In [163…
```
#Load libraries
import os
#I Have problems with packages try running without this command
os.environ['KMP_DUPLICATE_LIB_OK']='True'
import numpy as np
import torch
import glob
import torch.nn as nn
import torch.nn.functional as F
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
from torch.optim import Adam
from torch.autograd import Variable
import torchvision
import pathlib
import sys
import platform
import time
# from google.colab import drive #used to access files
# drive.mount('/content/gdrive')
```

In [164…
```
#checking for device
has_gpu = torch.cuda.is_available()
has_mps = torch.backends.mps.is_built()
device = "mps" if torch.backends.mps.is_built() \
    else "gpu" if torch.cuda.is_available() else "cpu"

print(f"Python Platform: {platform.platform()}")
print(f"PyTorch Version: {torch.__version__}")
print()
print(f"Python {sys.version}")
print(f"Pandas {pd.__version__}")
print(f"Scikit-Learn {sk.__version__}")
print("GPU is", "available" if has_gpu else "NOT AVAILABLE")
print("MPS (Apple Metal) is", "AVAILABLE" if has_mps else "NOT AVAILABLE")
print(f"Target device is {device}")
```

```
Python Platform: macOS-13.2-arm64-arm-64bit
PyTorch Version: 2.1.0.dev20230617

Python 3.9.16 | packaged by conda-forge | (main, Feb  1 2023, 21:38:11)
[Clang 14.0.6 ]
Pandas 2.0.2
Scikit-Learn 1.2.2
GPU is NOT AVAILABLE
MPS (Apple Metal) is AVAILABLE
Target device is mps
```

In [165…
```
print(device)
```

```
mps
```

In [166…
```
#Transforms
transformer=transforms.Compose([
    transforms.Resize((150,150)), #recommended size with 3x3 filters bigger
    transforms.RandomHorizontalFlip(), #chance of image being flipped is 0.5
    transforms.ToTensor(),  #0-255 to 0-1, numpy to tensors (Pytorch uses te
    transforms.Normalize([0.5,0.5,0.5],
                         [0.5,0.5,0.5]) # 0-1 to [-1,1] , formula (x-mean)/s
])
```

In [167…
```
#Dataloader

#Path for training and testing directory
train_path='/Users/hadi/Desktop/Concordia/Comp 6721/AIproject/fruits/trainir
test_path='/Users/hadi/Desktop/Concordia/Comp 6721/AIproject/fruits/testing/

train_loader=DataLoader(
    torchvision.datasets.ImageFolder(train_path,transform=transformer),
    batch_size=64, shuffle=True
)
#by deault it was 64 but ta said use 32 (Personally I think 64 is better bec

test_loader=DataLoader(
    torchvision.datasets.ImageFolder(test_path,transform=transformer),
    batch_size=32, shuffle=True
)
```

In [168…
```
#Returning classes
root=pathlib.Path(train_path)
classes = []
for j in root.iterdir():
    if j.name.split('/')[-1] != ".DS_Store":
        classes.append(j.name.split('/')[-1])
```

In [169…
```
print(classes)
```

```
['Banana_Training', 'Kiwi_Training', 'Mango_Training', 'Orange_Training', 'Pl
um_Training', 'Apple_Training']
```

