Using device: cuda

Loading JPEG images...

100%|██████████| 50/50 [00:00<00:00, 6107.56it/s]

Found 4479 JPEG images in 50 classes

/usr/local/lib/python3.10/dist-packages/torchvision/models/\_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.

warnings.warn(

/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=EfficientNet\_B0\_Weights.IMAGENET1K\_V1`. You can also use `weights=EfficientNet\_B0\_Weights.DEFAULT` to get the most up-to-date weights.

warnings.warn(msg)

Downloading: "<https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth>" to /root/.cache/torch/hub/checkpoints/efficientnet\_b0\_rwightman-7f5810bc.pth

100%|██████████| 20.5M/20.5M [00:00<00:00, 119MB/s]

Epoch 1/20: 100%|██████████| 112/112 [00:46<00:00, 2.42it/s, loss=3.0911, acc=41.31%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.12it/s]

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

Epoch 1/20 Results:

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

Training:

Loss: 3.0911

Accuracy: 41.31%

Precision: 0.5027

Recall: 0.3943

F1 Score: 0.3992

Validation:

Loss: 1.7266

Accuracy: 79.69%

Precision: 0.8398

Recall: 0.7895

F1 Score: 0.7775

Time taken: 55.36 seconds

Epoch 2/20: 100%|██████████| 112/112 [00:45<00:00, 2.48it/s, loss=1.1498, acc=85.01%]

Validation: 100%|██████████| 28/28 [00:09<00:00, 2.86it/s]

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

Epoch 2/20 Results:

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

Training:

Loss: 1.1498

Accuracy: 85.01%

Precision: 0.8643

Recall: 0.8414

F1 Score: 0.8451

Validation:

Loss: 0.5512

Accuracy: 91.18%

Precision: 0.9200

Recall: 0.9156

F1 Score: 0.9094

Time taken: 54.98 seconds

Epoch 3/20: 100%|██████████| 112/112 [00:45<00:00, 2.48it/s, loss=0.4584, acc=92.41%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.22it/s]

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

Epoch 3/20 Results:

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

Training:

Loss: 0.4584

Accuracy: 92.41%

Precision: 0.9254

Recall: 0.9221

F1 Score: 0.9226

Validation:

Loss: 0.3261

Accuracy: 92.63%

Precision: 0.9275

Recall: 0.9293

F1 Score: 0.9225

Time taken: 53.95 seconds

Epoch 4/20: 100%|██████████| 112/112 [00:45<00:00, 2.47it/s, loss=0.2651, acc=95.12%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.26it/s]

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

Epoch 4/20 Results:

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

Training:

Loss: 0.2651

Accuracy: 95.12%

Precision: 0.9521

Recall: 0.9509

F1 Score: 0.9512

Validation:

Loss: 0.2469

Accuracy: 94.31%

Precision: 0.9407

Recall: 0.9409

F1 Score: 0.9392

Time taken: 53.92 seconds

Epoch 5/20: 100%|██████████| 112/112 [00:45<00:00, 2.45it/s, loss=0.1628, acc=97.18%]

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

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

Epoch 5/20 Results:

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

Training:

Loss: 0.1628

Accuracy: 97.18%

Precision: 0.9722

Recall: 0.9711

F1 Score: 0.9714

Validation:

Loss: 0.2080

Accuracy: 94.87%

Precision: 0.9469

Recall: 0.9477

F1 Score: 0.9461

Time taken: 53.55 seconds

Epoch 6/20: 100%|██████████| 112/112 [00:45<00:00, 2.45it/s, loss=0.1143, acc=97.93%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.45it/s]

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

Epoch 6/20 Results:

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

Training:

Loss: 0.1143

Accuracy: 97.93%

Precision: 0.9795

Recall: 0.9790

F1 Score: 0.9791

Validation:

Loss: 0.2038

Accuracy: 94.31%

Precision: 0.9427

Recall: 0.9437

F1 Score: 0.9413

Time taken: 53.97 seconds

Epoch 7/20: 100%|██████████| 112/112 [00:46<00:00, 2.41it/s, loss=0.0857, acc=98.66%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.15it/s]

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

Epoch 7/20 Results:

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

Training:

Loss: 0.0857

Accuracy: 98.66%

Precision: 0.9867

Recall: 0.9859

F1 Score: 0.9862

Validation:

Loss: 0.2043

Accuracy: 94.42%

Precision: 0.9404

Recall: 0.9473

F1 Score: 0.9413

Time taken: 55.36 seconds

Epoch 8/20: 100%|██████████| 112/112 [00:44<00:00, 2.50it/s, loss=0.0592, acc=98.80%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.19it/s]

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

Epoch 8/20 Results:

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

Training:

Loss: 0.0592

Accuracy: 98.80%

Precision: 0.9881

Recall: 0.9876

F1 Score: 0.9878

Validation:

Loss: 0.1894

Accuracy: 95.42%

Precision: 0.9557

Recall: 0.9522

F1 Score: 0.9519

Time taken: 53.57 seconds

Epoch 9/20: 100%|██████████| 112/112 [00:44<00:00, 2.51it/s, loss=0.0485, acc=99.13%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.20it/s]

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

Epoch 9/20 Results:

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

Training:

Loss: 0.0485

Accuracy: 99.13%

Precision: 0.9911

Recall: 0.9909

F1 Score: 0.9910

Validation:

Loss: 0.1934

Accuracy: 95.09%

Precision: 0.9498

Recall: 0.9507

F1 Score: 0.9492

Time taken: 53.44 seconds

Epoch 10/20: 100%|██████████| 112/112 [00:44<00:00, 2.51it/s, loss=0.0389, acc=99.16%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.25it/s]

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

Epoch 10/20 Results:

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

Training:

Loss: 0.0389

Accuracy: 99.16%

Precision: 0.9921

Recall: 0.9921

F1 Score: 0.9921

Validation:

Loss: 0.1926

Accuracy: 94.64%

Precision: 0.9468

Recall: 0.9465

F1 Score: 0.9453

Time taken: 53.37 seconds

Epoch 11/20: 100%|██████████| 112/112 [00:45<00:00, 2.45it/s, loss=0.0350, acc=99.55%]

Validation: 100%|██████████| 28/28 [00:07<00:00, 3.59it/s]

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

Epoch 11/20 Results:

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

Training:

Loss: 0.0350

Accuracy: 99.55%

Precision: 0.9958

Recall: 0.9955

F1 Score: 0.9956

Validation:

Loss: 0.1841

Accuracy: 95.09%

Precision: 0.9514

Recall: 0.9501

F1 Score: 0.9497

Time taken: 53.46 seconds

Epoch 12/20: 100%|██████████| 112/112 [00:46<00:00, 2.41it/s, loss=0.0290, acc=99.39%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.36it/s]

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

Epoch 12/20 Results:

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

Training:

Loss: 0.0290

Accuracy: 99.39%

Precision: 0.9937

Recall: 0.9938

F1 Score: 0.9938

Validation:

Loss: 0.2033

Accuracy: 95.09%

Precision: 0.9507

Recall: 0.9528

F1 Score: 0.9490

Time taken: 54.86 seconds

Epoch 13/20: 100%|██████████| 112/112 [00:45<00:00, 2.47it/s, loss=0.0257, acc=99.55%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.18it/s]

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

Epoch 13/20 Results:

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

Training:

Loss: 0.0257

Accuracy: 99.55%

Precision: 0.9957

Recall: 0.9957

F1 Score: 0.9957

Validation:

Loss: 0.2037

Accuracy: 95.09%

Precision: 0.9521

Recall: 0.9495

F1 Score: 0.9493

Time taken: 54.26 seconds

Epoch 14/20: 100%|██████████| 112/112 [00:45<00:00, 2.49it/s, loss=0.0192, acc=99.64%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.17it/s]

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

Epoch 14/20 Results:

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

Training:

Loss: 0.0192

Accuracy: 99.64%

Precision: 0.9964

Recall: 0.9962

F1 Score: 0.9963

Validation:

Loss: 0.2051

Accuracy: 94.87%

Precision: 0.9474

Recall: 0.9489

F1 Score: 0.9467

Time taken: 53.93 seconds

Epoch 15/20: 100%|██████████| 112/112 [00:44<00:00, 2.49it/s, loss=0.0182, acc=99.80%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.19it/s]

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

Epoch 15/20 Results:

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

Training:

Loss: 0.0182

Accuracy: 99.80%

Precision: 0.9980

Recall: 0.9981

F1 Score: 0.9980

Validation:

Loss: 0.2175

Accuracy: 94.53%

Precision: 0.9469

Recall: 0.9487

F1 Score: 0.9451

Time taken: 53.81 seconds

Epoch 16/20: 100%|██████████| 112/112 [00:45<00:00, 2.48it/s, loss=0.0163, acc=99.75%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.28it/s]

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

Epoch 16/20 Results:

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

Training:

Loss: 0.0163

Accuracy: 99.75%

Precision: 0.9975

Recall: 0.9976

F1 Score: 0.9975

Validation:

Loss: 0.2019

Accuracy: 95.20%

Precision: 0.9536

Recall: 0.9545

F1 Score: 0.9524

Time taken: 53.81 seconds

Epoch 17/20: 100%|██████████| 112/112 [00:46<00:00, 2.40it/s, loss=0.0158, acc=99.75%]

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

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

Epoch 17/20 Results:

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

Training:

Loss: 0.0158

Accuracy: 99.75%

Precision: 0.9977

Recall: 0.9973

F1 Score: 0.9975

Validation:

Loss: 0.2015

Accuracy: 95.09%

Precision: 0.9494

Recall: 0.9508

F1 Score: 0.9488

Time taken: 54.56 seconds

Epoch 18/20: 100%|██████████| 112/112 [00:45<00:00, 2.45it/s, loss=0.0135, acc=99.78%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.37it/s]

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

Epoch 18/20 Results:

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

Training:

Loss: 0.0135

Accuracy: 99.78%

Precision: 0.9978

Recall: 0.9979

F1 Score: 0.9978

Validation:

Loss: 0.1972

Accuracy: 95.42%

Precision: 0.9540

Recall: 0.9547

F1 Score: 0.9534

Time taken: 54.02 seconds

Epoch 19/20: 100%|██████████| 112/112 [00:45<00:00, 2.46it/s, loss=0.0162, acc=99.67%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.20it/s]

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

Epoch 19/20 Results:

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

Training:

Loss: 0.0162

Accuracy: 99.67%

Precision: 0.9969

Recall: 0.9969

F1 Score: 0.9969

Validation:

Loss: 0.2070

Accuracy: 95.76%

Precision: 0.9561

Recall: 0.9597

F1 Score: 0.9561

Time taken: 54.35 seconds

Epoch 20/20: 100%|██████████| 112/112 [00:44<00:00, 2.49it/s, loss=0.0114, acc=99.80%]

Validation: 100%|██████████| 28/28 [00:08<00:00, 3.20it/s]

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

Epoch 20/20 Results:

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

Training:

Loss: 0.0114

Accuracy: 99.80%

Precision: 0.9981

Recall: 0.9982

F1 Score: 0.9981

Validation:

Loss: 0.2198

Accuracy: 95.31%

Precision: 0.9520

Recall: 0.9532

F1 Score: 0.9505

Time taken: 53.70 seconds

Final Results after Training:

Training Accuracy: 99.80%

Validation Accuracy: 95.31%

Model saved to efficientnet\_b0\_model\_20241105\_015821.pth

A group of graphs showing different types of data

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\_efficientnet\_b0():

    # Create timestamp for logging

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

    log\_file = f'training\_results\_efficientnet\_b0\_{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: EfficientNet-B0\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.efficientnet\_b0(pretrained=True)

        num\_ftrs = model.classifier[1].in\_features

        model.classifier[1] = 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("="\*50 + "\n")

        # Final results

        print("\nFinal Results after Training:")

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

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

        # Save the model

        model\_path = f"efficientnet\_b0\_model\_{timestamp}.pth"

        torch.save(model.state\_dict(), model\_path)

        print(f"Model saved to {model\_path}")

        # Plotting metrics (optional)

        plot\_metrics(metrics)

    except Exception as e:

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

def plot\_metrics(metrics):

    # Plot training and validation metrics

    epochs = range(1, len(metrics['train\_loss']) + 1)

    fig, axes = plt.subplots(2, 2, figsize=(12, 10))

    # Loss plot

    axes[0, 0].plot(epochs, metrics['train\_loss'], label='Train Loss')

    axes[0, 0].plot(epochs, metrics['valid\_loss'], label='Validation Loss')

    axes[0, 0].set\_title('Loss')

    axes[0, 0].set\_xlabel('Epochs')

    axes[0, 0].set\_ylabel('Loss')

    axes[0, 0].legend()

    # Accuracy plot

    axes[0, 1].plot(epochs, metrics['train\_acc'], label='Train Accuracy')

    axes[0, 1].plot(epochs, metrics['valid\_acc'], label='Validation Accuracy')

    axes[0, 1].set\_title('Accuracy')

    axes[0, 1].set\_xlabel('Epochs')

    axes[0, 1].set\_ylabel('Accuracy (%)')

    axes[0, 1].legend()

    # Precision plot

    axes[1, 0].plot(epochs, metrics['train\_precision'], label='Train Precision')

    axes[1, 0].plot(epochs, metrics['valid\_precision'], label='Validation Precision')

    axes[1, 0].set\_title('Precision')

    axes[1, 0].set\_xlabel('Epochs')

    axes[1, 0].set\_ylabel('Precision')

    axes[1, 0].legend()

    # F1 Score plot

    axes[1, 1].plot(epochs, metrics['train\_f1'], label='Train F1 Score')

    axes[1, 1].plot(epochs, metrics['valid\_f1'], label='Validation F1 Score')

    axes[1, 1].set\_title('F1 Score')

    axes[1, 1].set\_xlabel('Epochs')

    axes[1, 1].set\_ylabel('F1 Score')

    axes[1, 1].legend()

    plt.tight\_layout()

    plt.show()

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

    train\_efficientnet\_b0()