- Satheesh D M
- MA24M023
- Task 4 - HED

```python
# Import necessary libraries
import torch
import torch.nn as nn
import torchvision.models as models
import logging
import os
import csv
import numpy as np
import random
import torch.nn.functional as F
import os
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt

# Seed setting function
def set_seed(seed=42):
    """
    Set the seed for reproducibility in PyTorch, NumPy, and Python's
random module on MPS.
    """
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)

    # For MPS (Apple Silicon)
    if torch.backends.mps.is_available():
        torch.mps.manual_seed(seed)
        print("Seed set for MPS.")

    torch.use_deterministic_algorithms(True, warn_only=True)

    print(f"Seed set to: {seed}")

# Example Usage
set_seed(42)
```

```
Seed set for MPS.
Seed set to: 42
```

```python
# HED (Holistically-Nested Edge Detection) model class
# This class implements the HED model using a VGG16 backbone.
class HED(nn.Module):
    def __init__(self):
```

```python
        super(HED, self).__init__()

        # Load VGG16 as base network
        vgg16 = models.vgg16(weights=models.VGG16_Weights.DEFAULT)
        features = list(vgg16.features.children())

        # Encoder (VGG16 backbone, WITHOUT last maxpool)
        self.conv1 = nn.Sequential(*features[:5])
        self.conv2 = nn.Sequential(*features[5:10])
        self.conv3 = nn.Sequential(*features[10:17])
        self.conv4 = nn.Sequential(*features[17:24])
        self.conv5 = nn.Sequential(*features[24:29])

        # Side output layers (1x1 conv to get single-channel logits)
        self.side1 = nn.Conv2d(64, 1, kernel_size=1)
        self.side2 = nn.Conv2d(128, 1, kernel_size=1)
        self.side3 = nn.Conv2d(256, 1, kernel_size=1)
        self.side4 = nn.Conv2d(512, 1, kernel_size=1)
        self.side5 = nn.Conv2d(512, 1, kernel_size=1)

        # Learnable fusion weights
        self.weights = nn.Parameter(torch.ones(5,
dtype=torch.float32))

    def forward(self, x):
        img_size = x.shape[2:]

        # Forward pass through VGG16 layers
        c1 = self.conv1(x)
        c2 = self.conv2(c1)
        c3 = self.conv3(c2)
        c4 = self.conv4(c3)
        c5 = self.conv5(c4)

        # Compute side outputs (raw logits)
        s1 = self.side1(c1)
        s2 = self.side2(c2)
        s3 = self.side3(c3)
        s4 = self.side4(c4)
        s5 = self.side5(c5)

        # Upsample side outputs to match input size
        s1 = F.interpolate(s1, size=img_size, mode="bilinear",
align_corners=True)
        s2 = F.interpolate(s2, size=img_size, mode="bilinear",
align_corners=True)
        s3 = F.interpolate(s3, size=img_size, mode="bilinear",
align_corners=True)
        s4 = F.interpolate(s4, size=img_size, mode="bilinear",
align_corners=True)
```

```
        s5 = F.interpolate(s5, size=img_size, mode="bilinear",
align_corners=True)

        # Normalize the weights using softmax (to ensure non-negative
weights)
        normalized_weights = F.softmax(self.weights, dim=0)

        # Final fused output using learnable weighted sum
        fused = sum(w * s for w, s in zip(normalized_weights, [s1, s2,
s3, s4, s5]))

        return s1, s2, s3, s4, s5, fused

# Balanced BCE with logits loss function
# This loss function is designed to handle class imbalance in binary
classification tasks.
