Using device: cuda

Loading JPEG images...

100%|██████████| 50/50 [00:00<00:00, 3055.25it/s]

Found 4479 JPEG images in 50 classes

Epoch 1/20: 100%|██████████| 112/112 [00:46<00:00, 2.39it/s, loss=2.9602, acc=22.27%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.64it/s]

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

Epoch 1/20 Results:

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

Training:

Loss: 2.9602

Accuracy: 22.27%

Precision: 0.2487

Recall: 0.2155

F1 Score: 0.2055

Validation:

Loss: 2.2361

Accuracy: 36.27%

Precision: 0.4060

Recall: 0.3518

F1 Score: 0.3104

Time taken: 54.71 seconds

Epoch 2/20: 100%|██████████| 112/112 [00:42<00:00, 2.61it/s, loss=1.9719, acc=45.74%]

Validation: 100%|██████████| 28/28 [00:06<00:00, 4.04it/s]

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

Epoch 2/20 Results:

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

Training:

Loss: 1.9719

Accuracy: 45.74%

Precision: 0.4650

Recall: 0.4491

F1 Score: 0.4422

Validation:

Loss: 1.8540

Accuracy: 46.09%

Precision: 0.5015

Recall: 0.4537

F1 Score: 0.4266

Time taken: 49.96 seconds

Epoch 3/20: 100%|██████████| 112/112 [00:42<00:00, 2.65it/s, loss=1.5301, acc=56.41%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.61it/s]

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

Epoch 3/20 Results:

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

Training:

Loss: 1.5301

Accuracy: 56.41%

Precision: 0.5615

Recall: 0.5573

F1 Score: 0.5538

Validation:

Loss: 1.6250

Accuracy: 53.46%

Precision: 0.5883

Recall: 0.5432

F1 Score: 0.5191

Time taken: 50.10 seconds

Epoch 4/20: 100%|██████████| 112/112 [00:42<00:00, 2.61it/s, loss=1.2165, acc=66.40%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.44it/s]

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

Epoch 4/20 Results:

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

Training:

Loss: 1.2165

Accuracy: 66.40%

Precision: 0.6616

Recall: 0.6610

F1 Score: 0.6588

Validation:

Loss: 1.4939

Accuracy: 56.47%

Precision: 0.6180

Recall: 0.5680

F1 Score: 0.5486

Time taken: 51.09 seconds

Epoch 5/20: 100%|██████████| 112/112 [00:42<00:00, 2.64it/s, loss=0.9786, acc=72.79%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.57it/s]

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

Epoch 5/20 Results:

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

Training:

Loss: 0.9786

Accuracy: 72.79%

Precision: 0.7293

Recall: 0.7246

F1 Score: 0.7251

Validation:

Loss: 1.3740

Accuracy: 59.71%

Precision: 0.6768

Recall: 0.6045

F1 Score: 0.5875

Time taken: 50.22 seconds

Epoch 6/20: 100%|██████████| 112/112 [00:42<00:00, 2.64it/s, loss=0.7658, acc=78.45%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.34it/s]

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

Epoch 6/20 Results:

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

Training:

Loss: 0.7658

Accuracy: 78.45%

Precision: 0.7846

Recall: 0.7821

F1 Score: 0.7820

Validation:

Loss: 1.2145

Accuracy: 61.83%

Precision: 0.7041

Recall: 0.6213

F1 Score: 0.6171

Time taken: 50.86 seconds

Epoch 7/20: 100%|██████████| 112/112 [00:43<00:00, 2.57it/s, loss=0.6095, acc=83.95%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.55it/s]

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

Epoch 7/20 Results:

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

Training:

Loss: 0.6095

Accuracy: 83.95%

Precision: 0.8385

Recall: 0.8383

F1 Score: 0.8378

Validation:

Loss: 1.0329

Accuracy: 68.08%

Precision: 0.7230

Recall: 0.6808

F1 Score: 0.6784

Time taken: 51.52 seconds

Epoch 8/20: 100%|██████████| 112/112 [00:42<00:00, 2.63it/s, loss=0.5101, acc=86.94%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.65it/s]

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

Epoch 8/20 Results:

