```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

```
!pip install torch torchvision efficientnet_pytorch matplotlib
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dis
Collecting efficientnet_pytorch
  Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/p
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/pyth
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3
Building wheels for collected packages: efficientnet_pytorch
  Building wheel for efficientnet_pytorch (setup.py) ... done
  Created wheel for efficientnet_pytorch: filename=efficientnet_pytorch-0.7
  Stored in directory: /root/.cache/pip/wheels/03/3f/e9/911b1bc46869644912b
Successfully built efficientnet_pytorch
Installing collected packages: efficientnet_pytorch
Successfully installed efficientnet_pytorch-0.7.1
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from efficientnet_pytorch import EfficientNet
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
import os
import warnings as w
w.filterwarnings('ignore')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
# Image transformations for augmentation, especially for classes with fewer sam
train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```
dataset = datasets.ImageFolder("/content/drive/MyDrive/Food Classification data

# Split the dataset into training and testing sets (e.g., 80% train, 20% test)
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_data, test_data = random_split(dataset, [train_size, test_size])

# Apply test transforms to the test dataset
test_data.dataset.transform = test_transforms
```

```
train_labels = [dataset.targets[i] for i in train_data.indices]
class_counts = Counter(train_labels)
print("Class counts in training data:", class_counts)

# Oversampling or data augmentation strategy based on class imbalance
class_weights = 1. / np.array([class_counts[i] for i in range(len(class_counts)
sample_weights = [class_weights[label] for label in train_labels]

# Weighted sampler for handling imbalanced classes
weighted_sampler = torch.utils.data.WeightedRandomSampler(sample_weights, len(s
```

```
Class counts in training data: Counter({7: 815, 2: 812, 8: 797, 27: 796, 21
```

```
batch_size = 32

train_loader = DataLoader(train_data, batch_size=batch_size, sampler=weighted_s
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

```
model = EfficientNet.from_pretrained('efficientnet-b0')

# Modify the final layer to match the number of food classes
num_classes = 28
model._fc = nn.Linear(model._fc.in_features, num_classes)

# Move the model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
```

```
Downloading: "https://github.com/lukemelas/EfficientNet-PyTorch/releases/do
100%|██████████| 20.4M/20.4M [00:00<00:00, 384MB/s]
Loaded pretrained weights for efficientnet-b0
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
def train_model(model, criterion, optimizer, train_loader, test_loader, num_epo
    train_loss_history = []
    test_loss_history = []
    train_acc_history = []
    test_acc_history = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
```

```
        total = 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        train_loss_history.append(running_loss / len(train_loader))
        train_acc_history.append(100 * correct / total)

        # Validation
        model.eval()
        val_loss = 0.0
        val_correct = 0
        val_total = 0
        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        test_loss_history.append(val_loss / len(test_loader))
        test_acc_history.append(100 * val_correct / val_total)

        print(f'Epoch [{epoch+1}/{num_epochs}], '
              f'Train Loss: {running_loss/len(train_loader):.4f}, '
              f'Train Acc: {100 * correct / total:.2f}%, '
              f'Val Loss: {val_loss/len(test_loader):.4f}, '
              f'Val Acc: {100 * val_correct / val_total:.2f}%')

    return train_loss_history, test_loss_history, train_acc_history, test_acc_h
```

```
num_epochs = 10
train_loss, test_loss, train_acc, test_acc = train_model(model, criterion, opti


def plot_metrics(train_loss, test_loss, train_acc, test_acc):
    epochs = range(1, len(train_loss) + 1)

    plt.figure(figsize=(12, 4))

    # Plot loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_loss, 'b', label='Train Loss')
    plt.plot(epochs, test_loss, 'r', label='Test Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    # Plot accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_acc, 'b', label='Train Accuracy')
    plt.plot(epochs, test_acc, 'r', label='Test Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy (%)')
    plt.legend()

    plt.show()

# Cell 11: Call the plotting function
plot_metrics(train_loss, test_loss, train_acc, test_acc)
```

```
def continue_training_with_early_stopping(model, criterion, optimizer, train_lo
    best_acc = 0.0  # To track the best validation accuracy
    for epoch in range(start_epoch, start_epoch + num_more_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
```

```
        total = 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        # Validation phase
        model.eval()
        val_loss = 0.0
        val_correct = 0
        val_total = 0
        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_acc = 100 * val_correct / val_total

        print(f'Epoch [{epoch+1}/{start_epoch + num_more_epochs}], '
              f'Train Acc: {100 * correct / total:.2f}%, '
              f'Val Loss: {val_loss/len(test_loader):.4f}, '
              f'Val Acc: {val_acc:.2f}%')

        # Early stopping if validation accuracy reaches or exceeds the threshol
        if val_acc >= early_stop_acc:
            print(f'Early stopping triggered at epoch {epoch+1} with validatior
            break

# Continue training with early stopping
start_epoch = 10  # Continue from the 6th epoch
num_more_epochs = 10  # Number of additional epochs to train
continue_training_with_early_stopping(model, criterion, optimizer, train_loader
```

```
Epoch [11/20], Train Acc: 96.58%, Val Loss: 0.7156, Val Acc: 82.90%
Epoch [12/20], Train Acc: 96.93%, Val Loss: 0.6305, Val Acc: 84.06%
Epoch [13/20], Train Acc: 97.02%, Val Loss: 0.7211, Val Acc: 82.32%
Epoch [14/20], Train Acc: 97.09%, Val Loss: 0.7753, Val Acc: 80.04%
Epoch [15/20], Train Acc: 97.32%, Val Loss: 0.8426, Val Acc: 82.07%
Epoch [16/20], Train Acc: 98.22%, Val Loss: 0.5235, Val Acc: 86.00%
Early stopping triggered at epoch 16 with validation accuracy: 86.00%
```

```
# Cell 13: Save the trained model
torch.save(model.state_dict(), 'food_classification_model.pth')
print("Model saved as 'food_classification_model.pth'")
```

```
Model saved as 'food_classification_model.pth'
```

```
import os
import pickle

# Define the path to the dataset
dataset_path = "/content/drive/MyDrive/Food Classification dataset1/"

# Get class names from folder names
class_names = sorted(os.listdir(dataset_path))

# Create a dictionary mapping class indices to class names
class_mapping = {idx: class_name for idx, class_name in enumerate(class_names)}

# Save the class mapping as a pickle file
with open('class_mapping.pkl', 'wb') as f:
    pickle.dump(class_mapping, f)
```

```
class_mapping
```

```
{0: 'Donut',
 1: 'Hot Dog',
 2: 'apple_pie',
 3: 'burger',
 4: 'butter_naan',
 5: 'chai',
 6: 'chapati',
 7: 'cheesecake',
 8: 'chicken_curry',
 9: 'chole_bhature',
 10: 'dal_makhani',
 11: 'dhokla',
 12: 'fried_rice',
 13: 'ice_cream',
 14: 'idli',
 15: 'jalebi',
 16: 'kaathi_rolls',
 17: 'kadai_paneer',
 18: 'kulfi',
 19: 'masala_dosa',
 20: 'momos',
 21: 'omelette',
 22: 'paani_puri',
 23: 'pakode',
 24: 'pav_bhaji',
 25: 'pizza',
 26: 'samosa',
 27: 'sushi'}
```