In [170…
```
#Building CNN netwrok



class ConvNet(nn.Module):
    def __init__(self,num_classes=6):
        super(ConvNet,self).__init__()
```

```
        #Output size after convolution filter
        #((w-f+2P)/s) +1 w: width (32) - f: kernel size (filter) - p:padding

        #Input shape= (32,3,32,32) 32: batch size - 3: rgb channels - 32x32:

        self.conv1=nn.Conv2d(in_channels=3,out_channels=12,kernel_size=3,str
        #Shape= (256,12,150,150)
        self.bn1=nn.BatchNorm2d(num_features=12)
        #Shape= (256,12,150,150)
        self.relu1=nn.ReLU()
        #Shape= (256,12,150,150)

        self.pool=nn.MaxPool2d(kernel_size=2)
        #Reduce the image size be factor 2
        #Shape= (256,12,75,75)


        self.conv2=nn.Conv2d(in_channels=12,out_channels=20,kernel_size=3,st
        #Shape= (256,20,75,75)
        self.relu2=nn.ReLU()
        #Shape= (256,20,75,75)



        self.conv3=nn.Conv2d(in_channels=20,out_channels=32,kernel_size=3,st
        #Shape= (256,32,75,75)
        self.bn3=nn.BatchNorm2d(num_features=32)
        #Shape= (256,32,75,75)
        self.relu3=nn.ReLU()
        #Shape= (256,32,75,75)


        self.fc=nn.Linear(in_features=75 * 75 * 32,out_features=num_classes)



        #Feed forwad function

    def forward(self,input):
        output=self.conv1(input)
        output=self.bn1(output)
        output=self.relu1(output)

        output=self.pool(output)

        output=self.conv2(output)
        output=self.relu2(output)

        output=self.conv3(output)
```

```
        output=self.bn3(output)
        output=self.relu3(output)


            #Above output will be in matrix form, with shape (256,32,75,75)

        output=output.view(-1,32*75*75)


        output=self.fc(output)

        return output
```

In [171…
```
model=ConvNet(num_classes=6).to(device)
```

In [172…
```
#Optmizer and loss function
optimizer=Adam(model.parameters(),lr=0.001,weight_decay=0.0005) #0.0005 #0.0
loss_function=nn.CrossEntropyLoss()
```

In [173…
```
num_epochs=10
```

In [174…
```
#calculating the size of training and testing images
train_count=len(glob.glob(train_path+'/**/*.png'))
test_count=len(glob.glob(test_path+'/**/*.png'))
print(train_count,test_count)
```

```
18341 4583
```