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

Training:

Loss: 0.5101

Accuracy: 86.94%

Precision: 0.8688

Recall: 0.8684

F1 Score: 0.8677

Validation:

Loss: 1.0405

Accuracy: 69.87%

Precision: 0.7432

Recall: 0.7050

F1 Score: 0.7053

Time taken: 50.26 seconds

Epoch 9/20: 100%|██████████| 112/112 [00:43<00:00, 2.59it/s, loss=0.3796, acc=90.96%]

Validation: 100%|██████████| 28/28 [00:06<00:00, 4.09it/s]

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

Epoch 9/20 Results:

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

Training:

Loss: 0.3796

Accuracy: 90.96%

Precision: 0.9091

Recall: 0.9077

F1 Score: 0.9076

Validation:

Loss: 0.9727

Accuracy: 71.65%

Precision: 0.7752

Recall: 0.7101

F1 Score: 0.7116

Time taken: 50.19 seconds

Epoch 10/20: 100%|██████████| 112/112 [00:43<00:00, 2.60it/s, loss=0.2699, acc=94.06%]

Validation: 100%|██████████| 28/28 [00:06<00:00, 4.10it/s]

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

Epoch 10/20 Results:

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

Training:

Loss: 0.2699

Accuracy: 94.06%

Precision: 0.9407

Recall: 0.9392

F1 Score: 0.9396

Validation:

Loss: 1.0824

Accuracy: 69.08%

Precision: 0.7230

Recall: 0.7035

F1 Score: 0.6809

Time taken: 50.00 seconds

Epoch 11/20: 100%|██████████| 112/112 [00:42<00:00, 2.63it/s, loss=0.1953, acc=95.45%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.88it/s]

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

Epoch 11/20 Results:

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

Training:

Loss: 0.1953

Accuracy: 95.45%

Precision: 0.9544

Recall: 0.9539

F1 Score: 0.9539

Validation:

Loss: 0.9732

Accuracy: 72.10%

Precision: 0.7716

Recall: 0.7245

F1 Score: 0.7174

Time taken: 49.89 seconds

Epoch 12/20: 100%|██████████| 112/112 [00:42<00:00, 2.63it/s, loss=0.1429, acc=97.26%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.53it/s]

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

Epoch 12/20 Results:

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

Training:

Loss: 0.1429

Accuracy: 97.26%

Precision: 0.9722

Recall: 0.9714

F1 Score: 0.9717

Validation:

Loss: 1.1198

Accuracy: 68.19%

Precision: 0.7419

Recall: 0.6908

F1 Score: 0.6855

Time taken: 50.62 seconds

Epoch 13/20: 100%|██████████| 112/112 [00:42<00:00, 2.65it/s, loss=0.0847, acc=98.91%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.55it/s]

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

Epoch 13/20 Results:

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

Training:

Loss: 0.0847

Accuracy: 98.91%

Precision: 0.9884

Recall: 0.9887

F1 Score: 0.9885

Validation:

Loss: 0.8052

Accuracy: 78.57%

Precision: 0.8112

Recall: 0.7882

F1 Score: 0.7873

Time taken: 50.26 seconds

Epoch 14/20: 100%|██████████| 112/112 [00:43<00:00, 2.60it/s, loss=0.0733, acc=98.97%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.60it/s]

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

Epoch 14/20 Results:

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

Training:

Loss: 0.0733

Accuracy: 98.97%

Precision: 0.9894

Recall: 0.9890

F1 Score: 0.9891

Validation:

Loss: 0.9765

Accuracy: 72.66%

Precision: 0.7764

Recall: 0.7224

F1 Score: 0.7079

Time taken: 50.92 seconds

Epoch 15/20: 100%|██████████| 112/112 [00:42<00:00, 2.65it/s, loss=0.0629, acc=98.74%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.95it/s]

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

Epoch 15/20 Results:

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

Training:

Loss: 0.0629

Accuracy: 98.74%

Precision: 0.9865

Recall: 0.9864

F1 Score: 0.9864

Validation:

