In [ ]:
```
import torch
from torch.utils.data import Dataset,DataLoader
import numpy as np
import os
from torchvision import datasets
from torchvision.transforms import transforms
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import cv2 as cv
from sklearn.cluster import KMeans
```

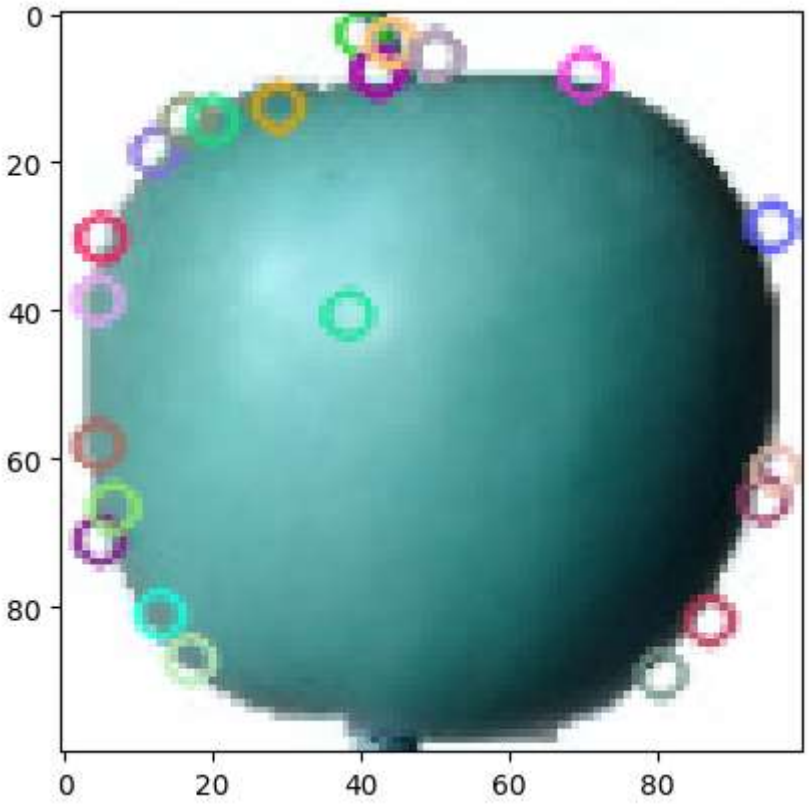
In [ ]:
```
# Data downloaded from https://github.com/Horea94/Fruit-Images-Dataset

img_folder = "/Data/Training/"
classes = {
            "Apple Granny Smith" : 0,
            "Apple Red 1" : 1,
            "Apple Golden 1" : 2,
            "Dates" : 3,
            "Nut Pecan" : 4,
            "Tangelo" : 5,
            "Kohlrabi": 6,
            "Plum" : 7,
            "Peach" : 8,
            "Walnut" : 9
        }
```

a. Traditional feature extraction

In [ ]:
```
#SIFT https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html


import cv2 as cv


img = cv.imread(r"Data/Training/Apple Granny Smith/0_100.jpg")
sift = cv.SIFT_create()

kp = sift.detect(img,None)
```

```
kp,des  = sift.compute(img,kp)
img=cv.drawKeypoints(img,kp,img)

import matplotlib.pyplot as plt

plt.imshow(img)
```

Out[ ]: <matplotlib.image.AxesImage at 0x234a9fa2c80>



b. New keypoints dataset KP that consists of all the keypoints from all the training images

In [ ]:
```
#SIFT extraction of keypoints https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

# Function to extract keypoints from an image
def extract_keypoints(image_path, sift):
    img = cv.imread(image_path)
```

```
    kp = sift.detect(img, None)
    kp, des = sift.compute(img, kp)
    return kp, des

# Initialize SIFT object
sift = cv.SIFT_create()

# Initialize a dictionary to store keypoints for each class
train_keypoints = {}

# Iterate through each class directory
for class_name,idx in classes.items():
    train_keypoints[class_name] = []  # Initialize an empty list for the class
    class_dir = r"Data/Training/" + class_name + "/"
    # Iterate through each image in the class directory
    for filename in os.listdir(class_dir):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Extract keypoints from the image
            image_path = os.path.join(class_dir, filename)
            kp, _ = extract_keypoints(image_path, sift)
            # Add keypoints to the list for the current class
            train_keypoints[class_name].append([list(kp),[idx]])

train_kp_x = []
train_kp_y = []

for i in train_keypoints.items():
    for X in i[1]:
        train_kp_x.append(X[0])
        train_kp_y.append(X[1])
```

c. K-Mean Clustering with K = 100 on the keypoints

In [ ]:
```
#K-Means clustering https://scikit-learn.org/stable/modules/clustering.html

# Extract coordinate values from train_kp_x
coords = []
for kp_list in train_kp_x:
    coords.extend([kp.pt for kp in kp_list])

# Convert the list of tuples to a NumPy array
```

```
data = np.array(coords)

# Initialize KMeans with 100 clusters
kmeans = KMeans(n_clusters=100, random_state=0)

# Fit the KMeans model to your data
kmeans.fit(data)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Get the cluster centers
centroids = kmeans.cluster_centers_
```

```
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarni
ng: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

d. Construct a 100-D vector for each Image.

In [ ]:
```
#SIFT extraction of keypoints https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
#K-Means clustering https://scikit-learn.org/stable/modules/clustering.html

# Initialize a dictionary to store the 100-D vectors for each image
train_image_vectors = {}

# Iterate through each class directory
for class_name, idx in classes.items():
    class_dir = r"Data/Training/" + class_name + "/"

    # Iterate through each image in the class directory
    for filename in os.listdir(class_dir):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Extract keypoints and descriptors from the image
            image_path = os.path.join(class_dir, filename)
            kp, _ = extract_keypoints(image_path, sift)

            # Extract coordinate values from keypoints
            coords = np.array([kp_.pt for kp_ in kp])

            # Predict the cluster labels for each keypoint coordinate
```

```
            cluster_labels = kmeans.predict(coords)

            # Create a 100-D vector for the image by counting the number of keypoints in each cluster
            vector = np.zeros(100)
            for label in cluster_labels:
                vector[label] += 1

            # Add the 100-D vector to the dictionary with the image path as the key
            train_image_vectors[image_path] = vector
```

e. Creating a new 100-D dataset D

In [ ]:
```
dataset_train = []
labels_train = []

for image_path, vector in train_image_vectors.items():
    class_name = image_path.split("/")[-2]  # Extract the class name from the image path
    label = classes[class_name]  # Get the corresponding label for the class

    dataset_train.append(vector)
    labels_train.append(label)