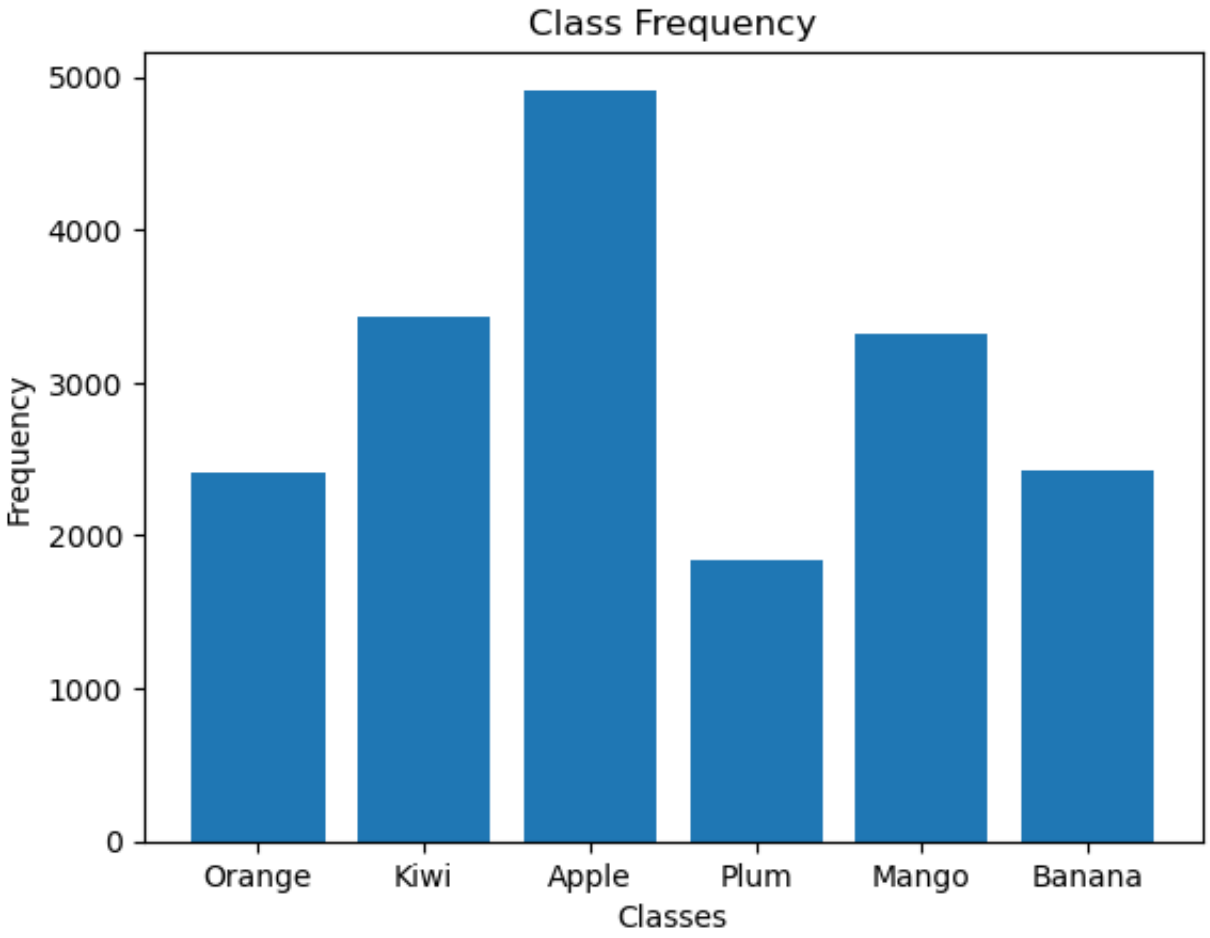
In [176…
```
#Model training and saving best model

best_accuracy=0.0
metrics = np.empty((0, 3))  # Initialize an empty matrix

for epoch in range(num_epochs):

    start_time = time.time()  # Start time of the epoch

    #Evaluation and training on training dataset
    model.train()
    train_accuracy=0.0
    train_loss=0.0

    for i, (images,labels) in enumerate(train_loader):
        if torch.backends.mps.is_built():
            images=Variable(images.to(device))
            labels=Variable(labels.to(device))

        optimizer.zero_grad()
```

```
        outputs=model(images)
        loss=loss_function(outputs,labels)
        loss.backward()
        optimizer.step()


        train_loss+= loss.cpu().data*images.size(0)
        _,prediction=torch.max(outputs.data,1)

        train_accuracy+=int(torch.sum(prediction==labels.data))

    train_accuracy=train_accuracy/train_count
    train_loss=train_loss/train_count


    # Evaluation on testing dataset
    model.eval()

    test_accuracy=0.0
    for i, (images,labels) in enumerate(test_loader):
        if torch.backends.mps.is_built():
            images=Variable(images.to(device))
            labels=Variable(labels.to(device))

        outputs=model(images)
        _,prediction=torch.max(outputs.data,1)
        test_accuracy+=int(torch.sum(prediction==labels.data))

    test_accuracy=test_accuracy/test_count

    metric_per_epoch = np.array([train_loss,train_accuracy,test_accuracy])
    metrics = np.vstack((metrics, metric_per_epoch))


    print('Epoch: '+str(epoch)+' Train Loss: '+str(train_loss)+' Train Accur


    #Save the best model
    if test_accuracy>best_accuracy:
        torch.save(model.state_dict(),'best_checkpoint.model')
        best_accuracy=test_accuracy

    # Calculate elapsed time
    end_time = time.time()
    elapsed_time = end_time - start_time

    print(' Elapsed Time: ' + str(elapsed_time) + ' seconds')

    #end of an epoch
```

```
Epoch: 0 Train Loss: tensor(0.2667) Train Accuracy: 0.9736655580393654 Test A
ccuracy: 0.9454505782238708
 Elapsed Time: 93.12236785888672 seconds
Epoch: 1 Train Loss: tensor(0.2140) Train Accuracy: 0.9780273703723897 Test A
ccuracy: 0.8199869081387737
 Elapsed Time: 92.1808078289032 seconds
Epoch: 2 Train Loss: tensor(0.1875) Train Accuracy: 0.9828798865928794 Test A
ccuracy: 0.9389046476107353
 Elapsed Time: 92.26931309700012 seconds
Epoch: 3 Train Loss: tensor(0.1135) Train Accuracy: 0.9900768769423696 Test A
ccuracy: 0.9186122627100153
 Elapsed Time: 93.57984685897827 seconds
Epoch: 4 Train Loss: tensor(0.1279) Train Accuracy: 0.9890409465132762 Test A
ccuracy: 0.9567968579533057
 Elapsed Time: 93.66771793365479 seconds
Epoch: 5 Train Loss: tensor(0.2347) Train Accuracy: 0.9842974756011122 Test A
ccuracy: 0.9264673794457778
 Elapsed Time: 92.78005313873291 seconds
Epoch: 6 Train Loss: tensor(0.2256) Train Accuracy: 0.9845155662177635 Test A
ccuracy: 0.9498145319659611
 Elapsed Time: 92.7351438999176 seconds
Epoch: 7 Train Loss: tensor(0.1239) Train Accuracy: 0.9898587863257183 Test A
ccuracy: 0.967924939995636
 Elapsed Time: 92.43332409858704 seconds
Epoch: 8 Train Loss: tensor(0.0788) Train Accuracy: 0.9933482361921379 Test A
ccuracy: 0.9447959851625573
 Elapsed Time: 95.8816499710083 seconds
Epoch: 9 Train Loss: tensor(0.0700) Train Accuracy: 0.9945477345837196 Test A
ccuracy: 0.915121099716343
 Elapsed Time: 92.71082425117493 seconds
```

