REPORT

MODEL: SeggFormer + DeepLabv3(Backbone = resnet50)

LAYERS: 296

PARAMETERS: 304326632

# System level diagram:

1. Input Image: The initial input image to be segmented.
2. DeeplabV3 Backbone: The backbone network that captures hierarchical features from the input image. It might include convolutional layers, pooling, and other operations.
3. Intermediate Features: Intermediate features extracted by the DeeplabV3 backbone, representing rich information about the input.
4. Segformer Decoder Head: The Segformer decoder processes intermediate features, incorporating transformer-based operations for context-aware feature refinement.
5. Upsampling and Aggregation: The Segformer decoder may include upsampling operations and feature aggregation to enhance spatial resolution and fuse information.
6. Final Segmentation Map: The output of the Segformer decoder, providing a pixel-wise segmentation map with class labels.

# Modules:

## **DeepLabV3 Backbone:**

Purpose: Captures hierarchical features from the input image.

Components:

ResNet-50 architecture, pretrained on a large dataset.

Consists of convolutional layers, pooling, and residual blocks.

Responsibility: Extracts rich information at different scales.

## **Intermediate Features:**

Purpose: Represents hierarchical information about the input image.

Components:

Features extracted from different layers of the DeeplabV3 backbone.

Responsibility: Provides a set of feature maps capturing details at various levels.

## **Segformer Decoder Head:**

Purpose: Processes intermediate features for segmentation.

Components:

Transformer-based decoder, inspired by the Segformer architecture.

Responsibility: Utilizes self-attention mechanisms for context-aware feature refinement.

## **Upsampling and Aggregation:**

Purpose: Enhances spatial resolution and fuses information.

Components:

Upsampling operations (e.g., bilinear upsampling).

Feature aggregation techniques (e.g., skip connections).

Responsibility: Ensures that the output segmentation map has sufficient detail and context.

## **Final Segmentation Map:**

Purpose: Provides pixel-wise segmentation with class labels.

Components:

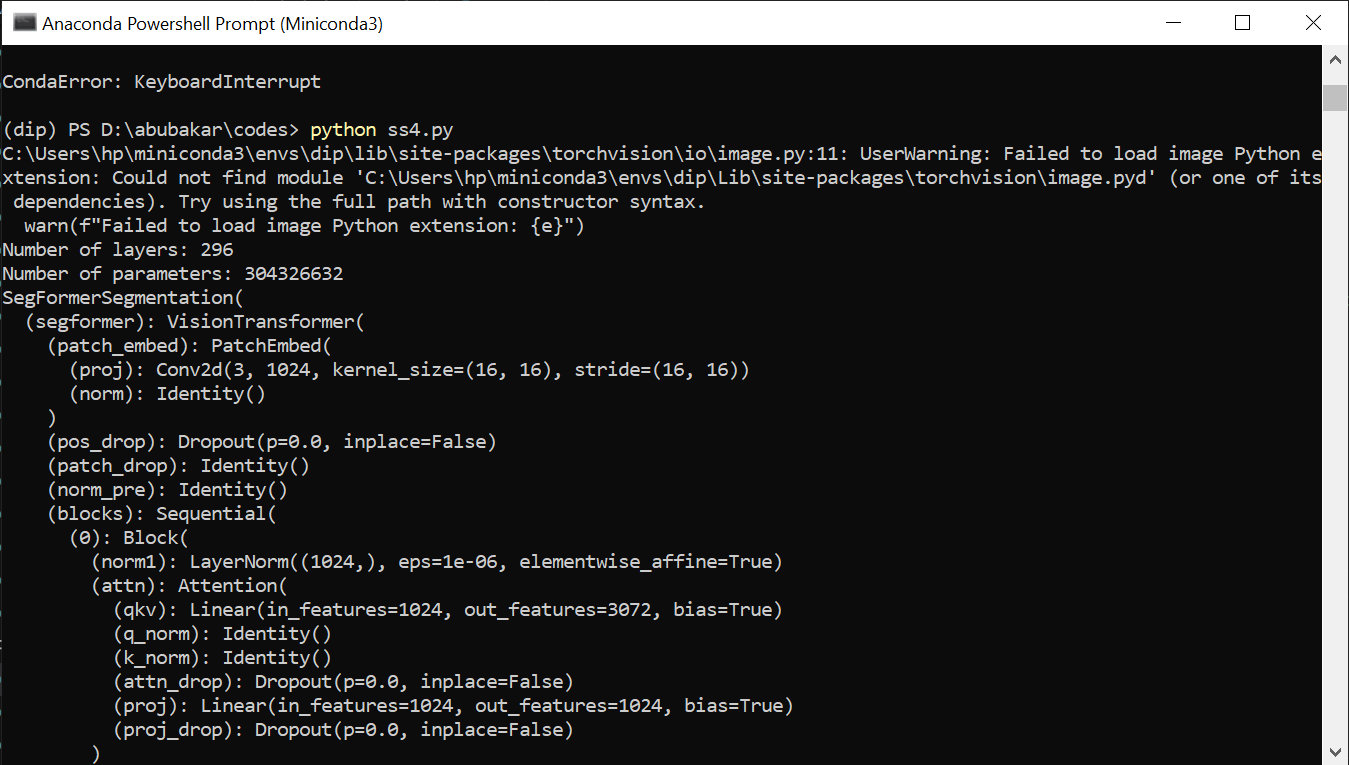
Output layer with the desired number of classes.

Activation function appropriate for the task (e.g., sigmoid or softmax).

Responsibility: Produces the final segmentation output.

# Model parameters:

NOTE: SS4.py is specified to print only the selected model layers and parameters count and generate a model summary by simply using: print(model)



## **PARAMETERS OF SEGFORMER**

## **image\_size (tuple):**

Specifies the size of the input image. It is a tuple of two integers (height, width).

## **patch\_size (int):**

Determines the size of non-overlapping patches that the image is divided into for processing by the transformer. Larger patch sizes can capture more global context but may result in a larger model.

## **in\_channels (int):**

The number of input channels. This should be set to the number of output channels from the backbone network that you want to use as an encoder. In your case, it should match the output channels of the DeepLabV3 backbone.

## **embed\_dim (int):**

The dimensionality of the embeddings for each patch. This value determines the size of the hidden representations in the transformer layers.

## **num\_heads (int):**

The number of attention heads in the multi-head self-attention mechanism. More heads can capture different aspects of relationships in the input data.

## **num\_encoder\_layers (int):**

The number of transformer encoder layers. This parameter controls the depth of the model. Deeper models may capture more complex patterns but may require more computation.

## **num\_decoder\_layers (int):**

The number of transformer decoder layers. Similar to the encoder layers, this parameter controls the depth of the decoder part of the model.

## **num\_classes (int):**

The number of output classes for semantic segmentation. It defines the dimensionality of the output predictions.

## **dropout\_rate (float):**

Dropout rate used in the transformer layers. Dropout can be applied to prevent overfitting and improve generalization.

# LEARNING PROGRAM:

Learing rate = 0.001

Epoche = 2

Optimizer = ADAM (adaptive moment estimation)

Loss function = BCE(Binary cross entropy)

Images = 80-20 train & validation split

Note: I tried to minimize the system requirements by distributing training in worker but didn’t worked!

## **MODEL**

from segformer.model.segformer import Segformer

class SegFormerWithDeepLabBackbone(nn.Module):

    def \_\_init\_\_(self, num\_classes):

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

        #USING DeepLabV3 model with ResNet-50 backbone

        deeplabv3\_model = deeplabv3\_resnet50(pretrained=True)

        deeplabv3\_backbone = nn.Sequential(\*list(deeplabv3\_model.children())[:-1])

        self.segformer = Segformer(

            image\_size=(256, 256),

            patch\_size=4,

            in\_channels=2048,

            embed\_dim=768,

            num\_heads=12,

            num\_encoder\_layers=12,

            num\_decoder\_layers=12,

            num\_classes=num\_classes,

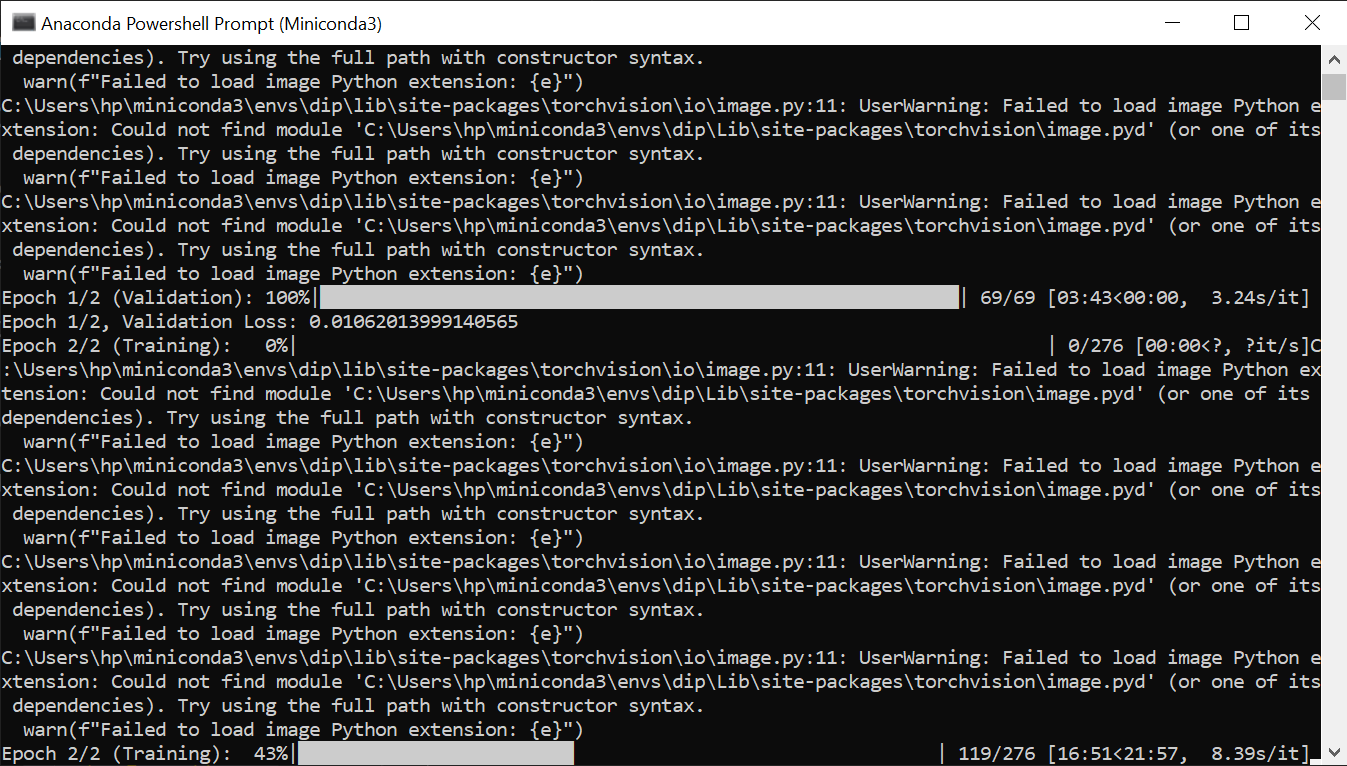
        )

        self.segformer.encoder = deeplabv3\_backbone

    def forward(self, x):

        return self.segformer(x)

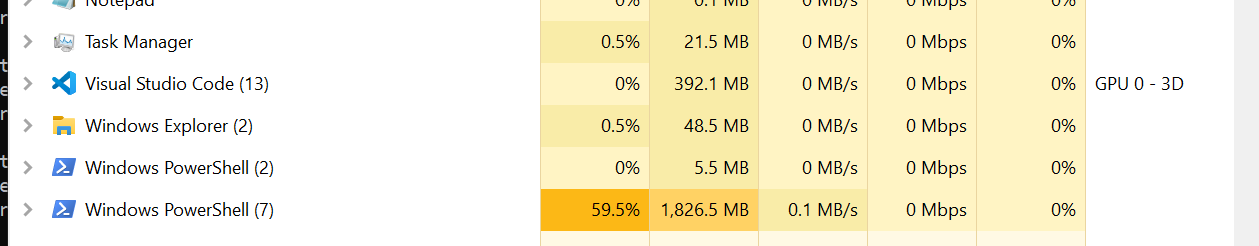
NOTE: libs are marked red because none are installed in pip env as im working on venv

 A screenshot of a computer program

Description automatically generated



NOTE: getting these warnings because my conda environment is unable to execute a python protocol



## 

## Learning code:

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, Dataset

from torchvision import transforms

from torchvision.datasets import ImageFolder

from torchvision.models.segmentation import deeplabv3\_resnet50

from PIL import Image

from tqdm import tqdm

import torch.nn.functional as F

from torch.utils.data import random\_split

import matplotlib.pyplot as plt

import cv2

import numpy as np

from torch.utils.data import Subset

class DeepLabV3Segmentation(nn.Module):

    def \_\_init\_\_(self, num\_classes):

        super(DeepLabV3Segmentation, self).\_\_init\_\_()

        self.deeplabv3 = deeplabv3\_resnet50(pretrained=True)

        self.deeplabv3.classifier = nn.Conv2d(2048, num\_classes, kernel\_size=1)

    def forward(self, x):

        return self.deeplabv3(x)['out']

transform\_input = transforms.Compose([

    transforms.Resize((256, 256)),

    transforms.ToTensor(),

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

])

transform\_mask = transforms.Compose([

    transforms.Resize((256, 256)),

    transforms.ToTensor(),

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

])

class SemanticSegmentationDataset(Dataset):

    def \_\_init\_\_(self, root, transform\_input, transform\_mask):

        self.dataset = ImageFolder(root, transform=None)

        self.transform\_mask = transform\_mask

        self.transform\_input = transform\_input

    def \_\_len\_\_(self):

        return len(self.dataset)

    def \_\_getitem\_\_(self, idx):

        img, \_ = self.dataset[idx]

        mask\_path = self.dataset.imgs[idx][0].replace("images", "gt")

        mask = Image.open(mask\_path).convert("L")

        #print("before trans:",mask.size)

        #annotations\_path = mask\_path

        #original\_annotations = cv2.imread(annotations\_path, cv2.IMREAD\_GRAYSCALE)

        #annotations\_normalized = (original\_annotations - np.min(original\_annotations)) / (np.max(original\_annotations) - np.min(original\_annotations)) \* 255

        #dilation\_kernel\_size = 5

        #kernel = np.ones((dilation\_kernel\_size, dilation\_kernel\_size), np.uint8)

        #dilated\_annotations = cv2.dilate(annotations\_normalized, kernel, iterations=1)

        #dilated\_annotations\_pil = Image.fromarray(dilated\_annotations)

        #mask = self.transform\_mask(dilated\_annotations\_pil)

        mask = self.transform\_mask(mask)

        img = self.transform\_input(img)

        #mask = mask.squeeze(0) if mask.dim() == 3 else mask

        #print("mask:",mask.shape,"\nimage:",img.shape)

        #input("Press Enter to continue...")

        return img, mask

def visualize(images, targets, predictions):

    images\_np = images.cpu().numpy()

    targets\_np = targets.cpu().numpy()

    predictions\_np = predictions.cpu().numpy()

    plt.figure(figsize=(15, 5))

    for i in range(images\_np.shape[0]):

        plt.subplot(3, images\_np.shape[0], i + 1)

        plt.imshow(images\_np[i, 0], cmap='gray')

        plt.title('Input')

        plt.subplot(3, images\_np.shape[0], i + 1 + images\_np.shape[0])

        plt.imshow(predictions\_np[i, 0], cmap='gray')

        plt.title('Predicted')

        plt.subplot(3, images\_np.shape[0], i + 1 + 2 \* images\_np.shape[0])

        plt.imshow(targets\_np[i, 0], cmap='gray')

        plt.title('Ground Truth')

    plt.show()

def visualize\_samples(model, dataloader, device):

    model.eval()

    images, targets = next(iter(dataloader))

    images, targets = images.to(device), targets.to(device)

    outputs = model(images)

    predictions = torch.argmax(F.softmax(outputs['out'], dim=1), dim=1)

    visualize(images, targets, predictions)

num\_epochs = 2

batch\_size = 4

# LOADING DATA, CALLING DATALOADER, 80-20 SPLIT

dataset\_path = r"D:\abubakar\dataset"

semantic\_dataset = SemanticSegmentationDataset(root=dataset\_path, transform\_input = transform\_input, transform\_mask=transform\_mask)

total\_size = len(semantic\_dataset)

#total\_size = int(0.1 \* total\_size)

train\_size = int(0.8 \* total\_size)

val\_size = total\_size - train\_size

train\_dataset, val\_dataset = random\_split(semantic\_dataset, [train\_size, val\_size])

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

if \_\_name\_\_ == "\_\_main\_\_":

    torch.multiprocessing.freeze\_support()

    train\_dataloader = DataLoader(train\_dataset, batch\_size, shuffle=True, num\_workers=4, pin\_memory=True)

    val\_dataloader = DataLoader(val\_dataset, batch\_size, shuffle=False, num\_workers=4, pin\_memory=True)

    # Initialize model, loss function, and optimizer

    model = DeepLabV3Segmentation(num\_classes=1).to(device)

    #criterion = nn.CrossEntropyLoss()

    criterion = nn.BCEWithLogitsLoss()

    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # Count layers and parameters

    num\_layers = len(list(model.parameters()))

    num\_parameters = sum(p.numel() for p in model.parameters())

    print(f"Number of layers: {num\_layers}")

    print(f"Number of parameters: {num\_parameters}")

    #   TRAINING LOOP

    for epoch in range(num\_epochs):

        model.train()

        total\_loss = 0

        for images, targets in tqdm(train\_dataloader, desc=f'Epoch {epoch + 1}/{num\_epochs} (Training)'):

            optimizer.zero\_grad()

            outputs = model(images)

            loss = criterion(outputs, targets)

            loss.backward()

            optimizer.step()

            total\_loss += loss.item()

        average\_loss = total\_loss / len(train\_dataloader)

        print(f'Epoch {epoch + 1}/{num\_epochs}, Training Loss: {average\_loss}')

        #   VALIDATION LOOP

        model.eval()

        total\_val\_loss = 0

        with torch.no\_grad():

            for val\_images, val\_targets in tqdm(val\_dataloader, desc=f'Epoch {epoch + 1}/{num\_epochs} (Validation)'):

                val\_outputs = model(val\_images)

                val\_loss = criterion(val\_outputs, val\_targets)

                total\_val\_loss += val\_loss.item()

        average\_val\_loss = total\_val\_loss / len(val\_dataloader)

        print(f'Epoch {epoch + 1}/{num\_epochs}, Validation Loss: {average\_val\_loss}')

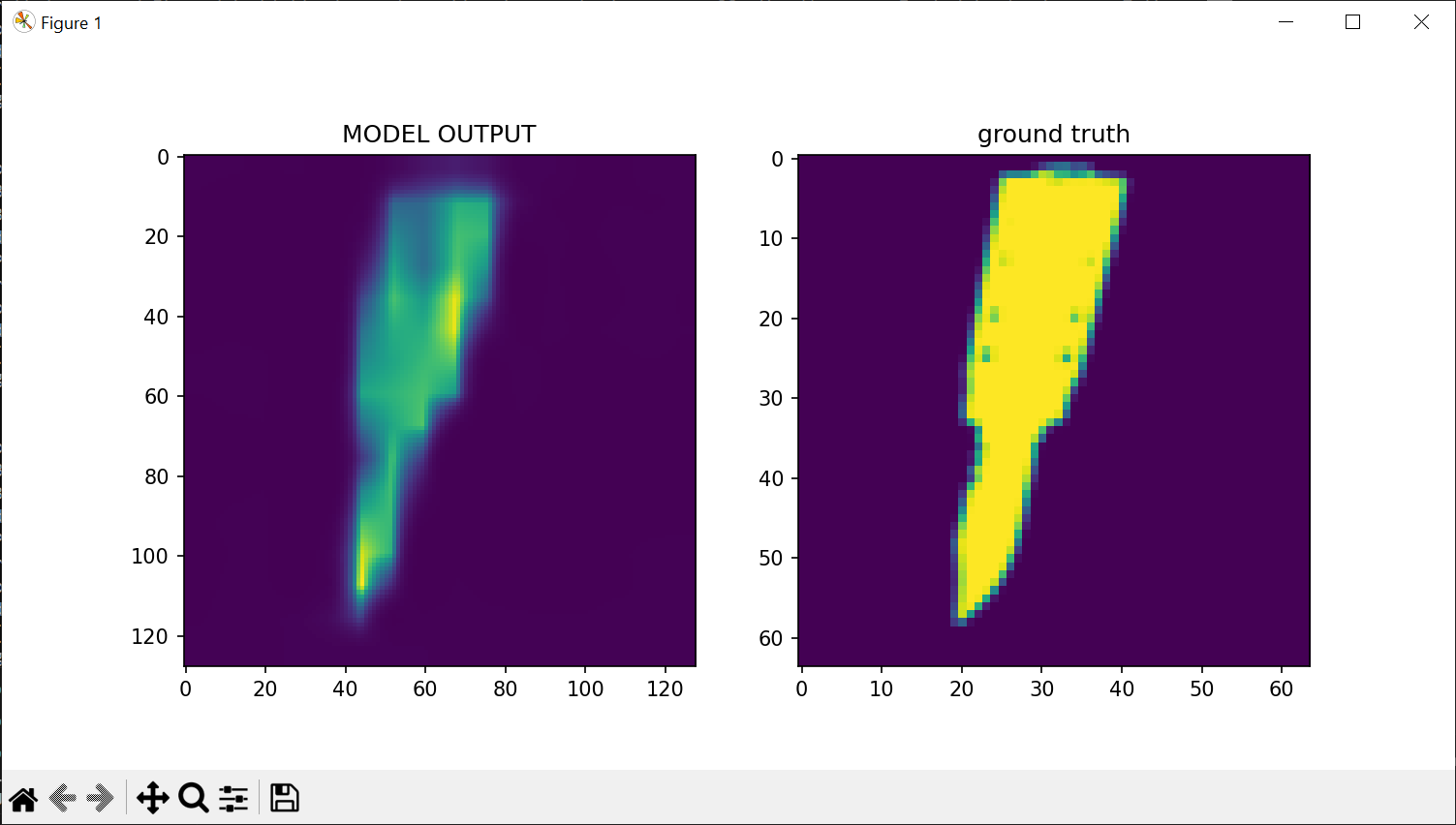
        #visualize\_samples(model, val\_dataloader, device)

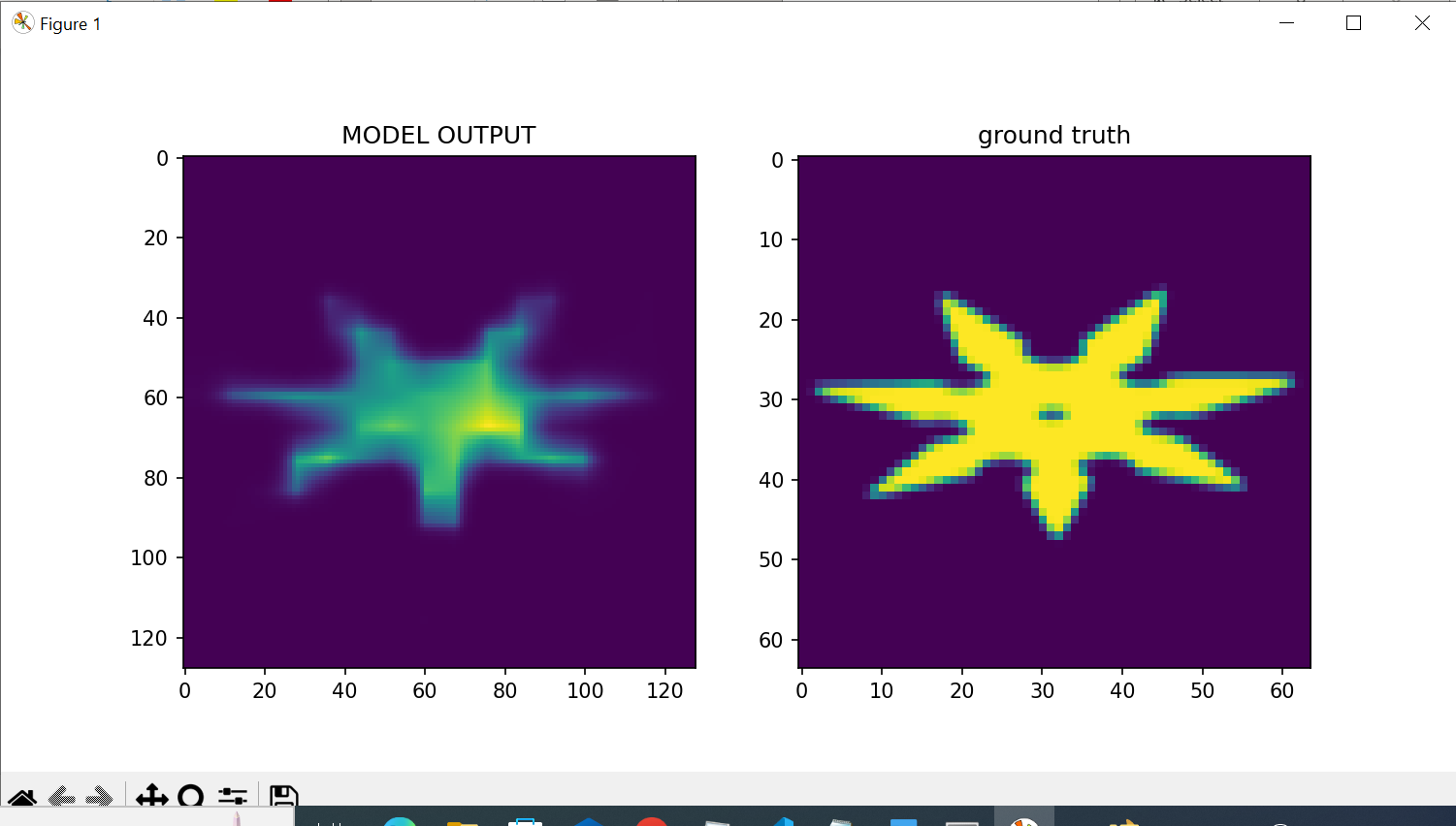
    # Save the trained model

    torch.save(model.state\_dict(), r'D:\abubakar\models\deeplabv3\_semantic\_segmentation\_model1.pth')