class BalancedBCEWithLogitsLoss(nn.Module):
    def __init__(self):
        super(BalancedBCEWithLogitsLoss, self).__init__()

    def forward(self, pred, target):
        # Class balancing
        pos_count = torch.sum(target)
        neg_count = target.numel() - pos_count
        beta = neg_count / (pos_count + neg_count + 1e-6)

        weights = beta * target + (1 - beta) * (1 - target) + 1e-4
        loss = F.binary_cross_entropy_with_logits(pred, target,
weight=weights)

        return loss

def train_and_validate(model, train_loader, val_loader, criterion,
optimizer, num_epochs=100,
                        save_path='checkpoints',
model_filename='model.pth', csv_filename='losses.csv',
                        unfreeze_epoch=20, clip_value=1.0):

    # Device Configuration
    device = torch.device('mps' if torch.backends.mps.is_available()
else 'cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)

    # Logging Configuration
    logging.basicConfig(filename='training.log', level=logging.INFO,
                        format='%(asctime)s - %(levelname)s - %
(message)s')
    os.makedirs(save_path, exist_ok=True)

    # CSV Logging
```

```
    csv_path = os.path.join(save_path, csv_filename)
    with open(csv_path, mode='w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(["Epoch", "Train Loss", "Validation Loss",
"LR"])

    # Freeze Encoder Initially
    for param in model.conv1.parameters():
        param.requires_grad = False
    for param in model.conv2.parameters():
        param.requires_grad = False

    logging.info(f"Encoder frozen until epoch {unfreeze_epoch}")

    for epoch in range(num_epochs):

        # Unfreeze Encoder
        if epoch == unfreeze_epoch:
            for param in model.conv1.parameters():
                param.requires_grad = True
            for param in model.conv2.parameters():
                param.requires_grad = True
            logging.info(f"Encoder unfrozen at epoch {epoch+1}")

        # Training Phase
        model.train()
        epoch_loss = 0
        for images, edges in train_loader:
            images, edges = images.to(device), edges.to(device)

            optimizer.zero_grad()
            side_outputs = model(images)
            loss = sum(0.1 * criterion(out, edges) for out in
side_outputs[:-1]) + 0.5 * criterion(side_outputs[-1], edges)
            loss.backward()

            # torch.nn.utils.clip_grad_norm_(model.parameters(),
clip_value)
            optimizer.step()

            epoch_loss += loss.item()

        train_loss = epoch_loss / len(train_loader)

        # Validation Phase
        model.eval()
        val_loss = 0
        with torch.no_grad():
            for images, edges in val_loader:
                images, edges = images.to(device), edges.to(device)
```

```
                side_outputs = model(images)
                loss = sum(0.1 * criterion(out, edges) for out in
side_outputs[:-1]) + 0.5 * criterion(side_outputs[-1], edges)
                val_loss += loss.item()

        val_loss /= len(val_loader)

        # Logging
        with open(csv_path, mode='a', newline='') as f:
            writer = csv.writer(f)
            writer.writerow([epoch+1, train_loss, val_loss])

        log_msg = (f'Epoch [{epoch+1}/{num_epochs}], Train Loss:
{train_loss}, '
                  f'Validation Loss: {val_loss}')
        logging.info(log_msg)
        print(log_msg)

    logging.info("Training completed.")
    return train_loss, val_loss

# Dataloader for BSDS500 dataset
class BSDS500(Dataset):
    def __init__(self, image_dir, edge_dir, transform=None,
edge_transform=None):
        """
        Custom dataloader for BSDS500 edge detection dataset using JPG
ground truth.

        Args:
            image_dir (str): Path to image directory (train, val,
test).
            edge_dir (str): Path to corresponding edge ground truth
directory.
            transform (callable, optional): Transformations for
images.
            edge_transform (callable, optional): Transformations for
edge maps.
