In [1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import ImageFolder
from torchvision import transforms
from torch.utils.data import DataLoader
from torchmetrics import Recall, Precision
import matplotlib.pyplot as plt
```

In [2]:
```
# Data augmentation at training time
# Define transforms
train_transforms = transforms.Compose([
    # Add horizontal flip and rotation
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(45),
    transforms.ToTensor(),
    transforms.Resize((128,128))])

# Create dataset using ImageFolder
dataset_train = ImageFolder(
    "C:/Users/Sanayak/Desktop/Exam DB/Exam DB/Train multi class fruits detection",
    transform = train_transforms,)
# Data augmentation at test time
test_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((128,128))])
dataset_test = ImageFolder(
    "C:/Users/Sanayak/Desktop/Exam DB/Exam DB/Test multi fruits detection",
    transform = test_transforms,)

dataloader_train = DataLoader(dataset_train, shuffle = True, batch_size = 1)
dataloader_test = DataLoader(dataset_test, shuffle = False, batch_size = 1)

image, label = next(iter(dataloader_train))
print(image.shape)
```

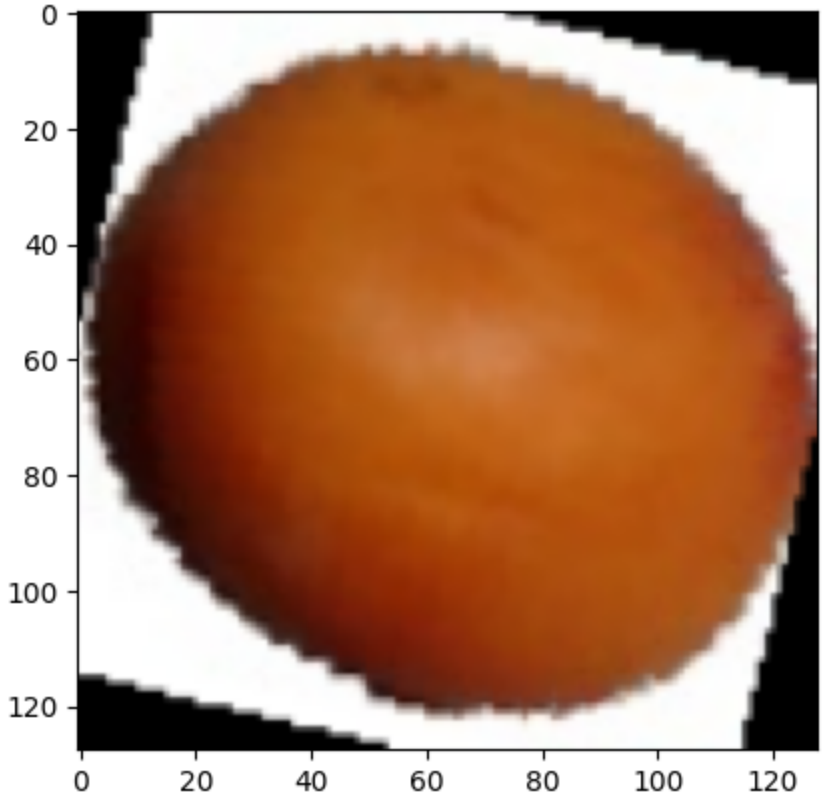
```
torch.Size([1, 3, 128, 128])
```

In [3]:
```
# Reshape the image tensor
image = image.squeeze().permute(1,2,0)
```

```
print(image.shape)
```

```
torch.Size([128, 128, 3])
```

In [4]:
```
# Display the image
plt.imshow(image)
plt.show()
```



In [5]:
```
# Define a CNN Model
class FruitClassifierCNN(nn.Module):
    def __init__(self,num_classes):
        super().__init__()
        self.feature_extraction = nn.Sequential(
            nn.Conv2d(3,32, kernel_size=3,padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(32,64,kernel_size=3, padding=1),
```

```
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Flatten(),
        )
        # Define classifier
        self.classifier = nn.Linear(64*64*16, num_classes)

    def forward(self,x):
        # Pass input through feature extractor classifier
        x = self.feature_extraction(x)
        x = self.classifier(x)
        return x
```

In [6]:
```
# Initialize model,loss and optimizer
model = FruitClassifierCNN(num_classes=4)
# Define the loss function
criterion = nn.CrossEntropyLoss()
# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# List to store losses
train_losses = []
test_losses = []

#training and testing loop with loss tracking
num_epochs = 20

for epoch in range(num_epochs):
    # Traing phase
    model.train()
    epoch_train_loss = 0.0
    for images, labels in dataloader_train:
        optimizer.zero_grad()
        outputs = model(images) # Forward pass
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        epoch_train_loss += loss.item()
    # Average training loss for the epoch
    avg_train_loss = epoch_train_loss / len(dataloader_train)
    train_losses.append(avg_train_loss)
```

In [7]:
```
# Evaluation Loop
metric_precision = Precision(task="multiclass",
                             num_classes=4,
                             average=None)
metric_recall = Recall(task="multiclass",
                       num_classes=4,
                       average=None)
model.eval()
epoch_test_loss = 0.0
with torch.no_grad():
    for images, labels in dataloader_test:
        outputs = model(images)
        _, preds = torch.max(outputs, 1)
        metric_precision(preds,labels)
        metric_recall(preds,labels)
        loss = criterion(outputs, labels)
        epoch_test_loss += loss.item()


    # Average test loss for the epoch
    avg_test_loss = epoch_test_loss / len(dataloader_test)
    test_losses.append(avg_test_loss)

precision = metric_precision.compute()
recall = metric_recall.compute()

print(f"Precision: {precision}")
print(f"Recall: {recall}")
# Prnt Epoch summary
print(f"Epoch {epoch+1}/{num_epochs}")
print(f"Epoch {epoch+1}, Loss:{avg_train_loss: .4f}")
print(f"Epoch {epoch+1}, Loss:{avg_test_loss: .4f}")
```

```
Precision: tensor([1., 1., 1., 1.])
Recall: tensor([1., 1., 1., 1.])
Epoch 20/20
Epoch 20, Loss: 0.0000
Epoch 20, Loss: 0.0000
```

In [8]:
```
model_path = "fruits_classifier_model.pth"
torch.save(model.state_dict(), model_path)
```

```
print(f"Model saved to {model_path}")
```

```
Model saved to fruits_classifier_model.pth
```

TRAINED AND PREPARED BY VICTOR ITINAH INIOBONG