# RESULTS:

## **Test images**

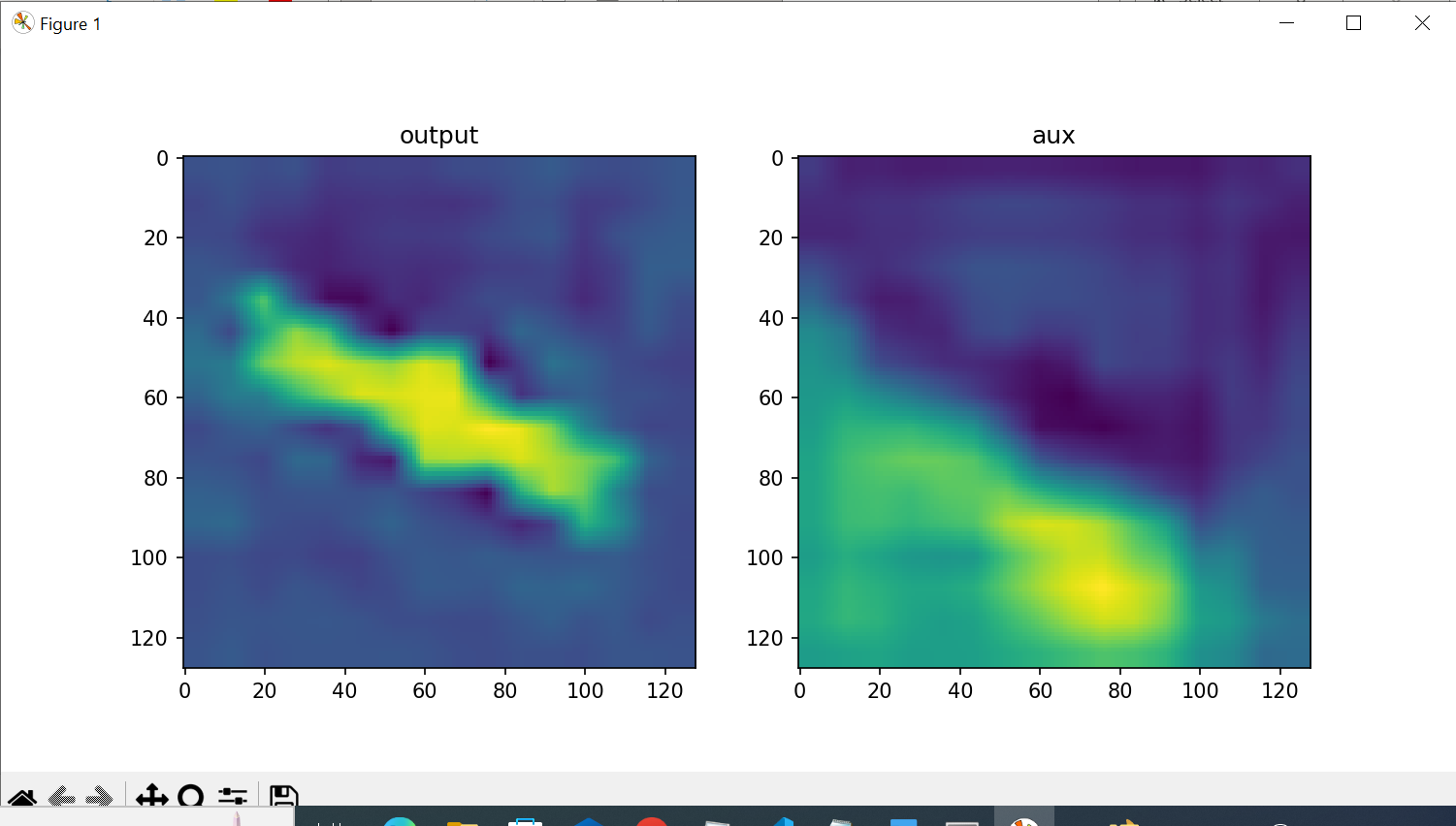




A screenshot of a computer generated image

Description automatically generated

## **Images produced by SEGFORMER**



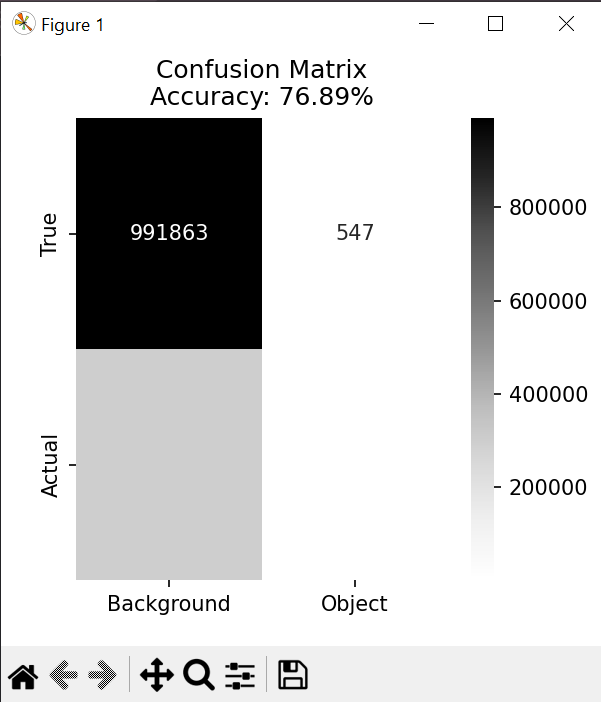
A screenshot of a computer screen

Description automatically generated

