Changes	Time	Difficulty
resize images to be 128 by 128 and convert it to tensor	10 mins	1
Create dataloaders and setting the batch size to 32. Shuffle training data but left testing data unshuffle	10 mins	1
Initialise the model with 2 conv layers and 1 pooling in between them. After running the training with 15 epocs , the loss is 0.0035 but the accuracy is 51.9%	30 mins	3
Try changing the number of layer but was met with error for number of output and input were different	1 hour	6
use 3 conv layer and pooling after every conv layer but accuracy was lower. I think is because it has too many pooling layer	30 mins	3
Try different epocs value but loss reduce less at around 15 epocs	20 mins	2
Change the learning rate to 0.0005 instead of 0.001 but similar result	5 mins	1
Add another conv layer to make it to 3 layer and do pooling at the end of conv layer instead of in between and increase the accuracy to 52.2%	30 mins	3
The loss is very low , it could be the neural network was memorizing the training set so include drop out to drop a neuron for better generalisation. The loss is now 0.3859 but accuracy increase to 55%	45 mins	5

In [2]:
```
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
from torchvision.datasets import ImageFolder
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
import torch.nn as nn
import torch.nn.functional as F
import os
os.environ["CUDA_LAUNCH_BLOCKING"] = "1"
```

In [174…
```
transform = transforms.Compose([
    transforms.Resize((128, 128)),  # Resize images
    transforms.ToTensor(),  # Convert to tensor
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize
])
```

In [176…

```
# Load training dataset
train_dataset = ImageFolder(root="fruit_data/train", transform=transform)

# Load validation dataset
val_dataset = ImageFolder(root="fruit_data/test", transform=transform)

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

In [178…

```
print(train_dataset.classes)  # Check labels
```

```
['Apple', 'Banana', 'avocado', 'cherry', 'kiwi', 'mango', 'orange', 'pinenapple',
'strawberries', 'watermelon']
```

class FruitCNN(nn.Module): def __init__(self, num_classes=10): # 10 classes of label super(FruitCNN, self).__init__() self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1) self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0) self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1) self.fc1 = nn.Linear(64 * 32 * 32, 128) self.fc2 = nn.Linear(128, num_classes) def forward(self, x): x = self.pool(F.relu(self.conv1(x))) x = self.pool(F.relu(self.conv2(x))) x = x.view(x.size(0), -1) # Flatten x = F.relu(self.fc1(x)) x = self.fc2(x) return x

In [180…

```
class FruitCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(FruitCNN, self).__init__()

        # Convolutional Layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)

        # Pooling Layer
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # Fully Connected Layers
        self.fc1 = nn.Sequential(nn.Linear(128 * 16 * 16, 256), nn.ReLU(), nn.Dr
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, num_classes)  # Final output layer

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # (B, 32, 64, 64) → (B, 32, 32, 32
        x = self.pool(F.relu(self.conv2(x)))  # (B, 64, 32, 32) → (B, 64, 16, 16
        x = self.pool(F.relu(self.conv3(x)))  # (B, 128, 16, 16) → (B, 128, 8, 8

        x = x.view(x.size(0), -1)  # Flatten (B, 128*8*8)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return x
```

In [182…

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

model = FruitCNN(num_classes=10).to(device)
```

```
Using device: cuda
```

In [184…

```
criterion = nn.CrossEntropyLoss()  # For classification tasks
optimizer = torch.optim.Adam(model.parameters(), lr=0.0005)
```

In [186…

```
num_epochs = 15

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

    # Show progress bar
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}", leav

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

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

        running_loss += loss.item()

        progress_bar.set_postfix(loss=f"{loss.item():.4f}")

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader

print("Training complete.")
```

```
Epoch [1/15], Loss: 1.9787
Epoch [2/15], Loss: 1.5860
Epoch [3/15], Loss: 1.4432
Epoch [4/15], Loss: 1.3581
Epoch [5/15], Loss: 1.1981
Epoch [6/15], Loss: 1.1433
Epoch [7/15], Loss: 1.0319
Epoch [8/15], Loss: 0.9079
Epoch [9/15], Loss: 0.8144
Epoch [10/15], Loss: 0.7014
Epoch [11/15], Loss: 0.5907
Epoch [12/15], Loss: 0.4919
Epoch [13/15], Loss: 0.4201
Epoch [14/15], Loss: 0.3389
```

```
Epoch [15/15], Loss: 0.2640
Training complete.
```

In [187…

```
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Validation Accuracy: {100 * correct / total:.2f}%")
```

```
Validation Accuracy: 54.44%
```

In [188…

```
correct = torch.zeros(10).to(device)
total = torch.zeros(10).to(device)

model.eval()
with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        preds = torch.argmax(outputs, dim=1)

        for label in range(10):
            correct[label] += (preds[labels == label] == label).sum()
            total[label] += (labels == label).sum()

accuracy_per_class = correct / total
print(accuracy_per_class.cpu().numpy())  # See per-class accuracy
```

```
[0.76404494 0.10476191 0.04716981 0.72380954 0.6952381  0.31428573
 0.82474226 0.83809525 0.61165047 0.5809524 ]
```

In [ ]: