**DreamBooth Code**

import torch

import torchvision.transforms as transforms

import torchvision.models as models

import numpy as np

from PIL import Image

import os

import shutil

def load\_images\_from\_folder(folder\_path):

images = []

for filename in os.listdir(folder\_path):

image\_path = os.path.join(folder\_path, filename)

if os.path.isfile(image\_path):

image = Image.open(image\_path).convert('RGB')

images.append((image, filename))

return images

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

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

])

class VGGNet(torch.nn.Module):

def \_\_init\_\_(self):

super(VGGNet, self).\_\_init\_\_()

self.vgg = models.vgg19(pretrained=False)

state\_dict = torch.hub.load\_state\_dict\_from\_url('https://download.pytorch.org/models/vgg19-dcbb9e9d.pth', progress=True, check\_hash=True)

self.vgg.load\_state\_dict(state\_dict)

self.vgg = self.vgg.features[:35]

def forward(self, x):

return self.vgg(x)

def apply\_dream\_booth(image\_tensor, vgg\_net, iterations=20, lr=0.01):

input\_image = torch.autograd.Variable(image\_tensor, requires\_grad=True)

optimizer = torch.optim.Adam([input\_image], lr=lr)

for i in range(iterations):

optimizer.zero\_grad()

features = vgg\_net(input\_image)

loss = torch.mean(features)

loss.backward()

optimizer.step()

output\_image = input\_image.squeeze(0).permute(1, 2, 0).detach().numpy()

output\_image = np.clip(output\_image, 0, 1)

return output\_image

input\_folder = r"C:\Users\Ananya\Desktop\drought"

**images = load\_images\_from\_folder(input\_folder)**

vgg\_net = VGGNet()

output\_folder = r"C:\Users\Ananya\Desktop\dreambooth"

os.makedirs(output\_folder, exist\_ok=True)

for image, filename in images:

image\_tensor = transform(image).unsqueeze(0)

dream\_image = apply\_dream\_booth(image\_tensor, vgg\_net)

output\_image\_path = os.path.join(output\_folder, f'{os.path.splitext(filename)[0]}\_dream.jpg')

dream\_image\_pil = Image.fromarray((dream\_image \* 255).astype(np.uint8))

dream\_image\_pil.save(output\_image\_path)

print("Dream Booth process is complete. Enhanced images saved in:", output\_folder)

**ANOTHER ONE FOR DREAM BOOTH**

import torch

import torchvision.transforms as transforms

import torchvision.models as models

import numpy as np

from PIL import Image

import os

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

def load\_images\_from\_folder(folder\_path):

images = []

for filename in os.listdir(folder\_path):

image\_path = os.path.join(folder\_path, filename)

if os.path.isfile(image\_path):

image = Image.open(image\_path).convert('RGB')

images.append((image, filename))

return images

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

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

])

class VGGNet(torch.nn.Module):

def \_init\_(self):

super(VGGNet, self).\_init\_()

self.vgg = models.vgg19(pretrained=False)

state\_dict = torch.hub.load\_state\_dict\_from\_url('https://download.pytorch.org/models/vgg19-dcbb9e9d.pth', progress=True, check\_hash=True)

self.vgg.load\_state\_dict(state\_dict)

self.vgg = self.vgg.features[:35] # Use first 35 layers for feature extraction

def forward(self, x):

return self.vgg(x)

def apply\_dream\_booth(image\_tensors, vgg\_net, iterations=20, lr=0.01):

input\_images = image\_tensors.to(device).requires\_grad\_(True)

optimizer = torch.optim.Adam([input\_images], lr=lr)

for i in range(iterations):

optimizer.zero\_grad()

features = vgg\_net(input\_images)

loss = torch.mean(features)

loss.backward()

optimizer.step()

output\_images = input\_images.squeeze(0).permute(0, 2, 3, 1).detach().cpu().numpy()

output\_images = np.clip(output\_images, 0, 1) # Clip to valid image range

return output\_images

input\_folder = r"C:\Users\MY\Desktop\drought"

images = load\_images\_from\_folder(input\_folder)

vgg\_net = VGGNet().to(device)

batch\_size = 4

num\_batches = len(images) // batch\_size + int(len(images) % batch\_size > 0)

output\_folder = r"C:\Users\MY\Desktop\aug\_drought"

os.makedirs(output\_folder, exist\_ok=True)

for i in range(num\_batches):

batch\_start = i \* batch\_size

batch\_end = min((i + 1) \* batch\_size, len(images))

batch\_images = [transform(image).unsqueeze(0) for image, \_ in images[batch\_start:batch\_end]]

batch\_tensor = torch.cat(batch\_images, dim=0)

processed\_batch = apply\_dream\_booth(batch\_tensor, vgg\_net)

for j, (image, filename) in enumerate(images[batch\_start:batch\_end]):

output\_image\_path = os.path.join(output\_folder, f'{os.path.splitext(filename)[0]}dream{i \* batch\_size + j}.jpg')

dream\_image\_pil = Image.fromarray((processed\_batch[j] \* 255).astype(np.uint8))

dream\_image\_pil.save(output\_image\_path)

print("Dream Booth process is complete. Enhanced images saved in:", output\_folder)