# Convert dataset_train and labels_train to NumPy arrays
dataset_train = np.array(dataset_train)
labels_train = np.array(labels_train)
```

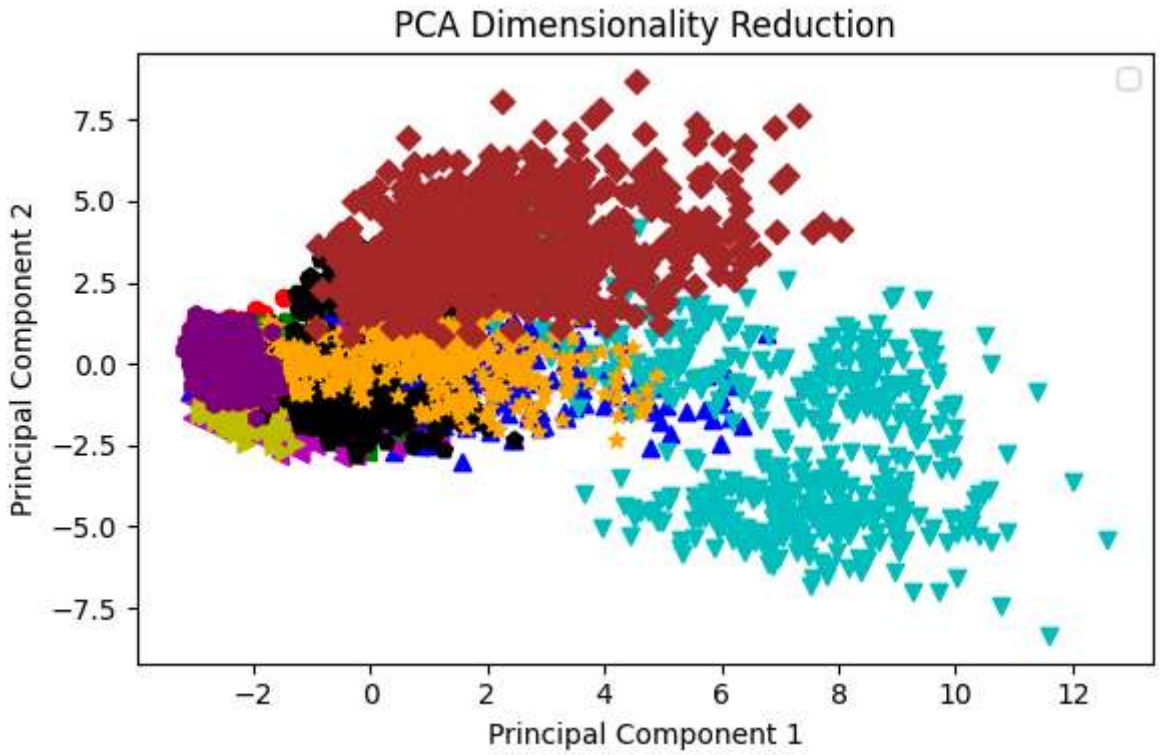
f. (i) Dimensionality reduction.

In [ ]:
```
# PCA https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

from sklearn.decomposition import PCA
import matplotlib.pyplot as plt


pca = PCA(n_components=2)
dataset_pca = pca.fit_transform(dataset_train)
```

f. (ii) Plotting the 2 components

In [ ]:
```
# Create a dictionary to map class labels to colors/symbols
colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k', 'orange', 'purple', 'brown']
symbols = ['o', 's', '^', 'v', '<', '>', 'p', '*', 'h', 'D']
class_colors = {label: color for label, color in zip(classes.values(), colors)}
class_symbols = {label: symbol for label, symbol in zip(classes.values(), symbols)}

# Plot the 2D points using different colors/symbols for each class
fig, ax = plt.subplots(figsize=(6, 4))  # Adjust the figure size as needed
for i, label in enumerate(labels_train):
    ax.scatter(dataset_pca[i, 0], dataset_pca[i, 1], c=class_colors[label], marker=class_symbols[label])

ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_title('PCA Dimensionality Reduction')
ax.legend()

plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored wh
en legend() is called with no argument.
```



4. Processing the test Images

In [ ]:
```
#SIFT extraction of keypoints https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
#K-Means clustering https://scikit-learn.org/stable/modules/clustering.html

# Initialize a dictionary to store the 100-D vectors for each image
test_image_vectors = {}

# Iterate through each class directory
for class_name, idx in classes.items():
    class_dir = r"Data/Test/" + class_name + "/"

    # Iterate through each image in the class directory
    for filename in os.listdir(class_dir):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Extract keypoints and descriptors from the image
            image_path = os.path.join(class_dir, filename)
```

```
            kp, _ = extract_keypoints(image_path, sift)

            # Extract coordinate values from keypoints
            coords = np.array([kp_.pt for kp_ in kp])

            # Predict the cluster labels for each keypoint coordinate
            cluster_labels = kmeans.predict(coords)

            # Create a 100-D vector for the image by counting the number of keypoints in each cluster
            vector = np.zeros(100)
            for label in cluster_labels:
                vector[label] += 1

            # Add the 100-D vector to the dictionary with the image path as the key
            test_image_vectors[image_path] = vector

# Create a new 100-D dataset D
dataset_test = []
labels_test = []

for image_path, vector in test_image_vectors.items():
    class_name = image_path.split("/")[-2]  # Extract the class name from the image path
    label = classes[class_name]  # Get the corresponding label for the class

    dataset_test.append(vector)
    labels_test.append(label)

# Convert dataset_test and labels_test to NumPy arrays
dataset_test = np.array(dataset_test)
labels_test = np.array(labels_test)
```

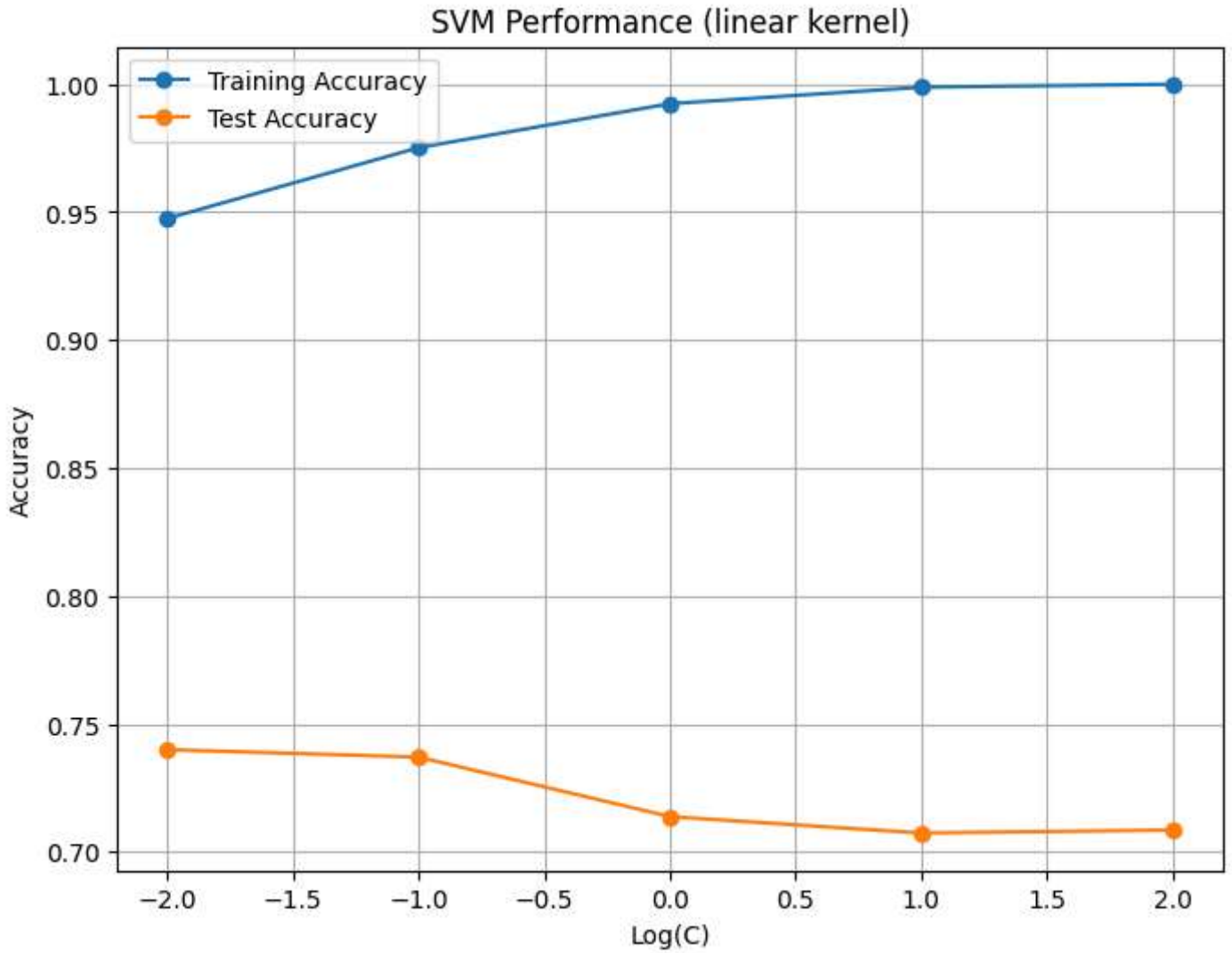
5. SVM for c_values = [0.01, 0.1, 1.0, 10, 100] and kernels = ['linear', 'rbf', 'poly', 'sigmoid']

In [ ]:
```
# SVM https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html