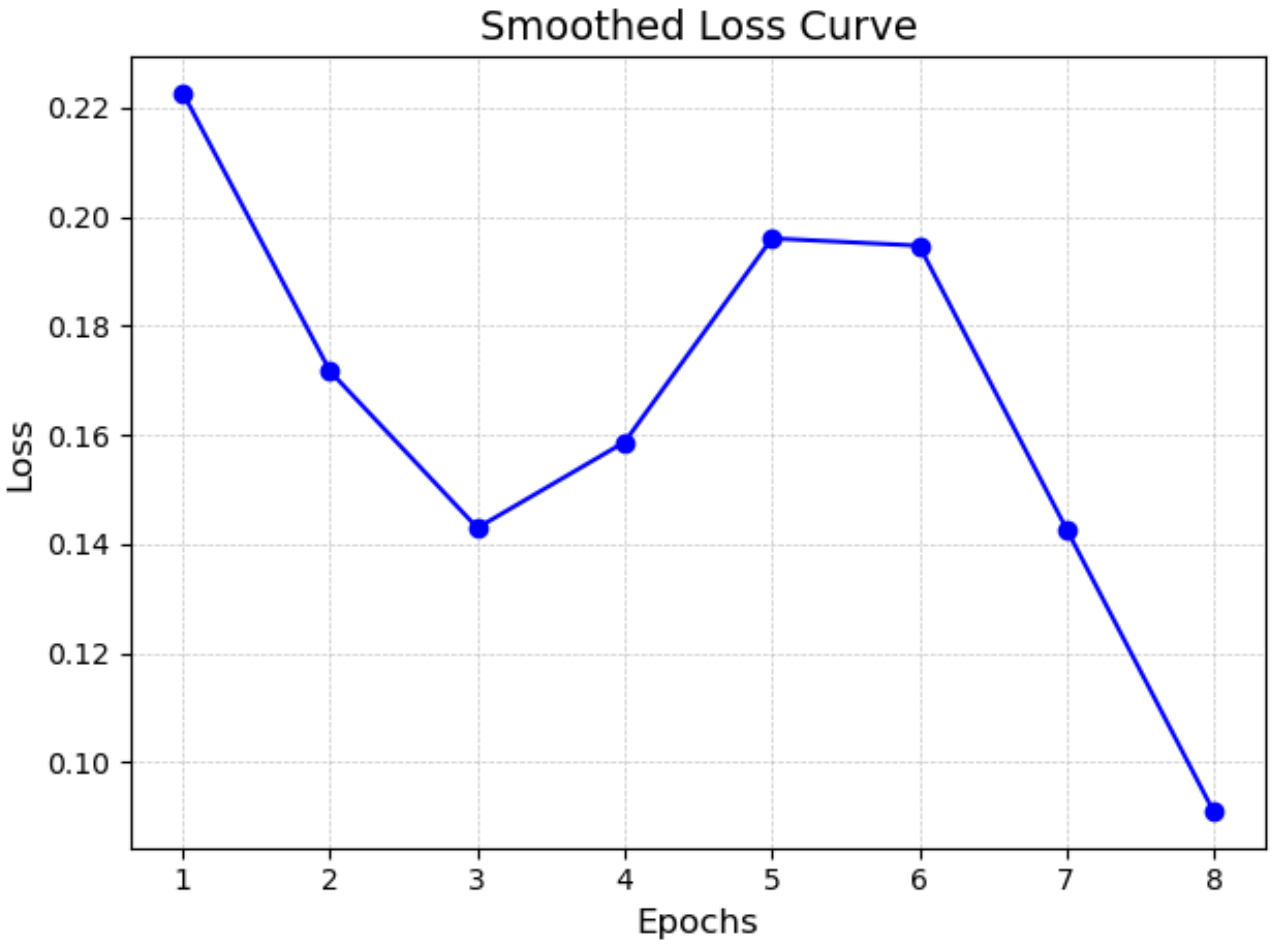
In [177…

```
import matplotlib.pyplot as plt

train_count_orange=len(glob.glob(train_path+'/Orange_Training/*.png'))
train_count_kiwi=len(glob.glob(train_path+'/Kiwi_Training/*.png'))
train_count_apple=len(glob.glob(train_path+'/Apple_Training/*.png'))
train_count_plum=len(glob.glob(train_path+'/Plum_Training/*.png'))
train_count_mango=len(glob.glob(train_path+'/Mango_Training/*.png'))
train_count_banana=len(glob.glob(train_path+'/Banana_Training/*.png'))

# Count the frequency of each label
label_counts = {"Orange":train_count_orange,"Kiwi":train_count_kiwi,"Apple":

# Extract the labels and frequencies
x = list(label_counts.keys())
y = list(label_counts.values())

# Plotting the graph
plt.bar(x, y)
plt.xlabel('Classes')
plt.ylabel('Frequency')
plt.title('Class Frequency')
plt.show()
```