        """
        self.image_dir = image_dir
        self.edge_dir = edge_dir
        self.transform = transform
        self.edge_transform = edge_transform
        self.image_files = [f for f in os.listdir(image_dir) if
f.endswith('.jpg')]

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

    def __getitem__(self, idx):
```

```
        # Load Image
        img_name = self.image_files[idx]
        img_path = os.path.join(self.image_dir, img_name)
        image = Image.open(img_path).convert('RGB')

        # Load Ground Truth Edge Image
        edge_path = os.path.join(self.edge_dir, img_name)
        edge_image = Image.open(edge_path).convert('L')

        # Apply transformations
        if self.transform:
            image = self.transform(image)
        if self.edge_transform:
            edge_image = self.edge_transform(edge_image)

        return image, edge_image

# Separate transforms
vgg_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]),
])

edge_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

g = torch.Generator()
g.manual_seed(42)

# Create Dataloaders
train_dataset = BSDS500(image_dir='archive/images/train',
edge_dir='archive/ground_truth_boundaries/train',
                         transform=vgg_transform,
edge_transform=edge_transform)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True,
num_workers=0, generator=g)

val_dataset = BSDS500(image_dir='archive/images/val',
edge_dir='archive/ground_truth_boundaries/val',
                       transform=vgg_transform,
edge_transform=edge_transform)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=True,
num_workers=0, generator=g)

import torch.optim as optim
# Initialize model, criterion, and optimizer
```

```
model = HED()
criterion = BalancedBCEWithLogitsLoss()
lrate = 0.00001
optimizer = optim.Adam(model.parameters(), lr=lrate)

# Train and Validate
train_losses, val_losses = train_and_validate(model, train_loader,
val_loader, criterion, optimizer, num_epochs=100, unfreeze_epoch=0)
```

```
Epoch [1/100], Train Loss: 0.024279602158528108, Validation Loss:
0.02119336597621441
Epoch [2/100], Train Loss: 0.021187837307269756, Validation Loss:
0.019885005839169027
Epoch [3/100], Train Loss: 0.020111691779815234, Validation Loss:
0.019289359785616397
Epoch [4/100], Train Loss: 0.019603514614013526, Validation Loss:
0.01882300157099962
Epoch [5/100], Train Loss: 0.01890871351441512, Validation Loss:
0.0184438105672597 9
Epoch [6/100], Train Loss: 0.018868174547186263, Validation Loss:
0.01813564945012331
Epoch [7/100], Train Loss: 0.018163534239507638, Validation Loss:
0.017873233407735823
Epoch [8/100], Train Loss: 0.01803696470764967, Validation Loss:
0.017659873701632023
Epoch [9/100], Train Loss: 0.017649224887673672, Validation Loss:
0.017505216635763644
Epoch [10/100], Train Loss: 0.01742118768967115, Validation Loss:
0.017359039336442946
Epoch [11/100], Train Loss: 0.017302318404500302, Validation Loss:
0.017249102368950844
Epoch [12/100], Train Loss: 0.017161480151116848, Validation Loss:
0.017139359451830385
Epoch [13/100], Train Loss: 0.016998963287243478, Validation Loss:
0.017026588693261145
Epoch [14/100], Train Loss: 0.01714673566703613, Validation Loss:
0.016968912594020367
Epoch [15/100], Train Loss: 0.016990341031207487, Validation Loss:
0.016888134367763995
Epoch [16/100], Train Loss: 0.01687725862631431, Validation Loss:
0.016817577704787253
Epoch [17/100], Train Loss: 0.016617100614194687, Validation Loss:
0.016735530011355877
Epoch [18/100], Train Loss: 0.0164961377875163, Validation Loss:
0.016697839349508286
Epoch [19/100], Train Loss: 0.01640286583166856, Validation Loss:
0.016633544694632293
Epoch [20/100], Train Loss: 0.01631886619501389, Validation Loss:
0.016623538509011267
Epoch [21/100], Train Loss: 0.016414052376953456, Validation Loss:
```

```
0.016556833051145078
Epoch [22/100], Train Loss: 0.016313091063728698, Validation Loss:
0.01652522847056389
Epoch [23/100], Train Loss: 0.01626077974931552, Validation Loss:
0.016481719501316546
Epoch [24/100], Train Loss: 0.01618527749983164, Validation Loss:
0.016431764401495456
Epoch [25/100], Train Loss: 0.016298842688019458, Validation Loss:
0.016439159102737903
Epoch [26/100], Train Loss: 0.016124458482059147, Validation Loss:
0.01637819580733776
Epoch [27/100], Train Loss: 0.01589798440153782, Validation Loss:
0.01635995365679264
Epoch [28/100], Train Loss: 0.016038744137264214, Validation Loss:
0.01632537376135588
Epoch [29/100], Train Loss: 0.015980552308834516, Validation Loss:
0.016302243657410145
Epoch [30/100], Train Loss: 0.0160257642945418, Validation Loss:
0.016266349628567697
Epoch [31/100], Train Loss: 0.015952598088635847, Validation Loss:
0.016233062967658043
Epoch [32/100], Train Loss: 0.01602500806061121, Validation Loss:
0.016252119056880475
Epoch [33/100], Train Loss: 0.015982428995462563, Validation Loss:
0.016187492609024048
Epoch [34/100], Train Loss: 0.015818949645528428, Validation Loss:
0.016169397458434105
Epoch [35/100], Train Loss: 0.01582758121479016, Validation Loss:
0.01621460847556591
Epoch [36/100], Train Loss: 0.015638086944818497, Validation Loss:
0.01617169991135597
Epoch [37/100], Train Loss: 0.015564376559968177, Validation Loss:
0.016139276698231696
Epoch [38/100], Train Loss: 0.015799374462893374, Validation Loss:
0.016112384423613547
Epoch [39/100], Train Loss: 0.015818725268428143, Validation Loss:
0.016111194044351577