from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import numpy as np
```

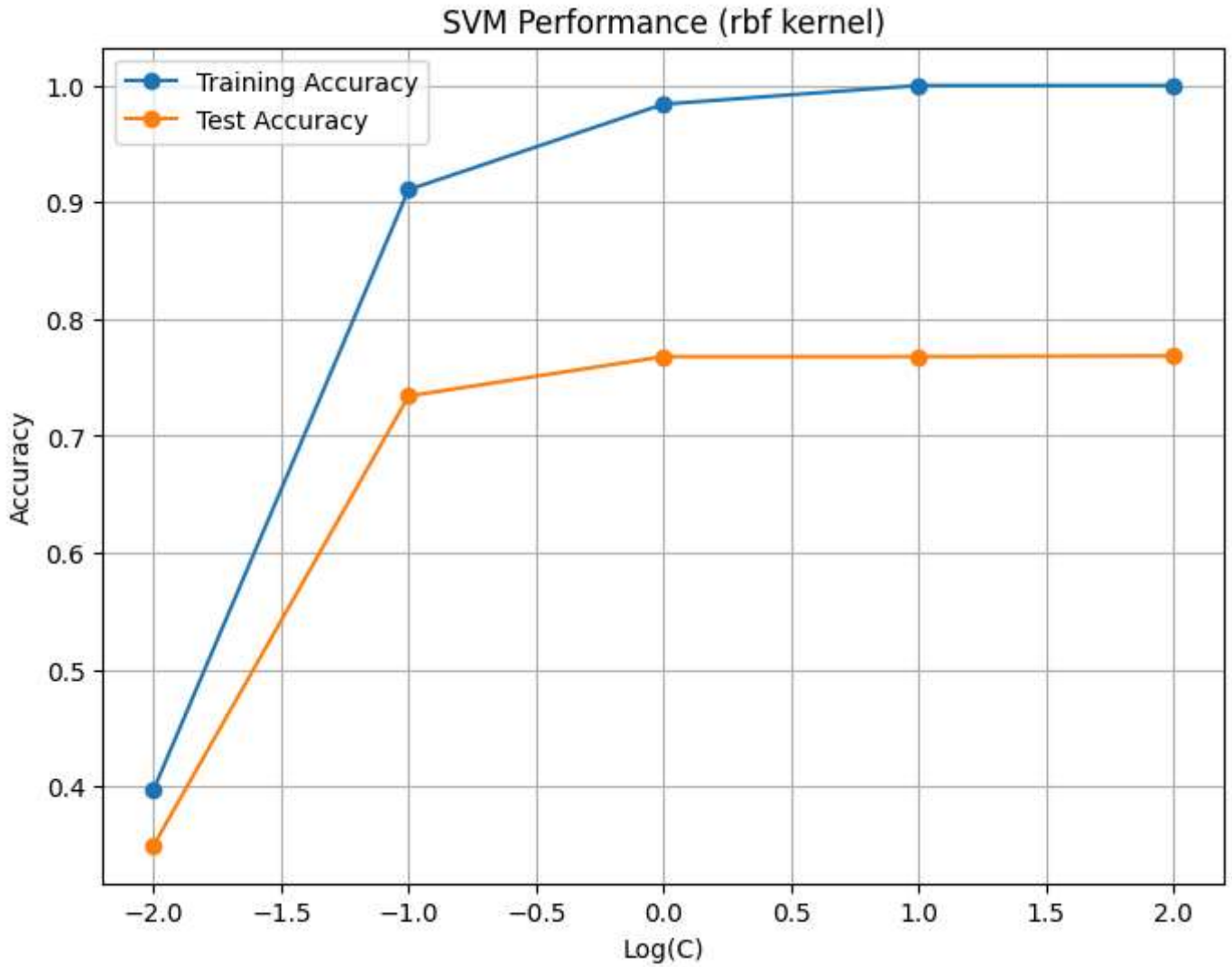
```
# Define the C parameters to test
c_values = [0.01, 0.1, 1.0, 10, 100]
log_c_values = np.log10(c_values)
kernels = ['linear', 'rbf', 'poly', 'sigmoid']

# Create dictionaries to store the best accuracy for each kernel
best_train_accuracy = {}
best_test_accuracy = {}

# Perform model selection and training for each kernel
for kernel in kernels:
    print(f"Training SVM with kernel: {kernel}")

    train_accuracies = []
    test_accuracies = []

    for c in c_values:
        # Create and train the SVM model
        svm = SVC(C=c, kernel=kernel)
        svm.fit(dataset_train, labels_train)

        # Evaluate the model on the training data
        train_predictions = svm.predict(dataset_train)
        train_accuracy = accuracy_score(labels_train, train_predictions)
        train_accuracies.append(train_accuracy)

        # Evaluate the model on the test data
        test_predictions = svm.predict(dataset_test)
        test_accuracy = accuracy_score(labels_test, test_predictions)
        test_accuracies.append(test_accuracy)

    # Plot the performance graph for the current kernel
    plt.figure(figsize=(8, 6))
    plt.plot(log_c_values, train_accuracies, marker='o', label='Training Accuracy')
    plt.plot(log_c_values, test_accuracies, marker='o', label='Test Accuracy')
    plt.xlabel('Log(C)')
    plt.ylabel('Accuracy')
    plt.title(f'SVM Performance ({kernel} kernel)')
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
    # Store the best accuracy for the current kernel
    best_train_accuracy[kernel] = max(train_accuracies)
    best_test_accuracy[kernel] = max(test_accuracies)

# Plot the best performance comparison graph
plt.figure(figsize=(8, 6))
plt.scatter(kernels, [best_train_accuracy[k] for k in kernels], label='Training Accuracy')
plt.scatter(kernels, [best_test_accuracy[k] for k in kernels], label='Test Accuracy')
plt.xlabel('Kernel')
plt.ylabel('Best Accuracy')
plt.title('SVM Best Performance Comparison')
plt.legend()
plt.show()
```

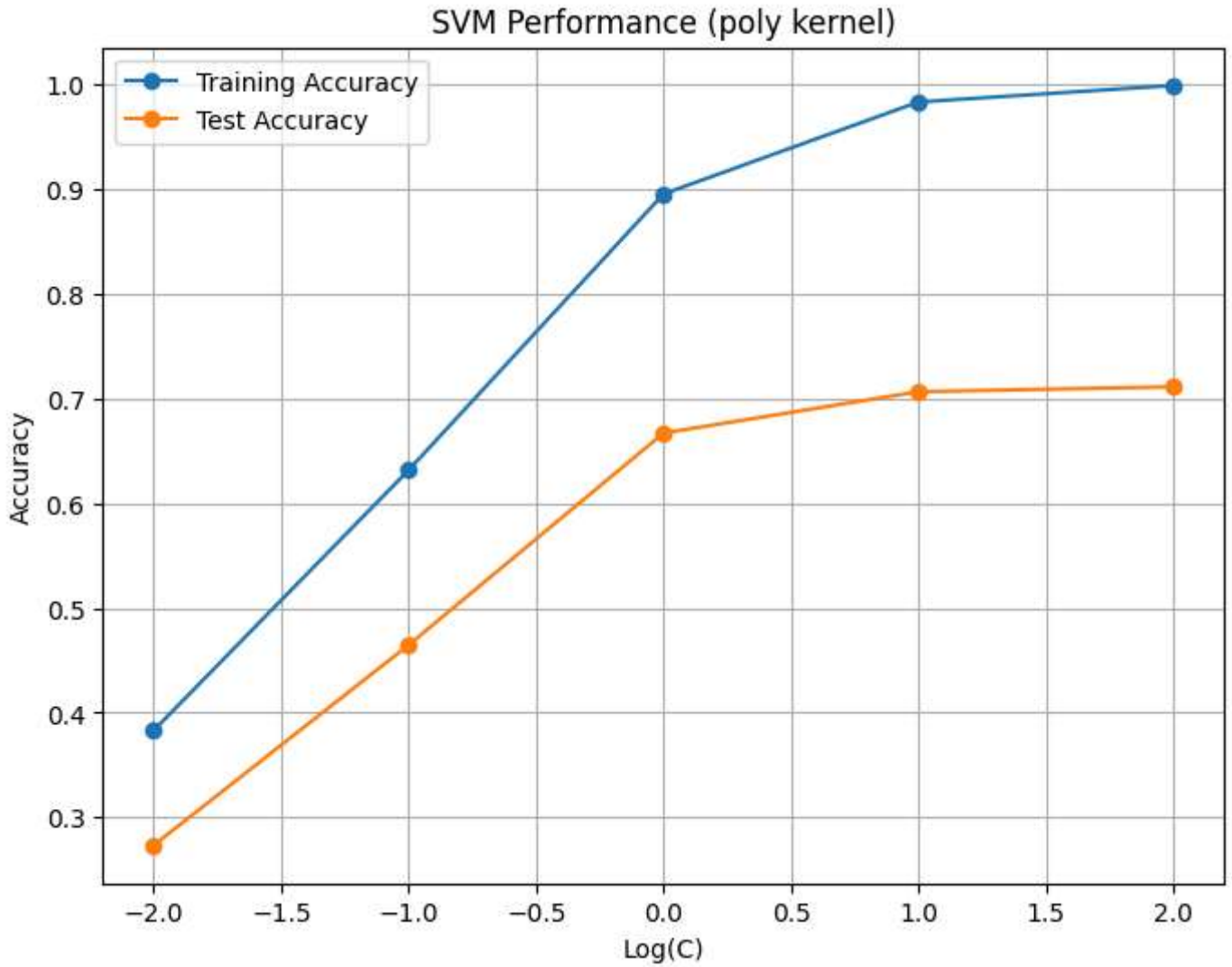
```
Training SVM with kernel: linear
```



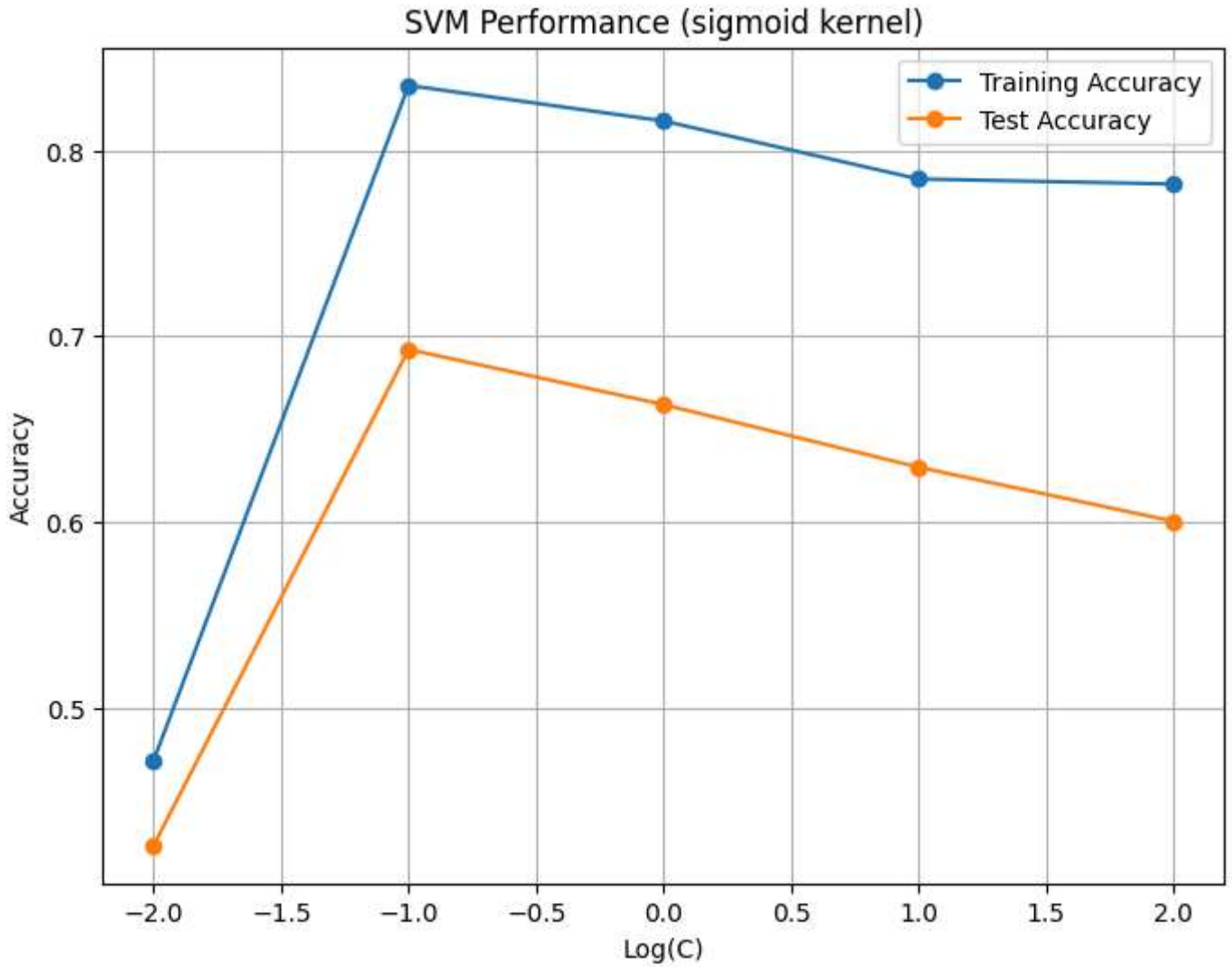

