Pipeline Overview

1. **Data Preparation**:
    - **Dataset**: Images and their corresponding segmentation masks are loaded. The dataset should be split into training, validation, and test sets.
    - **Preprocessing**:
        - Images are resized to 512x512 and normalized to the range [0, 1].
        - Masks are also resized to match the image dimensions.
        - Data augmentation can be added for better generalization.
2. **Model Definitions**:
    - **U-Net++**: A deep learning model for semantic segmentation using the EfficientNet-b3 encoder.
    - **FCN-8s**: Another segmentation model with a fully convolutional architecture.
    - **DC-GAN**: A Generative Adversarial Network for generating synthetic segmentation masks.
3. **Training Setup**:
    - **Loss Function**: Typically, CrossEntropyLoss for segmentation tasks.
    - **Optimizer**: Adam optimizer is used to minimize the loss function.
    - **Learning Rate**: A learning rate is defined for each model.
4. **Training Loop**:
    - **U-Net++ and FCN-8s**: Both models are trained using a supervised learning approach with segmentation masks.
    - **DC-GAN**: The generator and discriminator are trained adversarially, where the generator produces synthetic masks, and the discriminator distinguishes real from fake masks.
5. **Evaluation**:
    - After training, the models are evaluated on a test set.
    - Predicted masks are generated for test images and compared with ground truth masks.
    - Performance metrics like **IoU**, **F1 score**, **precision**, **recall**, etc., are computed using a classification report.
6. **Visualization**:
    - The predicted masks from all models (U-Net++, FCN-8s, and DC-GAN) are visualized for comparison against the ground truth.

CODE :

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import cv2
import numpy as np
import matplotlib.pyplot as plt
from torchvision import transforms
from sklearn.metrics import classification_report
import segmentation_models_pytorch as smp

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define class names
NUM_CLASSES = 6
CLASS_LABELS = ['background', 'barren land', 'roads', 'urban', 'vegetation', 'water']

# -------- Dataset Definition --------
class SegmentationDataset(Dataset):
    def __init__(self, image_paths, mask_paths, transform=None):
        self.image_paths = image_paths
        self.mask_paths = mask_paths
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image = cv2.imread(self.image_paths[idx])
        mask = cv2.imread(self.mask_paths[idx], cv2.IMREAD_GRAYSCALE)
```

```
        if self.transform:
            image = self.transform(image)
        return image, mask

# Example file paths (replace with actual paths)
image_paths = ["image1.jpg", "image2.jpg", "image3.jpg"]
mask_paths = ["mask1.png", "mask2.png", "mask3.png"]

# Example transformation
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((512, 512))
])

# Dataset and DataLoader
dataset = SegmentationDataset(image_paths, mask_paths, transform=transform)
dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

# -------- Model Definitions --------
# U-Net++ with EfficientNet backbone
unet_model = smp.UnetPlusPlus(
    encoder_name="efficientnet-b3",
    encoder_weights="imagenet",
    in_channels=3,
    classes=NUM_CLASSES,
    activation=None
).to(device)

# FCN-8s Model (as defined earlier)
class FCN8s(nn.Module):
    def __init__(self):
```

```
        super(FCN8s, self).__init__()
        # Define FCN-8s layers here...

    def forward(self, x):
        # Define forward pass for FCN-8s here...
        return x

fcn_model = FCN8s().to(device)

# DC-GAN Generator (as defined earlier)
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        # Define DC-GAN Generator layers here...

    def forward(self, z):
        # Define forward pass for DC-GAN Generator
        return z

generator = Generator().to(device)
discriminator = Discriminator().to(device)  # DC-GAN Discriminator (defined earlier)

# -------- Training Setup --------
criterion = torch.nn.CrossEntropyLoss()
optimizer_unet = torch.optim.Adam(unet_model.parameters(), lr=1e-4)
optimizer_fcn = torch.optim.Adam(fcn_model.parameters(), lr=1e-4)

# Optimizers for DC-GAN
optimizer_g = torch.optim.Adam(generator.parameters(), lr=1e-4)
optimizer_d = torch.optim.Adam(discriminator.parameters(), lr=1e-4)
```

```
# -------- Training Loop for U-Net++ and FCN-8s --------
epochs = 10  # Define number of epochs

for epoch in range(epochs):
    unet_model.train()
    fcn_model.train()

    for images, masks in dataloader:
        images = images.to(device)
        masks = masks.to(device)

        # U-Net++ Training
        optimizer_unet.zero_grad()
        outputs_unet = unet_model(images)
        loss_unet = criterion(outputs_unet, masks)
        loss_unet.backward()
        optimizer_unet.step()

        # FCN-8s Training
        optimizer_fcn.zero_grad()
        outputs_fcn = fcn_model(images)
        loss_fcn = criterion(outputs_fcn, masks)
        loss_fcn.backward()
        optimizer_fcn.step()

    print(f"Epoch [{epoch+1}/{epochs}], U-Net++ Loss: {loss_unet.item()}, FCN-8s Loss:
{loss_fcn.item()}")

# -------- DC-GAN Training --------
for epoch in range(epochs):
    generator.train()
```

```
    discriminator.train()

    for _ in range(len(dataloader)):  # Iterate over the dataloader
        # Real data
        real_images = next(iter(dataloader))[1].to(device)
        real_labels = torch.ones(batch_size, 1).to(device)

        # Fake data
        z = torch.randn(batch_size, 100).to(device)
        fake_images = generator(z)

        # Discriminator Loss (Real vs Fake)
        optimizer_d.zero_grad()
        real_loss = discriminator(real_images).mean()
        fake_loss = discriminator(fake_images.detach()).mean()
        d_loss = -(real_loss - fake_loss)
        d_loss.backward()
        optimizer_d.step()

        # Generator Loss
        optimizer_g.zero_grad()
        g_loss = -discriminator(fake_images).mean()
        g_loss.backward()
        optimizer_g.step()

    print(f"Epoch [{epoch+1}/{epochs}], DC-GAN Loss: {d_loss.item()}")

# -------- Model Evaluation --------
with torch.no_grad():
    test_image = preprocess_image("test_image.jpg")
    output_unet = unet_model(test_image)
```

```
    pred_mask_unet = torch.argmax(output_unet, dim=1).squeeze().cpu().numpy()

    output_fcn = fcn_model(test_image)
    pred_mask_fcn = torch.argmax(output_fcn, dim=1).squeeze().cpu().numpy()

    # Generate synthetic mask using DC-GAN
    noise = torch.randn((1, 100)).to(device)
    generated_mask = generator(noise).squeeze().cpu().numpy()
    generated_mask = (generated_mask * 255).astype(np.uint8)

# -------- Visualization --------
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title("Ground Truth")
plt.imshow(ground_truth_mask, cmap="jet")

plt.subplot(1, 3, 2)
plt.title("U-Net++ Prediction")
plt.imshow(pred_mask_unet, cmap="jet")

plt.subplot(1, 3, 3)
plt.title("FCN-8s Prediction")
plt.imshow(pred_mask_fcn, cmap="jet")

plt.show()

# -------- Classification Report --------
print("U-Net++ Classification Report:")
print(classification_report(ground_truth_mask.flatten(), pred_mask_unet.flatten(),
target_names=CLASS_LABELS))
```

```
print("FCN-8s Classification Report:")

print(classification_report(ground_truth_mask.flatten(), pred_mask_fcn.flatten(),
target_names=CLASS_LABELS))
```