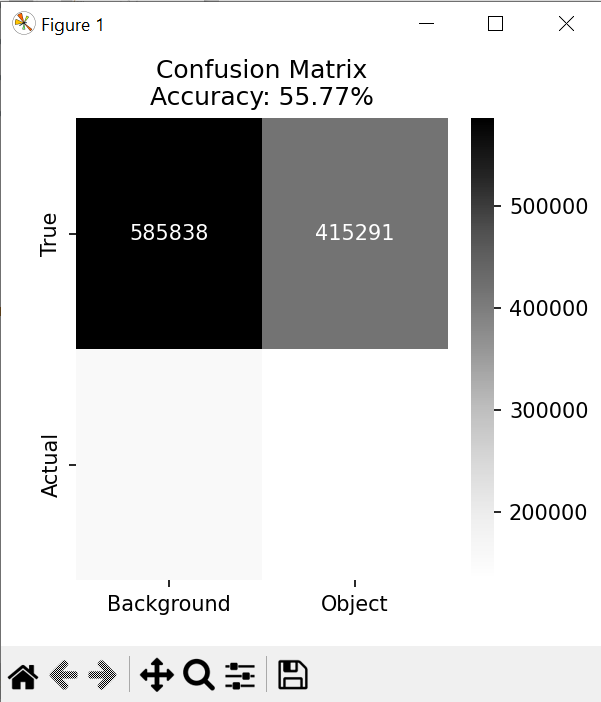
Screens screenshot of a computer screen

Description automatically generated

## **CONFUSION MATRIX**



## ADJUSTING THRESHOLD TO CREATE COMPLETE CONFUSION MATRIX

A graph of a graph

Description automatically generated with medium confidence

## **Test-Run window:**