**ANOTHER ONE**

import torch

import torchvision.transforms as transforms

import torchvision.models as models

import numpy as np

from PIL import Image

import os

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

def load\_images\_from\_folder(folder\_path):

images = []

for filename in os.listdir(folder\_path):

image\_path = os.path.join(folder\_path, filename)

if os.path.isfile(image\_path):

image = Image.open(image\_path).convert('RGB')

images.append((image, filename))

return images

augmentation\_transforms = transforms.Compose([

transforms.RandomHorizontalFlip(),

transforms.RandomVerticalFlip(),

transforms.RandomRotation(degrees=30),

transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),

transforms.RandomResizedCrop(224, scale=(0.8, 1.0), ratio=(0.75, 1.333))

])

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

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

])

class VGGNet(torch.nn.Module):

def \_init\_(self):

super(VGGNet, self).\_init\_()

self.vgg = models.vgg19(pretrained=False)

state\_dict = torch.hub.load\_state\_dict\_from\_url('https://download.pytorch.org/models/vgg19-dcbb9e9d.pth', progress=True, check\_hash=True)

self.vgg.load\_state\_dict(state\_dict)

self.vgg = self.vgg.features[:35]

def forward(self, x):

return self.vgg(x)

def apply\_dream\_booth(image\_tensors, vgg\_net, iterations=20, lr=0.01):

input\_images = image\_tensors.to(device).requires\_grad\_(True)

optimizer = torch.optim.Adam([input\_images], lr=lr)

for i in range(iterations):

optimizer.zero\_grad()

features = vgg\_net(input\_images)

loss = torch.mean(features)

loss.backward()

optimizer.step()

output\_images = input\_images.squeeze(0).permute(0, 2, 3, 1).detach().cpu().numpy()

output\_images = np.clip(output\_images, 0, 1)

return output\_images

input\_folder = r"C:\Users\MY\Desktop\drought"

images = load\_images\_from\_folder(input\_folder)

vgg\_net = VGGNet().to(device)

output\_folder = r"C:\Users\MY\Desktop\aug\_drought"

os.makedirs(output\_folder, exist\_ok=True)

for image, filename in images:

augmented\_images = [augmentation\_transforms(image) for \_ in range(5)]

augmented\_tensors = [transform(aug\_img).unsqueeze(0) for aug\_img in augmented\_images]

augmented\_tensors.append(transform(image).unsqueeze(0))

batch\_tensor = torch.cat(augmented\_tensors, dim=0)

processed\_batch = apply\_dream\_booth(batch\_tensor, vgg\_net)

for j, augmented\_image in enumerate(augmented\_images):

output\_image\_path = os.path.join(output\_folder, f'{os.path.splitext(filename)[0]}\_dream\_aug{j}.jpg')

dream\_image\_pil = Image.fromarray((processed\_batch[j] \* 255).astype(np.uint8))

dream\_image\_pil.save(output\_image\_path)

print("Dream Booth process is complete. Enhanced images saved in:", output\_folder)

**Control Net**

import torch

import torchvision.transforms as transforms

import torchvision.models as models

import numpy as np

from PIL import Image

import os

import matplotlib.pyplot as plt

def load\_images\_from\_folder(folder\_path):

images = []

for filename in os.listdir(folder\_path):

image\_path = os.path.join(folder\_path, filename)

if os.path.isfile(image\_path):

image = Image.open(image\_path).convert('RGB')

images.append((image, filename))

return images

transform = transforms.Compose([

transforms.RandomHorizontalFlip(p=0.5),

transforms.RandomVerticalFlip(p=0.5),

transforms.RandomRotation(degrees=45),

transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5, hue=0.5),

transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0)), # Normalization for ResNet

transforms.ToTensor()

])

class ControlNet(torch.nn.Module):

def \_init\_(self):

super(ControlNet, self).\_init\_()

self.resnet = models.resnet18(pretrained=True)

self.resnet.fc = torch.nn.Linear(self.resnet.fc.in\_features, 1)

def forward(self, x):

return torch.sigmoid(self.resnet(x)).squeeze()

def apply\_control\_net(image\_tensor, control\_net):

control\_signal = control\_net(image\_tensor)

augmented\_image = (image\_tensor \* control\_signal \* 255).clamp(0, 255).byte()

return augmented\_image

input\_folder = r"C:\Users\MY\Desktop\drought"

images = load\_images\_from\_folder(input\_folder)

control\_net = ControlNet()

output\_folder = r"C:\Users\MY\Desktop\aug"

os.makedirs(output\_folder, exist\_ok=True)

for image, filename in images:

for i in range(5): # Apply augmentation 5 times

image\_tensor = transform(image).unsqueeze(0)

augmented\_image\_tensor = apply\_control\_net(image\_tensor, control\_net)

augmented\_image = augmented\_image\_tensor.squeeze(0).permute(1, 2, 0).numpy().astype(np.uint8)

output\_image\_path = os.path.join(output\_folder, f'{os.path.splitext(filename)[0]}\_{i}.jpg')

augmented\_image\_pil = Image.fromarray(augmented\_image)

augmented\_image\_pil.save(output\_image\_path)