In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
import numpy as np
import shutil
import torch
import random
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from PIL import Image
from timeit import default_timer as timer
from tqdm.auto import tqdm
from torchvision import transforms,datasets, models, transforms
from torch.utils.data import TensorDataset,DataLoader, Dataset, WeightedRandomSampler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [3]:

```
device='cuda' if torch.cuda.is_available() else 'cpu'
device
```

Out[3]:

```
'cuda'
```

In [4]:

```
trainset="/kaggle/input/fruits/fruits-360_dataset_original-size/fruits-360-original-size/Training"
testset="/kaggle/input/fruits/fruits-360_dataset_original-size/fruits-360-original-size/Test"

transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5), ( 0.5))
])
batch_size = 32
```

In [5]:

```
train_data = datasets.ImageFolder(root=trainset, transform=transform)
test_data = datasets.ImageFolder(root=testset, transform=transform)
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
```

In [6]:

```
class_names = train_data.classes
print("Class names: ",class_names)
print(len(class_names))
```

```
Class names:  ['apple_6', 'apple_braeburn_1', 'apple_crimson_snow_1', 'apple_golden_1', 'apple_golden_2', 'apple_golden_3', 'apple_granny_smith_1', 'apple_hit_1', 'apple_pink_lady_1', 'apple_red_1', 'apple_red_2', 'apple_red_3', 'apple_red_delicios_1', 'apple_red_yellow_1', 'apple_rotten_1', 'cabbage_white_1', 'carrot_1', 'cucumber_1', 'cucumber_3', 'eggplant_long_1', 'pear_1', 'pear_3', 'zucchini_1', 'zucchini_dark_1']
24
```

In [ ]:

```
img,label = next(iter(train_data))
img_permute = img.permute(1, 2, 0)

print(f"Original shape: {img.shape} -> [color_channels, height, width]")
print(f"Image permute shape: {img_permute.shape} -> [height, width, color_channels]")

# Plot the image
plt.figure(figsize=(10, 7))
plt.imshow(img.permute(1, 2, 0))
plt.axis("off")
```

In [13]:

```
model=models.efficientnet_b0(pretrained=True).to(device)
#model=models.densenet201(pretrained=True).to(device)
for params in model.parameters():
            params.requires_grad = False
```

In [14]:

```
output_shape = len(class_names)
model.classifier = torch.nn.Sequential(
    torch.nn.Linear(in_features=1280, out_features=720,bias=True),
    torch.nn.Linear(in_features=720, out_features=360,bias=True),
    torch.nn.Linear(in_features=360, out_features=output_shape,bias=True)
).to(device)
```

In [11]:

```
def train_step(model: torch.nn.Module, dataloader: torch.utils.data.DataLoader,
               loss_fn: torch.nn.Module, optimizer: torch.optim.Optimizer):
    model.train()
    train_loss, train_acc = 0, 0
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)
        y_pred = model(X)
        loss = loss_fn(y_pred, y)
        train_loss += loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1)
        train_acc += (y_pred_class == y).sum().item()/len(y_pred)
    train_loss = train_loss / len(dataloader)
    train_acc = train_acc / len(dataloader)
    return train_loss, train_acc

def test_step(model: torch.nn.Module,dataloader: torch.utils.data.DataLoader,
              loss_fn: torch.nn.Module):
    model.eval()

    test_loss, test_acc = 0, 0
    with torch.inference_mode():
        for batch, (X, y) in enumerate(dataloader):
            X, y = X.to(device), y.to(device)
            test_pred_logits = model(X)
            loss = loss_fn(test_pred_logits, y)
            test_loss += loss.item()
            test_pred_labels = test_pred_logits.argmax(dim=1)
            test_acc += ((test_pred_labels == y).sum().item()/len(test_pred_labels))

    test_loss = test_loss / len(dataloader)
    test_acc = test_acc / len(dataloader)
    return test_loss, test_acc


def train(model: torch.nn.Module,train_dataloader: torch.utils.data.DataLoader,
          test_dataloader: torch.utils.data.DataLoader,optimizer: torch.optim.Optimizer,
          loss_fn: torch.nn.Module = nn.CrossEntropyLoss(),epochs: int = 5):

    results = {"train_loss": [],"train_acc": [],"test_loss": [],"test_acc": []}

    for epoch in tqdm(range(epochs)):
        train_loss, train_acc = train_step(model=model,dataloader=train_dataloader,loss_
fn=loss_fn,optimizer=optimizer)
        test_loss, test_acc = test_step(model=model,dataloader=test_dataloader,loss_fn=l
oss_fn)

        print( f"Epoch: {epoch+1} | "f"train_loss: {train_loss:.4f} | "f"train_acc: {tra
in_acc:.4f} | "
            f"test_loss: {test_loss:.4f} | "f"test_acc: {test_acc:.4f}" )

        results["train_loss"].append(train_loss)
        results["train_acc"].append(train_acc)
        results["test_loss"].append(test_loss)
        results["test_acc"].append(test_acc)

    return results
```

In [15]:

```
torch.manual_seed(42)
torch.cuda.manual_seed(42)
NUM_EPOCHS = 5
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(), lr=1e-3)
start_time = timer()

model_results = train(model=model,train_dataloader=train_dataloader,
                      test_dataloader=test_dataloader,optimizer=optimizer,
                      loss_fn=loss_fn,epochs=NUM_EPOCHS)

end_time = timer()
print(f"Total training time: {end_time-start_time:.3f} seconds")
```

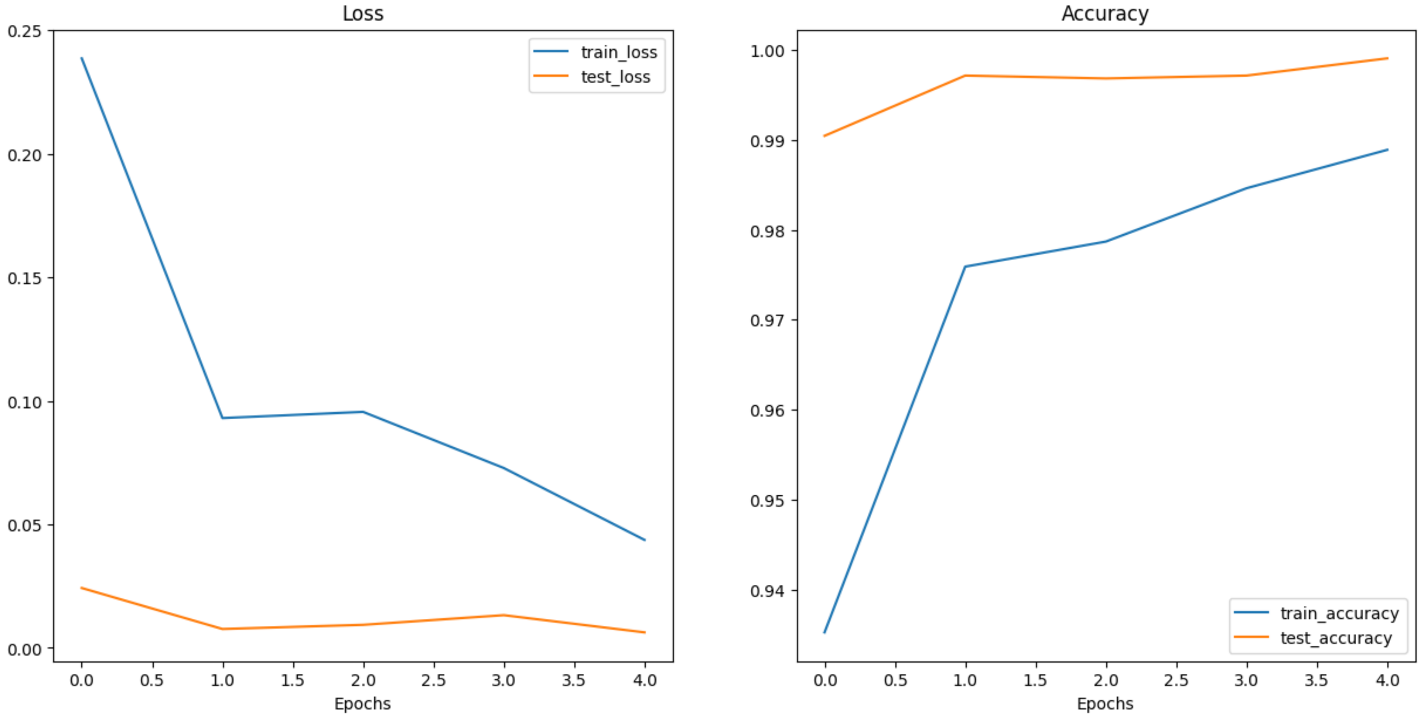
```
Epoch: 1 | train_loss: 0.2386 | train_acc: 0.9353 | test_loss: 0.0242 | test_acc: 0.9904
Epoch: 2 | train_loss: 0.0930 | train_acc: 0.9759 | test_loss: 0.0077 | test_acc: 0.9971
Epoch: 3 | train_loss: 0.0955 | train_acc: 0.9787 | test_loss: 0.0093 | test_acc: 0.9968
Epoch: 4 | train_loss: 0.0728 | train_acc: 0.9846 | test_loss: 0.0132 | test_acc: 0.9971
Epoch: 5 | train_loss: 0.0437 | train_acc: 0.9889 | test_loss: 0.0063 | test_acc: 0.9990
Total training time: 288.384 seconds
```

In [16]:

```
def plot_loss_curves(results):
    results = dict(list(model_results.items()))
    loss = results['train_loss']
    test_loss = results['test_loss']
    accuracy = results['train_acc']
    test_accuracy = results['test_acc']

    epochs = range(len(results['train_loss']))
    plt.figure(figsize=(15, 7))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, loss, label='train_loss')
    plt.plot(epochs, test_loss, label='test_loss')
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, accuracy, label='train_accuracy')
    plt.plot(epochs, test_accuracy, label='test_accuracy')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.legend();
```

In [17]:

```
plot_loss_curves(model_results)
```



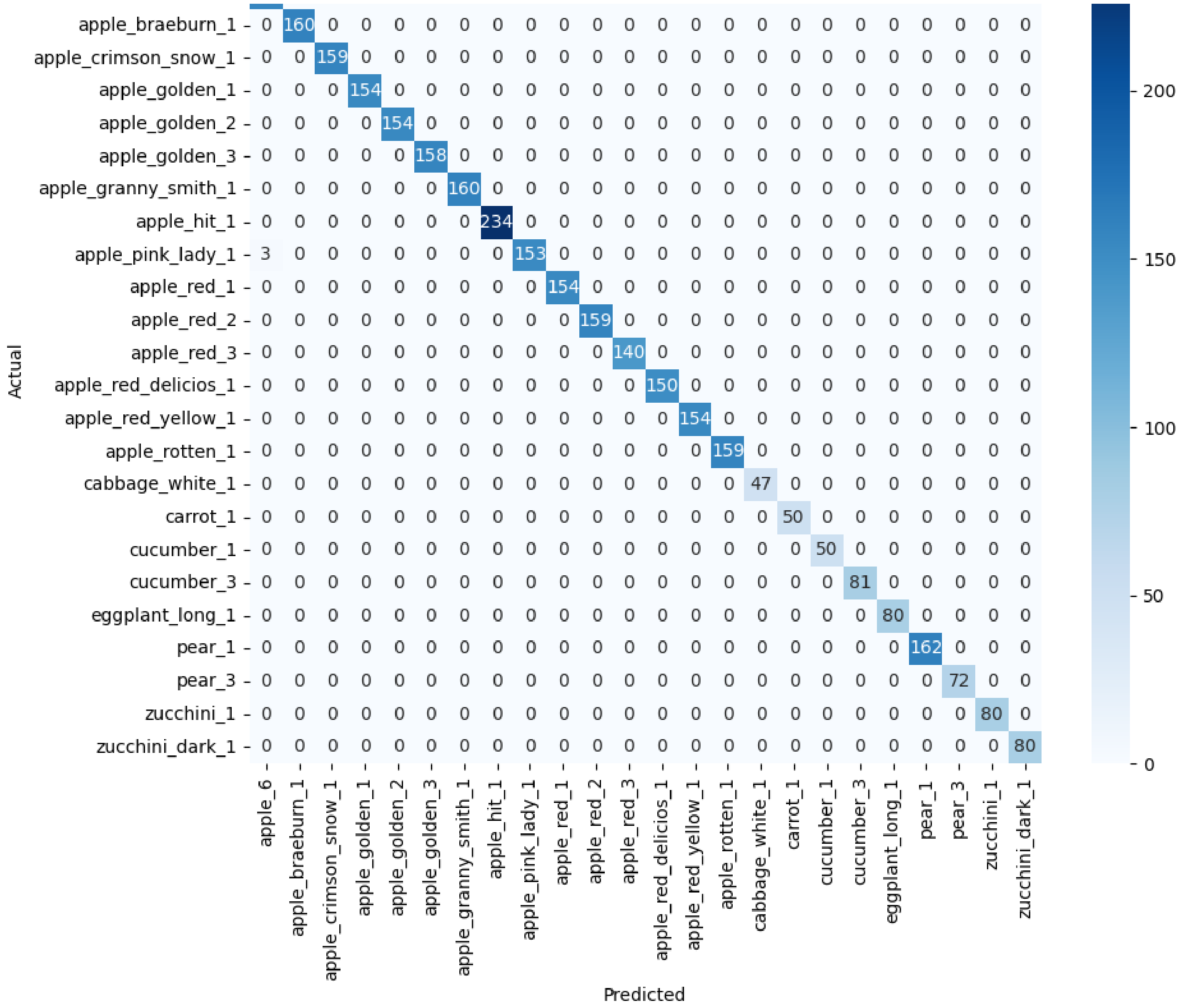
In [18]:

```
y_true = []
y_pred = []

model.eval()
with torch.inference_mode():
  for X, y in test_dataloader:
    X, y = X.to(device), y.to(device)
    test_pred_logits = model(X)
    test_pred_labels = test_pred_logits.argmax(dim=1)

    y_true.extend(y.cpu().numpy().tolist())
    y_pred.extend(test_pred_labels.cpu().numpy().tolist())

accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted')  # You can change 'weigh
ted' to 'macro', 'micro' based on your needs
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')


print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.9990
Precision: 0.9991
Recall: 0.9990
F1-score: 0.9990
```

In [20]:

```
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=
class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



	apple_6	apple_braeburn_1	apple_crimson_snow_1	apple_golden_1	apple_golden_2	apple_golden_3	apple_granny_smith_1	apple_hit_1	apple_pink_lady_1	apple_red_1	apple_red_2	apple_red_3	apple_red_delicios_1	apple_red_yellow_1	apple_rotten_1	cabbage_white_1	carrot_1	cucumber_1	cucumber_3	eggplant_long_1	pear_1	pear_3	zucchini_1	zucchini_dark_1
apple_braeburn_1	0	160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_crimson_snow_1	0	0	159	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_golden_1	0	0	0	154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_golden_2	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_golden_3	0	0	0	0	0	158	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_granny_smith_1	0	0	0	0	0	0	160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_hit_1	0	0	0	0	0	0	0	234	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_pink_lady_1	3	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_red_1	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_red_2	0	0	0	0	0	0	0	0	0	0	159	0	0	0	0	0	0	0	0	0	0	0	0	0
apple_red_3	0	0	0	0	0	0	0	0	0	0	0	140	0	0	0	0	0	0	0	0	0	0	0	0
apple_red_delicios_1	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	0	0
apple_red_yellow_1	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0
apple_rotten_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	159	0	0	0	0	0	0	0	0	0
cabbage_white_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0
carrot_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0
cucumber_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0
cucumber_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	81	0	0	0	0	0
eggplant_long_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80	0	0	0	0
pear_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	162	0	0	0
pear_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	72	0	0
zucchini_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80	0
zucchini_dark_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80

```
In [ ]:
```