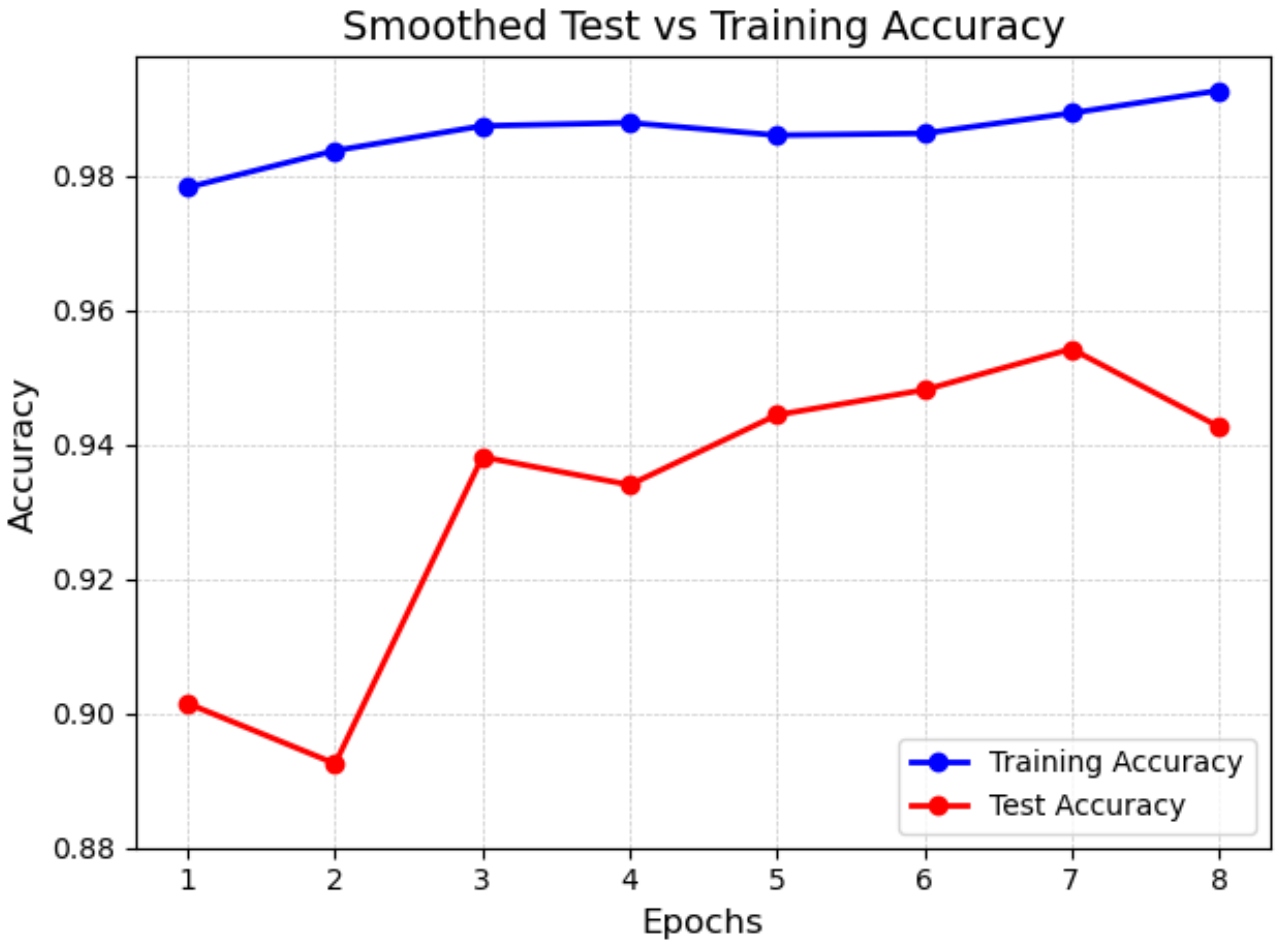
In [178…

```
#Plot for the loss curve

loss_values = []

for m in metrics:
  loss_values.append(m[0])

# Apply moving average to smooth out the loss values
window_size = 3  # Adjust the window size as desired
loss_values_smooth = np.convolve(loss_values, np.ones(window_size)/window_si

# Generate x-axis values for epochs
epochs = range(1, len(loss_values_smooth) + 1)

# Plotting the smoothed loss curve
plt.plot(epochs, loss_values_smooth, 'b-o')

# Customize plot appearance
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.title('Smoothed Loss Curve', fontsize=14)
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()

# Show the plot
plt.show()
```