```
Training SVM with kernel: rbf
```

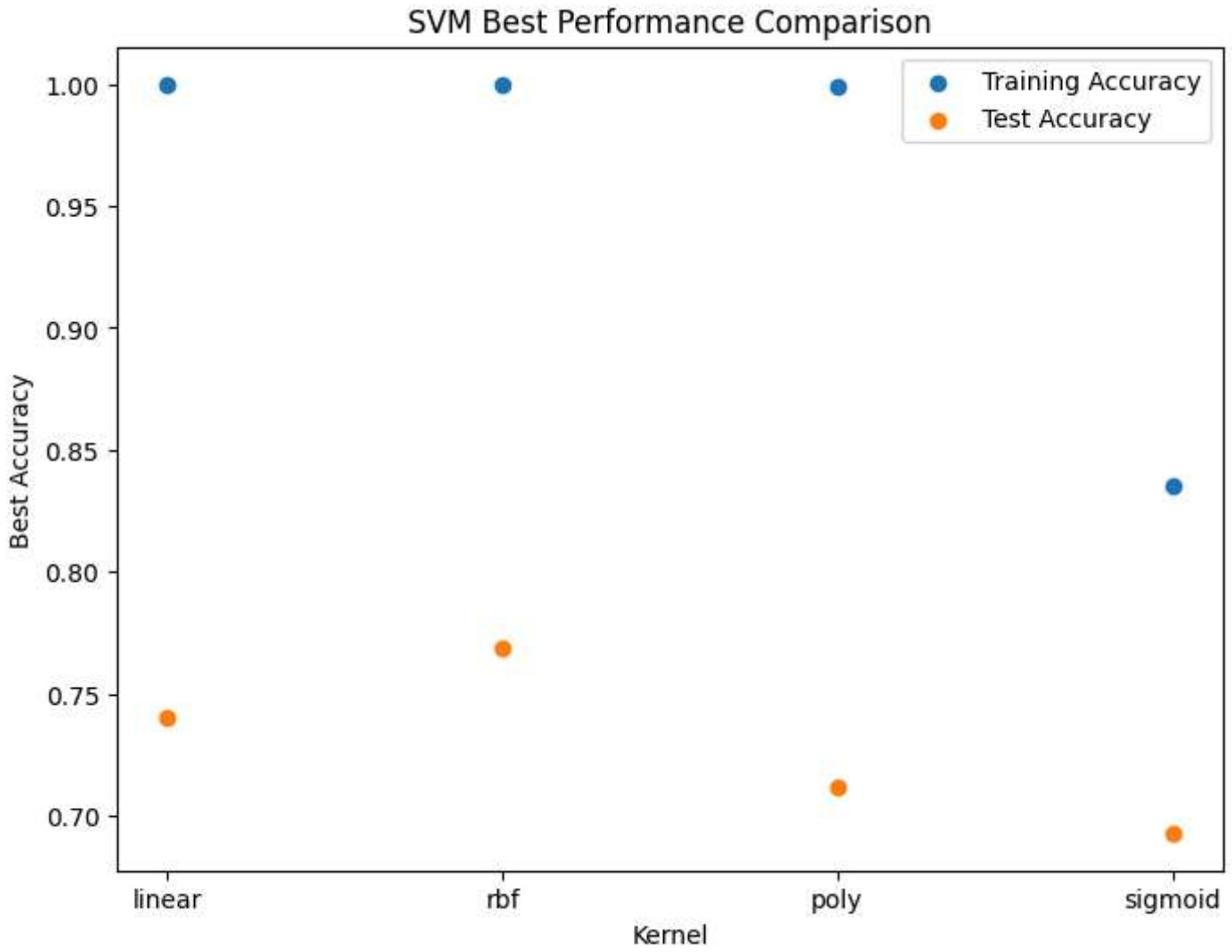



```
Training SVM with kernel: poly
```




```
Training SVM with kernel: sigmoid
```


SVM Performance (sigmoid kernel)
Training Accuracy
Test Accuracy
Accuracy
0.8
0.7
0.6
0.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Log(C)



6,7,8. Creating a custom dataset using Pytroch for the original train and test Images.

In [ ]:
```
# Custom Dataset https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

import os
from PIL import Image
from torch.utils.data import Dataset
from torchvision import transforms
```

```
class FruitDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.classes = [
            "Apple Granny Smith",
            "Apple Red 1",
            "Apple Golden 1",
            "Dates",
            "Nut Pecan",
            "Tangelo",
            "Kohlrabi",
            "Plum",
            "Peach",
            "Walnut"
        ]
        self.class_to_idx = {cls_name: i for i, cls_name in enumerate(self.classes)}

        self.samples = []
        for cls_name in self.classes:
            cls_dir = os.path.join(data_dir, cls_name)
            for img_name in os.listdir(cls_dir):
                img_path = os.path.join(cls_dir, img_name)
                self.samples.append((img_path, self.class_to_idx[cls_name]))

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, index):
        img_path, label = self.samples[index]
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image, label
```

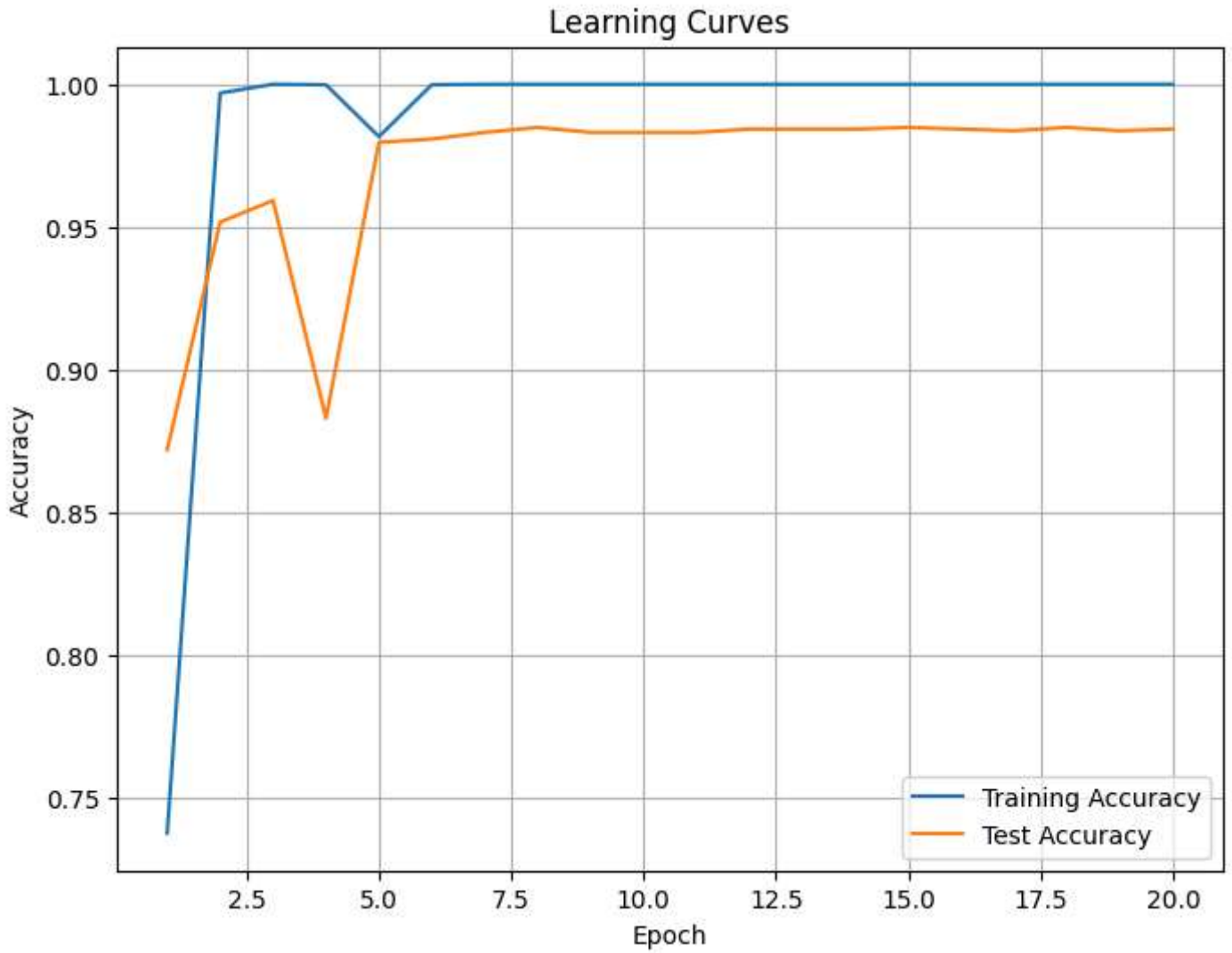
6. Simple CNN

In [ ]:
```python
# Model creation https://pytorch.org/tutorials/beginner/introyt/modelsyt_tutorial.html

import os
from PIL import Image
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt


# Define the data transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Create instances of the custom dataset
train_dataset = FruitDataset('Data/Training', transform=transform)
test_dataset = FruitDataset('Data/Test', transform=transform)

# Create data loaders
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Define the CNN model
class FruitCNN(nn.Module):
    def __init__(self, num_classes):
        super(FruitCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(8, 8, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(8 * 112 * 112, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, num_classes)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.relu(self.conv2(x))
```

```
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x

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

# Create an instance of the CNN model
num_classes = len(train_dataset.classes)
model = FruitCNN(num_classes).to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

# Train the model
num_epochs = 20
train_accuracies = []
test_accuracies = []

for epoch in range(num_epochs):
    model.train()
    train_correct = 0
    train_total = 0

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

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

        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()
```

```
    train_accuracy = train_correct / train_total
    train_accuracies.append(train_accuracy)

    model.eval()
    test_correct = 0
    test_total = 0

    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_accuracy = test_correct / test_total
    test_accuracies.append(test_accuracy)

    print(f"Epoch [{epoch+1}/{num_epochs}], Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}"

# Plot the learning curves
plt.figure(figsize=(8, 6))
plt.plot(range(1, num_epochs+1), train_accuracies, label='Training Accuracy')
plt.plot(range(1, num_epochs+1), test_accuracies, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Learning Curves')
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch [1/20], Train Accuracy: 0.7378, Test Accuracy: 0.8720
Epoch [2/20], Train Accuracy: 0.9969, Test Accuracy: 0.9517
Epoch [3/20], Train Accuracy: 1.0000, Test Accuracy: 0.9593
Epoch [4/20], Train Accuracy: 0.9998, Test Accuracy: 0.8831
Epoch [5/20], Train Accuracy: 0.9817, Test Accuracy: 0.9796
Epoch [6/20], Train Accuracy: 0.9998, Test Accuracy: 0.9808
Epoch [7/20], Train Accuracy: 1.0000, Test Accuracy: 0.9831
Epoch [8/20], Train Accuracy: 1.0000, Test Accuracy: 0.9849
Epoch [9/20], Train Accuracy: 1.0000, Test Accuracy: 0.9831
Epoch [10/20], Train Accuracy: 1.0000, Test Accuracy: 0.9831
Epoch [11/20], Train Accuracy: 1.0000, Test Accuracy: 0.9831
Epoch [12/20], Train Accuracy: 1.0000, Test Accuracy: 0.9843
Epoch [13/20], Train Accuracy: 1.0000, Test Accuracy: 0.9843
Epoch [14/20], Train Accuracy: 1.0000, Test Accuracy: 0.9843
Epoch [15/20], Train Accuracy: 1.0000, Test Accuracy: 0.9849
Epoch [16/20], Train Accuracy: 1.0000, Test Accuracy: 0.9843
Epoch [17/20], Train Accuracy: 1.0000, Test Accuracy: 0.9837
Epoch [18/20], Train Accuracy: 1.0000, Test Accuracy: 0.9849
Epoch [19/20], Train Accuracy: 1.0000, Test Accuracy: 0.9837
Epoch [20/20], Train Accuracy: 1.0000, Test Accuracy: 0.9843
```



7. Transfer Learning via Ferature Extraction

(a) Pre trained Resnet18 (all layers are frozen).

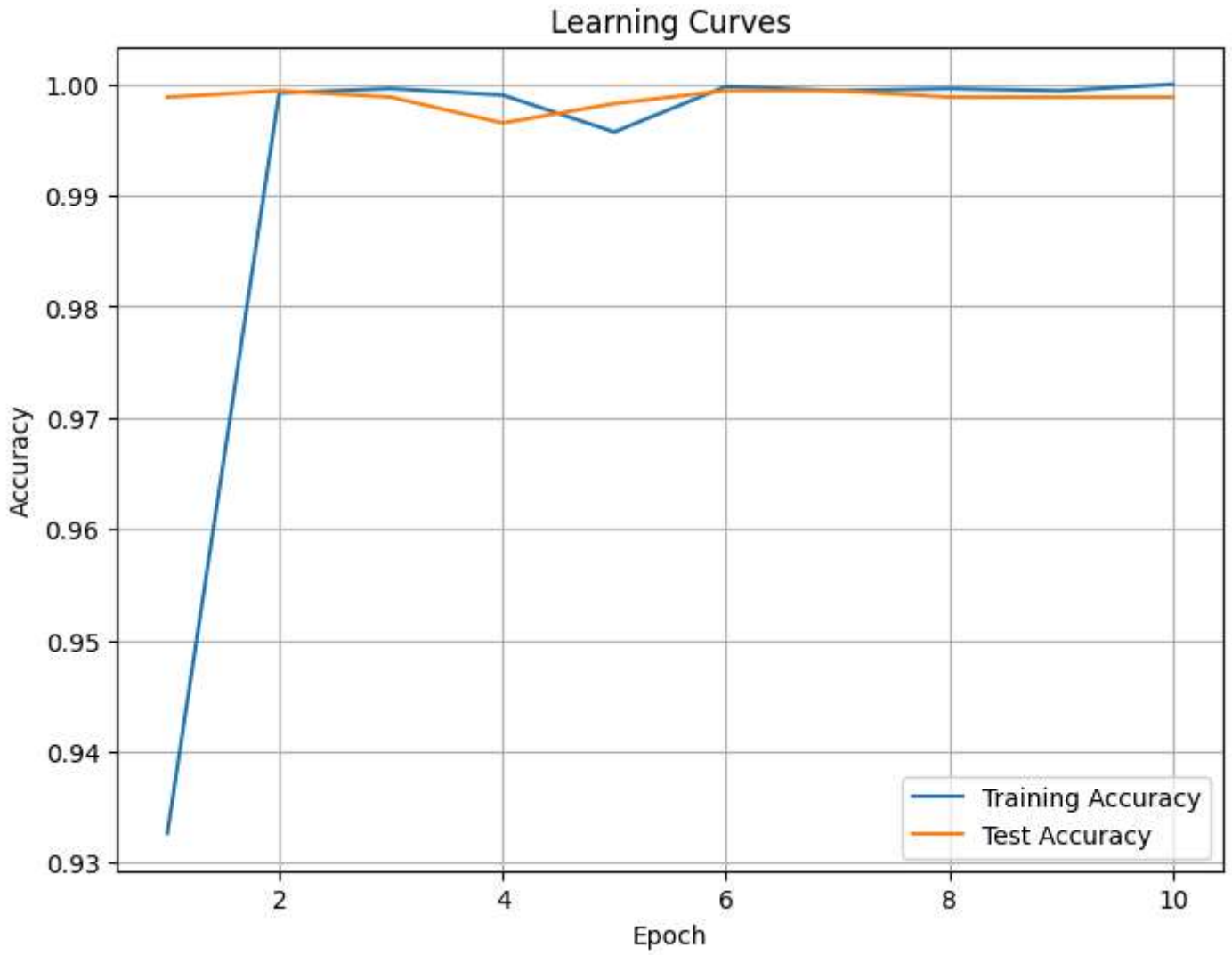
In [ ]:
```
# https://pyimagesearch.com/2021/10/11/pytorch-transfer-learning-and-image-classification/
# https://pytorch.org/vision/stable/models.html
```

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import models, transforms
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Define the data transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Create instances of the custom dataset
train_dataset = FruitDataset('Data/Training', transform=transform)
test_dataset = FruitDataset('Data/Test', transform=transform)

# Create data loaders
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Load the pre-trained ResNet18 model
model = models.resnet18(pretrained=True)

# Freeze all layers
for param in model.parameters():
    param.requires_grad = False


# Replace the last fully connected layer
num_features = model.fc.in_features
num_classes = len(train_dataset.classes)
model.fc = nn.Linear(num_features, num_classes)

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

```
# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters())

# Train the model
num_epochs = 10
train_accuracies = []
test_accuracies = []

for epoch in range(num_epochs):
    model.train()
    train_correct = 0
    train_total = 0

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

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

        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

    train_accuracy = train_correct / train_total
    train_accuracies.append(train_accuracy)

    model.eval()
    test_correct = 0
    test_total = 0

    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
```

```
            _, predicted = torch.max(outputs.data, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_accuracy = test_correct / test_total
    test_accuracies.append(test_accuracy)

    print(f"Epoch [{epoch+1}/{num_epochs}], Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}"

# Plot the learning curves
plt.figure(figsize=(8, 6))
plt.plot(range(1, num_epochs+1), train_accuracies, label='Training Accuracy')
plt.plot(range(1, num_epochs+1), test_accuracies, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Learning Curves')
plt.legend()
plt.grid(True)
plt.show()
```

```
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\models\_utils.py:208: UserWarnin
g: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instea
d.
  warnings.warn(
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\models\_utils.py:223: UserWarnin
g: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the fut
ure. The current behavior is equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use `weight
s=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Epoch [1/10], Train Accuracy: 0.9327, Test Accuracy: 0.9988
Epoch [2/10], Train Accuracy: 0.9992, Test Accuracy: 0.9994
Epoch [3/10], Train Accuracy: 0.9996, Test Accuracy: 0.9988
Epoch [4/10], Train Accuracy: 0.9990, Test Accuracy: 0.9965
Epoch [5/10], Train Accuracy: 0.9957, Test Accuracy: 0.9983
Epoch [6/10], Train Accuracy: 0.9998, Test Accuracy: 0.9994
Epoch [7/10], Train Accuracy: 0.9994, Test Accuracy: 0.9994
Epoch [8/10], Train Accuracy: 0.9996, Test Accuracy: 0.9988
Epoch [9/10], Train Accuracy: 0.9994, Test Accuracy: 0.9988
Epoch [10/10], Train Accuracy: 1.0000, Test Accuracy: 0.9988
```



(c).Feature Extraction

In [ ]:
```
# Feature Extraction https://pytorch.org/docs/stable/generated/torch.nn.modules.module.register_module_forward_hook.h

# Extract features from the last convolutional layer
features = {}
def hook(module, input, output):
    features['last_conv'] = output.detach()
```

```
model.layer4[-1].register_forward_hook(hook)
```

Out[ ]: <torch.utils.hooks.RemovableHandle at 0x23507b792d0>

d. Feature Extraction on Training and Testing Datasets

In [ ]:
```
# Feature Extraction https://pytorch.org/docs/stable/generated/torch.nn.modules.module.register_module_forward_hook.h


# Extract features from the training dataset
train_features = []
train_labels = []

model.eval()
with torch.no_grad():
    for images, labels in train_loader:
        images = images.to(device)
        model(images)
        train_features.append(features['last_conv'].view(images.size(0), -1).cpu().numpy())
        train_labels.extend(labels.cpu().numpy())

train_features = np.concatenate(train_features, axis=0)
train_labels = np.array(train_labels)

# Extract features from the test dataset
test_features = []
test_labels = []

model.eval()
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        model(images)
        test_features.append(features['last_conv'].view(images.size(0), -1).cpu().numpy())
        test_labels.extend(labels.cpu().numpy())

test_features = np.concatenate(test_features, axis=0)
test_labels = np.array(test_labels)
```

e. SVM with the best Kernel (rbf) and best C parameter

In [ ]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# Train an SVM model using the extracted features
svm = SVC(kernel='rbf', C=10)  # Use the best kernel and C value from Question 4(e)
svm.fit(train_features, train_labels)

# Evaluate the SVM model
train_predictions = svm.predict(train_features)
train_accuracy = accuracy_score(train_labels, train_predictions)
print(f"SVM Train Accuracy: {train_accuracy:.4f}")

test_predictions = svm.predict(test_features)
test_accuracy = accuracy_score(test_labels, test_predictions)
print(f"SVM Test Accuracy: {test_accuracy:.4f}")
```

```
SVM Train Accuracy: 1.0000
SVM Test Accuracy: 0.9988
```

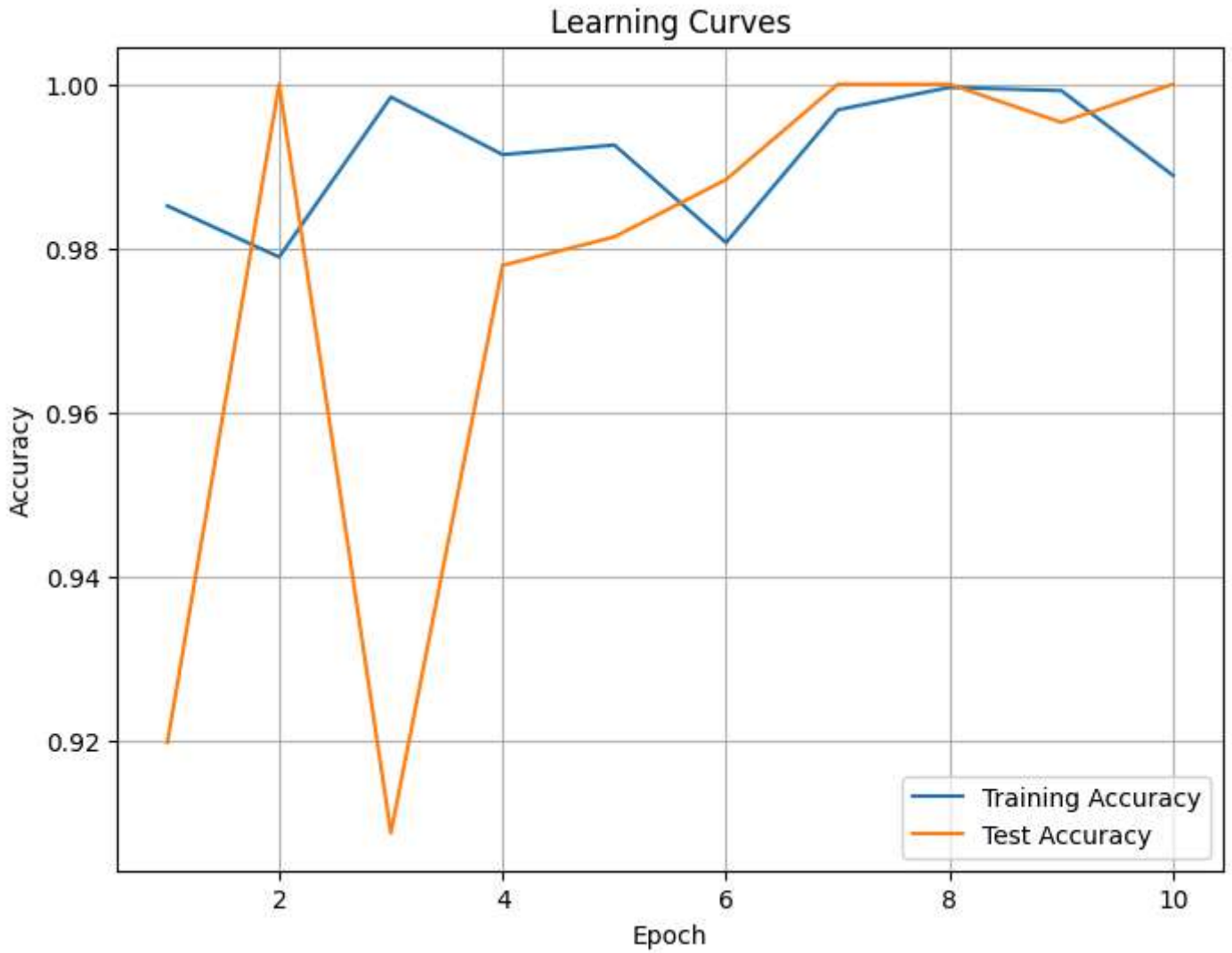
8. Transfer Learning with fine tuning.

In [ ]:
```
# https://pyimagesearch.com/2021/10/11/pytorch-transfer-learning-and-image-classification/
# https://pytorch.org/vision/stable/models.html


import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import models, transforms
import matplotlib.pyplot as plt

# Define the data transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Create instances of the custom dataset
```

```
train_dataset = FruitDataset('Data/Training', transform=transform)
test_dataset = FruitDataset('Data/Test', transform=transform)

# Create data loaders
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Load the pre-trained ResNet18 model
model = models.resnet18(pretrained=True)

# Replace the last fully connected layer
num_features = model.fc.in_features
num_classes = len(train_dataset.classes)
model.fc = nn.Linear(num_features, num_classes)

# Move the model to the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

# Train the model
num_epochs = 10
train_accuracies = []
test_accuracies = []

for epoch in range(num_epochs):
    model.train()
    train_correct = 0
    train_total = 0

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

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

```
        optimizer.step()

        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

    train_accuracy = train_correct / train_total
    train_accuracies.append(train_accuracy)

    model.eval()
    test_correct = 0
    test_total = 0

    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_accuracy = test_correct / test_total
    test_accuracies.append(test_accuracy)

    print(f"Epoch [{epoch+1}/{num_epochs}], Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}'

# Plot the learning curves
plt.figure(figsize=(8, 6))
plt.plot(range(1, num_epochs+1), train_accuracies, label='Training Accuracy')
plt.plot(range(1, num_epochs+1), test_accuracies, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Learning Curves')
plt.legend()
plt.grid(True)
plt.show()
```

```
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-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(
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-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=ResNet18_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Epoch [1/10], Train Accuracy: 0.9852, Test Accuracy: 0.9197
Epoch [2/10], Train Accuracy: 0.9789, Test Accuracy: 1.0000
Epoch [3/10], Train Accuracy: 0.9984, Test Accuracy: 0.9087
Epoch [4/10], Train Accuracy: 0.9914, Test Accuracy: 0.9779
Epoch [5/10], Train Accuracy: 0.9926, Test Accuracy: 0.9814
Epoch [6/10], Train Accuracy: 0.9807, Test Accuracy: 0.9884
Epoch [7/10], Train Accuracy: 0.9969, Test Accuracy: 1.0000
Epoch [8/10], Train Accuracy: 0.9996, Test Accuracy: 1.0000
Epoch [9/10], Train Accuracy: 0.9992, Test Accuracy: 0.9953
Epoch [10/10], Train Accuracy: 0.9889, Test Accuracy: 1.0000
```



9.

(a)For 6(a), 7(c), 8(b) The test accuracies for each method before overfitting are 98.49, 99.94 and 1.000

(b) The transfer learnt model with fine tuning gave the best accuracy of 100%.

(c) There are no performance results that surprised me. Since all the models gave around 99% test accuracy after few epochs.

References

- Data
- SIFT tutorial
- K-Means clustering
- PCA
- SVM
- Custom Dataset
- Model creation
- PyTorch transfer learning and image classification
- PyTorch models
- Feature Extraction