Loss: 1.0736

Accuracy: 69.98%

Precision: 0.7774

Recall: 0.6984

F1 Score: 0.7038

Time taken: 49.44 seconds

Epoch 16/20: 100%|██████████| 112/112 [00:42<00:00, 2.61it/s, loss=0.0617, acc=99.25%]

Validation: 100%|██████████| 28/28 [00:06<00:00, 4.11it/s]

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

Epoch 16/20 Results:

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

Training:

Loss: 0.0617

Accuracy: 99.25%

Precision: 0.9927

Recall: 0.9922

F1 Score: 0.9924

Validation:

Loss: 1.3126

Accuracy: 64.40%

Precision: 0.7509

Recall: 0.6507

F1 Score: 0.6437

Time taken: 49.84 seconds

Epoch 17/20: 100%|██████████| 112/112 [00:42<00:00, 2.62it/s, loss=0.0331, acc=99.64%]

Validation: 100%|██████████| 28/28 [00:06<00:00, 4.02it/s]

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

Epoch 17/20 Results:

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

Training:

Loss: 0.0331

Accuracy: 99.64%

Precision: 0.9963

Recall: 0.9964

F1 Score: 0.9963

Validation:

Loss: 0.8145

Accuracy: 76.45%

Precision: 0.7811

Recall: 0.7743

F1 Score: 0.7624

Time taken: 49.83 seconds

Epoch 18/20: 100%|██████████| 112/112 [00:42<00:00, 2.64it/s, loss=0.0268, acc=99.61%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.55it/s]

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

Epoch 18/20 Results:

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

Training:

Loss: 0.0268

Accuracy: 99.61%

Precision: 0.9959

Recall: 0.9959

F1 Score: 0.9959

Validation:

Loss: 0.7608

Accuracy: 78.91%

Precision: 0.8031

Recall: 0.7925

F1 Score: 0.7864

Time taken: 50.26 seconds

Epoch 19/20: 100%|██████████| 112/112 [00:43<00:00, 2.60it/s, loss=0.0167, acc=99.86%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.56it/s]

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

Epoch 19/20 Results:

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

Training:

Loss: 0.0167

Accuracy: 99.86%

Precision: 0.9986

Recall: 0.9985

F1 Score: 0.9985

Validation:

Loss: 0.7955

Accuracy: 77.46%

Precision: 0.7997

Recall: 0.7792

F1 Score: 0.7759

Time taken: 51.03 seconds

Epoch 20/20: 100%|██████████| 112/112 [00:42<00:00, 2.66it/s, loss=0.0153, acc=99.94%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.67it/s]

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

Epoch 20/20 Results:

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

Training:

Loss: 0.0153

Accuracy: 99.94%

Precision: 0.9994

Recall: 0.9994

F1 Score: 0.9994

Validation:

Loss: 0.8530

Accuracy: 76.12%

Precision: 0.7895

Recall: 0.7679

F1 Score: 0.7620

Time taken: 49.82 seconds

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

Training Summary:

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

Best Validation Loss: 0.7608

Final Train Accuracy: 99.94%

Final Valid Accuracy: 76.12%

Final Train Precision: 0.9994

Final Train Recall: 0.9994

Final Train F1 Score: 0.9994

Final Valid Precision: 0.7895

Final Valid Recall: 0.7679

Final Valid F1 Score: 0.7620

Results saved to: training\_results\_resnet18\_20241105\_013544.txt

A graph of different colored lines

Description automatically generated with medium confidence

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\_resnet18():

    # Create timestamp for logging

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

    log\_file = f'training\_results\_resnet18\_{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: ResNet-18\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 model

        model = models.resnet18(weights=None)

        num\_ftrs = model.fc.in\_features

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

        model = model.to(device)

        # Initialize metrics

       # 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"Time taken: {epoch\_time:.2f} seconds\n")

                f.write("-"\*50 + "\n")

            # Save best model

            if valid\_loss < best\_valid\_loss:

                best\_valid\_loss = valid\_loss

                torch.save({

                    'epoch': epoch,

                    'model\_state\_dict': model.state\_dict(),

                    'optimizer\_state\_dict': optimizer.state\_dict(),

                    'train\_loss': train\_loss,

                    'valid\_loss': valid\_loss,

                    'train\_acc': train\_acc,

                    'valid\_acc': valid\_acc,

                }, "best\_resnet18\_model.pth")

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

                    f.write(f"\nNew best model saved! Validation Loss: {valid\_loss:.4f}\n")

        # Print and log final summary

        print(f"\n{'='\*50}")

        print("Training Summary:")

        print(f"{'='\*50}")

        print(f"Best Validation Loss: {best\_valid\_loss:.4f}")

        print(f"Final Train Accuracy: {metrics['train\_acc'][-1]:.2f}%")

        print(f"Final Valid Accuracy: {metrics['valid\_acc'][-1]:.2f}%")

        print(f"Final Train Precision: {metrics['train\_precision'][-1]:.4f}")

        print(f"Final Train Recall: {metrics['train\_recall'][-1]:.4f}")

        print(f"Final Train F1 Score: {metrics['train\_f1'][-1]:.4f}")

        print(f"Final Valid Precision: {metrics['valid\_precision'][-1]:.4f}")

        print(f"Final Valid Recall: {metrics['valid\_recall'][-1]:.4f}")

        print(f"Final Valid F1 Score: {metrics['valid\_f1'][-1]:.4f}")

        print(f"\nResults saved to: {log\_file}")

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

            f.write(f"\n{'='\*50}\n")

            f.write("Training Summary:\n")

            f.write(f"{'='\*50}\n")

            f.write(f"Best Validation Loss: {best\_valid\_loss:.4f}\n")

            f.write(f"Final Train Accuracy: {metrics['train\_acc'][-1]:.2f}%\n")

            f.write(f"Final Valid Accuracy: {metrics['valid\_acc'][-1]:.2f}%\n")

            f.write(f"Final Train Precision: {metrics['train\_precision'][-1]:.4f}\n")

            f.write(f"Final Train Recall: {metrics['train\_recall'][-1]:.4f}\n")

            f.write(f"Final Train F1 Score: {metrics['train\_f1'][-1]:.4f}\n")

            f.write(f"Final Valid Precision: {metrics['valid\_precision'][-1]:.4f}\n")

            f.write(f"Final Valid Recall: {metrics['valid\_recall'][-1]:.4f}\n")

            f.write(f"Final Valid F1 Score: {metrics['valid\_f1'][-1]:.4f}\n")

            f.write("\nTraining completed successfully!\n")

        # Plot metrics

        plt.figure(figsize=(16, 8))

        plt.subplot(2, 3, 1)

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

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

        plt.xlabel('Epoch')

        plt.ylabel('Loss')

        plt.legend()

        plt.title('ResNet-18 - Loss')

        plt.subplot(2, 3, 2)

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

        plt.plot(metrics['valid\_acc'], label='Valid Acc')

        plt.xlabel('Epoch')

        plt.ylabel('Accuracy')

        plt.legend()

        plt.title('ResNet-18 - Accuracy')

        plt.subplot(2, 3, 3)

        plt.plot(metrics['train\_precision'], label='Train Precision')

        plt.plot(metrics['valid\_precision'], label='Valid Precision')

        plt.xlabel('Epoch')

        plt.ylabel('Precision')

        plt.legend()

        plt.title('ResNet-18 - Precision')

        plt.subplot(2, 3, 4)

        plt.plot(metrics['train\_recall'], label='Train Recall')

        plt.plot(metrics['valid\_recall'], label='Valid Recall')

        plt.xlabel('Epoch')

        plt.ylabel('Recall')

        plt.legend()

        plt.title('ResNet-18 - Recall')

        plt.subplot(2, 3, 5)

        plt.plot(metrics['train\_f1'], label='Train F1 Score')

        plt.plot(metrics['valid\_f1'], label='Valid F1 Score')

        plt.xlabel('Epoch')

        plt.ylabel('F1 Score')

        plt.legend()

        plt.title('ResNet-18 - F1 Score')

        # Save and display plot

        plt.tight\_layout()

        plt.savefig(f'training\_metrics\_resnet18\_{timestamp}.png')

        plt.show()

    except Exception as e:

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

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

            f.write(f"\nAn error occurred: {e}\n")

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

    train\_resnet18()