#Import the necessary libraries

import torch

import torchvision

import torchvision.transforms as transforms

import torchvision.datasets as datasets

import matplotlib.pyplot as plt

import numpy as np

from PIL import Image

from torch.autograd import Variable

import torchvision.thisModels as thisModels

from torch import nn, optim

#Load the data

data\_directory = 'flowers'

train\_directory = data\_directory + '/train'

valid\_directory = data\_directory + '/valid'

test\_directory = data\_directory + '/test'

#Define  the image transformation characteristics

train\_transformation = transforms.Compose([transforms.Resize(255),

                                transforms.CenterCrop(224),

                                transforms.RandomHorizontalFlip(),

                                transforms.ToTensor(),

                                transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])

valid\_transformation = transforms.Compose([transforms.Resize(255),

                                transforms.CenterCrop(224),

                                transforms.ToTensor(),

                                transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])

test\_transformation = transforms.Compose([transforms.Resize(255),

                                transforms.CenterCrop(224),

                                transforms.ToTensor(),

                                transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])

train\_dataset = datasets.ImageFolder(train\_directory, transform=train\_transformation)

valid\_dataset = datasets.ImageFolder(valid\_directory, transform=train\_transformation)

test\_dataset = datasets.ImageFolder(test\_directory, transform=train\_transformation)

trainset = torch.utils.data.DataLoader(train\_dataset, batch\_size=64, shuffle=True)

validset = torch.utils.data.DataLoader(valid\_dataset, batch\_size=64)

testset = torch.utils.data.DataLoader(test\_dataset, batch\_size=64)

#The mappings

import json

with open('cat\_to\_name.json', 'r') as f:

    cat\_to\_name = json.load(f)

#Testing the loads

images, labels = next(iter(testset))

print(len(images[0,2]))

plt.imshow(images[0,0])

# Training the classifier

import torchvision.thisModels as thisModels

from torch import nn, optim

thisDevice = torch.thisDevice("cuda" if torch.cuda.is\_available() else "cpu")

thisModel = thisModels.vgg16(pretrained=True)

for param in thisModel.parameters():

    param.requires\_grad = False

from collections import OrderedDict

thisModel.classifier = nn.Sequential(OrderedDict([

                          ('fc1', nn.Linear(25088, 2048)),

                          ('relu', nn.ReLU()),

                          ('fc2', nn.Linear(2048, 256)),

                          ('relu', nn.ReLU()),

                          ('fc3', nn.Linear(256, 102)),

                          ('output', nn.LogSoftmax(dim=1))

                          ]))

print(thisModel)

thisModel = thisModel.to('cuda')

criterion = nn.NLLLoss()

optimizer = optim.Adam(thisModel.classifier.parameters(), lr=0.001)

#Classifier training

epochs = 3

steps = 0

running\_loss = 0

print\_every = 5

for epoch in range(epochs):

    for inputs, labels in trainset:

        steps += 1

        inputs, labels = inputs.to('cuda'), labels.to('cuda')

        optimizer.zero\_grad()

        logps = thisModel.forward(inputs)

        loss = criterion(logps, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

        if steps % print\_every == 0:

            valid\_loss = 0

            accuracy = 0

            thisModel.eval()

            with torch.no\_grad():

                for inputs, labels in validset:

                    inputs, labels = inputs.to('cuda'), labels.to('cuda')

                    logps = thisModel.forward(inputs)

                    batch\_loss = criterion(logps, labels)

                    valid\_loss += batch\_loss.item()

                    ps = torch.exp(logps)

                    top\_p, top\_class = ps.topk(1, dim=1)

                    equals = top\_class == labels.view(\*top\_class.shape)

                    accuracy += torch.mean(equals.type(torch.FloatTensor)).item()

            print(f"Epoch {epoch+1}/{epochs}.. "

                  f"Loss: {running\_loss/print\_every:.3f}.. "

                  f"Validation Loss: {valid\_loss/len(validset):.3f}.. "

                  f"Accuracy: {accuracy/len(validset):.3f}")

            running\_loss = 0

            thisModel.train()

#Network testing

test\_loss = 0

accuracy = 0

thisModel.to('cuda')

with torch.no\_grad():

    for inputs, labels in testset:

        inputs, labels = inputs.to('cuda'), labels.to('cuda')

        logps = thisModel.forward(inputs)

        batch\_loss = criterion(logps, labels)

        test\_loss += batch\_loss.item()

        ps = torch.exp(logps)

        top\_p, top\_class = ps.topk(1, dim=1)

        equals = top\_class == labels.view(\*top\_class.shape)

        accuracy += torch.mean(equals.type(torch.FloatTensor)).item()

print(f"Test accuracy: {accuracy/len(testset):.3f}")

#Image processing

def process\_image(image):

    ''' Scales, crops, and normalizes a PIL image for a PyTorch thisModel,

        returns an Numpy array

    '''

    thisImg\_pil = Image.open(image)

    img\_transforms = transforms.Compose([

        transforms.Resize(256),

        transforms.CenterCrop(224),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])

    ])

    image = img\_transforms(thisImg\_pil)

    return image

#

def imshow(image, ax=None, title=None):

    """Imshow for Tensor."""

    if ax is None:

        fig, ax = plt.subplots()

    image = image.numpy().transpose((1, 2, 0))

    mean = np.array([0.485, 0.456, 0.406])

    std = np.array([0.229, 0.224, 0.225])

    image = std \* image + mean

    image = np.clip(image, 0, 1)

    ax.imshow(image)

    return ax

imshow(process\_image("flowers/test/1/image\_06764.jpg"))

def predict(image\_path, thisModel, topk=5):

    thisModel.to('cuda')

    thisModel.eval()

    img = process\_image(image\_path)

    img = img.numpy()

    img = torch.from\_numpy(np.array([img])).float()

    with torch.no\_grad():

        output = thisModel.forward(img.cuda())

    probability = torch.exp(output).data

    return probability.topk(topk)

img = "flowers/test/10/image\_07090.jpg"

probability, classes = predict(img, thisModel)

print (probability)

print (classes)

#Then predict

plt.rcParams["figure.figsize"] = (10,10)

plt.subplot(211)

index = 1

path = test\_directory + '/1/image\_06743.jpg'

probabilities = predict(path, thisModel)

image = process\_image(path)

axs = imshow(image, ax = plt)

axs.axis('off')

axs.title(cat\_to\_name[str(index)])

axs.show()

a = np.array(probabilities[0][0])

b = [cat\_to\_name[str(index+1)] for index in np.array(probabilities[1][0])]

N=float(len(b))

fig,ax = plt.subplots(figsize=(10,5))

width = 0.5

tickLocations = np.arange(N)

ax.bar(tickLocations, a, width, linewidth=4.0, align = 'center')

ax.set\_xticks(ticks = tickLocations)

ax.set\_xticklabels(b)

ax.set\_xlim(min(tickLocations)-0.6,max(tickLocations)+0.6)

ax.set\_yticks([0.2,0.4,0.6,0.8,1,1.2])

ax.set\_ylim((0,1))

ax.yaxis.grid(True)

plt.show()