Smoothed Loss Curve
Loss
0.22
0.20
0.18
0.16
0.14
0.12
0.10
1
2
3
4
5
6
7
8
Epochs

In [179…

```
#plot for the test vs training accuracy to check overfiiting

training_accuracy = []
test_accuracy = []

for m in metrics:
  training_accuracy.append(m[1])
  test_accuracy.append(m[2])

# Apply moving average to smooth out the accuracy values
window_size = 3  # Adjust the window size as desired
training_accuracy_smooth = np.convolve(training_accuracy, np.ones(window_siz
test_accuracy_smooth = np.convolve(test_accuracy, np.ones(window_size)/windo

# Generate x-axis values for epochs
epochs = range(1, len(training_accuracy_smooth) + 1)

# Plotting the smoothed test vs training accuracy
plt.plot(epochs, training_accuracy_smooth, 'b-o', label='Training Accuracy',
plt.plot(epochs, test_accuracy_smooth, 'r-o', label='Test Accuracy', linewid

# Customize plot appearance
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Smoothed Test vs Training Accuracy', fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
plt.xticks(fontsize=10)
plt.yticks(np.arange(0.88, 1, 0.02),fontsize=10)
plt.tight_layout()

# Show the plot
plt.show()
```


Smoothed Test vs Training Accuracy
Accuracy
Epochs
0.98
0.96
0.94
0.92
0.90
0.88
1
2
3
4
5
6
7
8
Training Accuracy
Test Accuracy

In [ ]: