = 0

            predicted\_train = []

            labels\_train = []

            # Progress bar for training

            with tqdm(train\_loader, desc=f"Epoch {epoch+1}/{num\_epochs}") as pbar:

                for inputs, labels in pbar:

                    inputs, labels = inputs.to(device), labels.to(device)

                    optimizer.zero\_grad()

                    outputs = model(inputs)

                    loss = criterion(outputs, labels)

                    loss.backward()

                    optimizer.step()

                    train\_loss += loss.item()

                    \_, predicted = torch.max(outputs, 1)

                    correct\_train += (predicted == labels).sum().item()

                    total\_train += labels.size(0)

                    predicted\_train.extend(predicted.cpu().numpy())

                    labels\_train.extend(labels.cpu().numpy())

                    # Update progress bar

                    pbar.set\_postfix({

                        'loss': f"{train\_loss/len(train\_loader):.4f}",

                        'acc': f"{100.\*correct\_train/total\_train:.2f}%"

                    })

            # Validation phase

            model.eval()

            valid\_loss = 0

            correct\_valid = 0

            total\_valid = 0

            predicted\_valid = []

            labels\_valid = []

            with torch.no\_grad():

                for inputs, labels in tqdm(valid\_loader, desc="Validation"):

                    inputs, labels = inputs.to(device), labels.to(device)

                    outputs = model(inputs)

                    loss = criterion(outputs, labels)

                    valid\_loss += loss.item()

                    \_, predicted = torch.max(outputs, 1)

                    correct\_valid += (predicted == labels).sum().item()

                    total\_valid += labels.size(0)

                    predicted\_valid.extend(predicted.cpu().numpy())

                    labels\_valid.extend(labels.cpu().numpy())

            # Calculate metrics

            train\_loss = train\_loss/len(train\_loader)

            valid\_loss = valid\_loss/len(valid\_loader)

            train\_acc = 100.\*correct\_train/total\_train

            valid\_acc = 100.\*correct\_valid/total\_valid

            # Calculate precision, recall, and F1 score

            train\_precision = precision(torch.tensor(predicted\_train), torch.tensor(labels\_train))

            train\_recall = recall(torch.tensor(predicted\_train), torch.tensor(labels\_train))

            train\_f1 = f1(torch.tensor(predicted\_train), torch.tensor(labels\_train))

            valid\_precision = precision(torch.tensor(predicted\_valid), torch.tensor(labels\_valid))

            valid\_recall = recall(torch.tensor(predicted\_valid), torch.tensor(labels\_valid))

            valid\_f1 = f1(torch.tensor(predicted\_valid), torch.tensor(labels\_valid))

            epoch\_time = time.time() - epoch\_start\_time

            # Store metrics

            metrics['train\_loss'].append(train\_loss)

            metrics['train\_acc'].append(train\_acc)

            metrics['valid\_loss'].append(valid\_loss)

            metrics['valid\_acc'].append(valid\_acc)

            metrics['train\_precision'].append(train\_precision.item())

            metrics['train\_recall'].append(train\_recall.item())

            metrics['train\_f1'].append(train\_f1.item())

            metrics['valid\_precision'].append(valid\_precision.item())

            metrics['valid\_recall'].append(valid\_recall.item())

            metrics['valid\_f1'].append(valid\_f1.item())

            # Print detailed results

            print(f"\n{'='\*50}")

            print(f"Epoch {epoch+1}/{num\_epochs} Results:")

            print(f"{'='\*50}")

            print(f"\nTraining:")

            print(f" Loss: {train\_loss:.4f}")

            print(f" Accuracy: {train\_acc:.2f}%")

            print(f" Precision: {train\_precision:.4f}")

            print(f" Recall: {train\_recall:.4f}")

            print(f" F1 Score: {train\_f1:.4f}")

            print(f"\nValidation:")

            print(f" Loss: {valid\_loss:.4f}")

            print(f" Accuracy: {valid\_acc:.2f}%")

            print(f" Precision: {valid\_precision:.4f}")

            print(f" Recall: {valid\_recall:.4f}")

            print(f" F1 Score: {valid\_f1:.4f}")

            print(f"\nTime taken: {epoch\_time:.2f} seconds")

            # Log results

            with open(log\_file, 'a') as f:

                f.write(f"\nEpoch {epoch+1}/{num\_epochs}\n")

                f.write(f"Train Loss: {train\_loss:.4f}, Train Accuracy: {train\_acc:.2f}%\n")

                f.write(f"Train Precision: {train\_precision:.4f}, Train Recall: {train\_recall:.4f}, Train F1 Score: {train\_f1:.4f}\n")

                f.write(f"Valid Loss: {valid\_loss:.4f}, Valid Accuracy: {valid\_acc:.2f}%\n")

                f.write(f"Valid Precision: {valid\_precision:.4f}, Valid Recall: {valid\_recall:.4f}, Valid F1 Score: {valid\_f1:.4f}\n")

                f.write(f"Epoch time: {epoch\_time:.2f} seconds\n")

        # Save final model

        torch.save(model.state\_dict(), 'vit\_base\_model.pth')

        # Plot and save metrics

        plt.figure(figsize=(10, 6))

        plt.plot(metrics['train\_loss'], label="Train Loss")

        plt.plot(metrics['valid\_loss'], label="Validation Loss")

        plt.xlabel('Epoch')

        plt.ylabel('Loss')

        plt.legend()

        plt.savefig(f"{timestamp}\_loss\_plot.png")

        plt.figure(figsize=(10, 6))

        plt.plot(metrics['train\_acc'], label="Train Accuracy")

        plt.plot(metrics['valid\_acc'], label="Validation Accuracy")

        plt.xlabel('Epoch')

        plt.ylabel('Accuracy (%)')

        plt.legend()

        plt.savefig(f"{timestamp}\_accuracy\_plot.png")

    except Exception as e:

        print(f"An error occurred: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    train\_vit\_base()