Epoch [40/100], Train Loss: 0.0156363844871521, Validation Loss:
0.016144463643431663
Epoch [41/100], Train Loss: 0.01572175803952492, Validation Loss:
0.016111608669161795
Epoch [42/100], Train Loss: 0.015430356877354475, Validation Loss:
0.016056072860956193
Epoch [43/100], Train Loss: 0.015627660788595676, Validation Loss:
0.016115845516324043
Epoch [44/100], Train Loss: 0.015485102812258096, Validation Loss:
0.016100288107991217
Epoch [45/100], Train Loss: 0.015628578499532662, Validation Loss:
0.016085770986974238
```

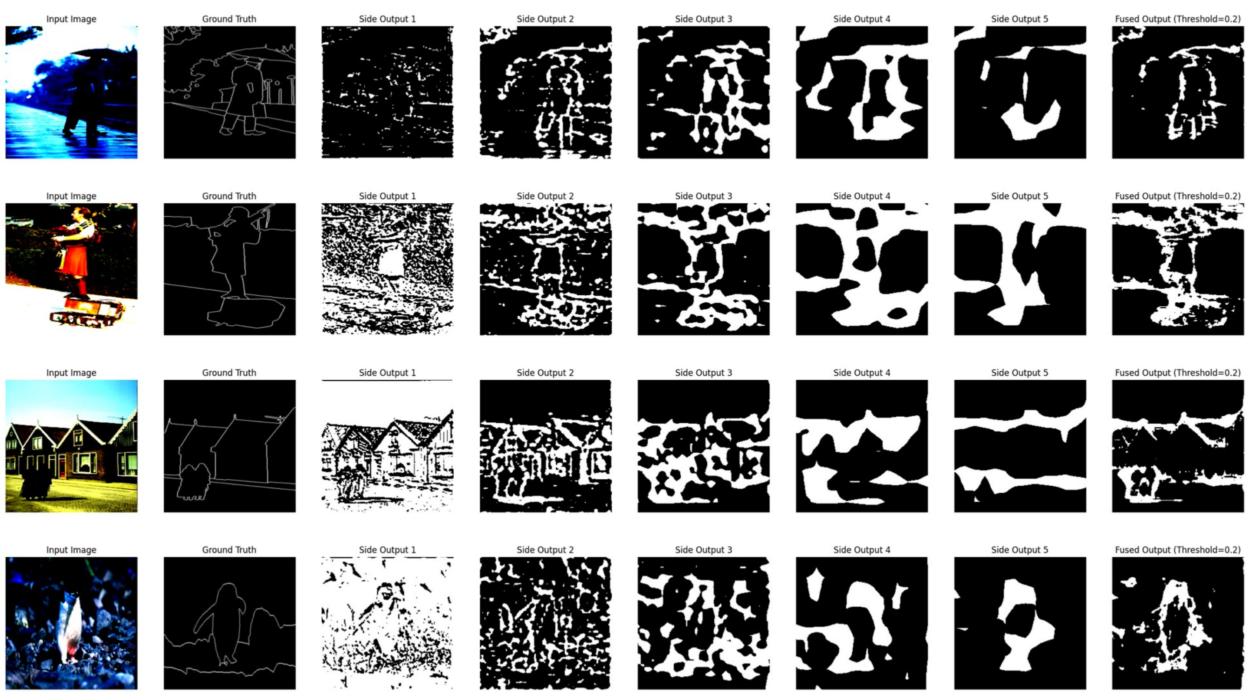
```
Epoch [46/100], Train Loss: 0.015550131479708048, Validation Loss:
0.01605994362384081
Epoch [47/100], Train Loss: 0.015394902859742824, Validation Loss:
0.01608270350843668
Epoch [48/100], Train Loss: 0.015398946590721607, Validation Loss:
0.015989103615283967
Epoch [49/100], Train Loss: 0.015568933736246366, Validation Loss:
0.016034671440720558
Epoch [50/100], Train Loss: 0.015545105991455225, Validation Loss:
0.01600679289549589
Epoch [51/100], Train Loss: 0.015415653586387634, Validation Loss:
0.016043171770870685
Epoch [52/100], Train Loss: 0.015303882268758921, Validation Loss:
0.015973283238708974
Epoch [53/100], Train Loss: 0.015376987723776927, Validation Loss:
0.015995102822780608
Epoch [54/100], Train Loss: 0.015535309051091854, Validation Loss:
0.016034921184182167
Epoch [55/100], Train Loss: 0.01519948960496829, Validation Loss:
0.016023233830928803
Epoch [56/100], Train Loss: 0.015217981946009856, Validation Loss:
0.01598191022872925
Epoch [57/100], Train Loss: 0.015326954710942049, Validation Loss:
0.015928137674927713
Epoch [58/100], Train Loss: 0.015165386936412407, Validation Loss:
0.016021780148148538
Epoch [59/100], Train Loss: 0.015252058609173847, Validation Loss:
0.01605103839188814
Epoch [60/100], Train Loss: 0.015290567149909643, Validation Loss:
0.016029346697032452
Epoch [61/100], Train Loss: 0.015101490828853387, Validation Loss:
0.016072307825088502
Epoch [62/100], Train Loss: 0.015267609188762995, Validation Loss:
0.016086377501487732
Epoch [63/100], Train Loss: 0.015329518856910558, Validation Loss:
0.01607125226408243
Epoch [64/100], Train Loss: 0.015134073793888092, Validation Loss:
0.015989905446767805
Epoch [65/100], Train Loss: 0.01515350963633794, Validation Loss:
0.015959665887057783
Epoch [66/100], Train Loss: 0.015115459449589252, Validation Loss:
0.015946740433573723
Epoch [67/100], Train Loss: 0.015187134679693442, Validation Loss:
0.0160121014341712
Epoch [68/100], Train Loss: 0.014906012596419225, Validation Loss:
0.01602654866874218
Epoch [69/100], Train Loss: 0.015153917125784434, Validation Loss:
0.015979383438825608
Epoch [70/100], Train Loss: 0.015020310090711484, Validation Loss:
```

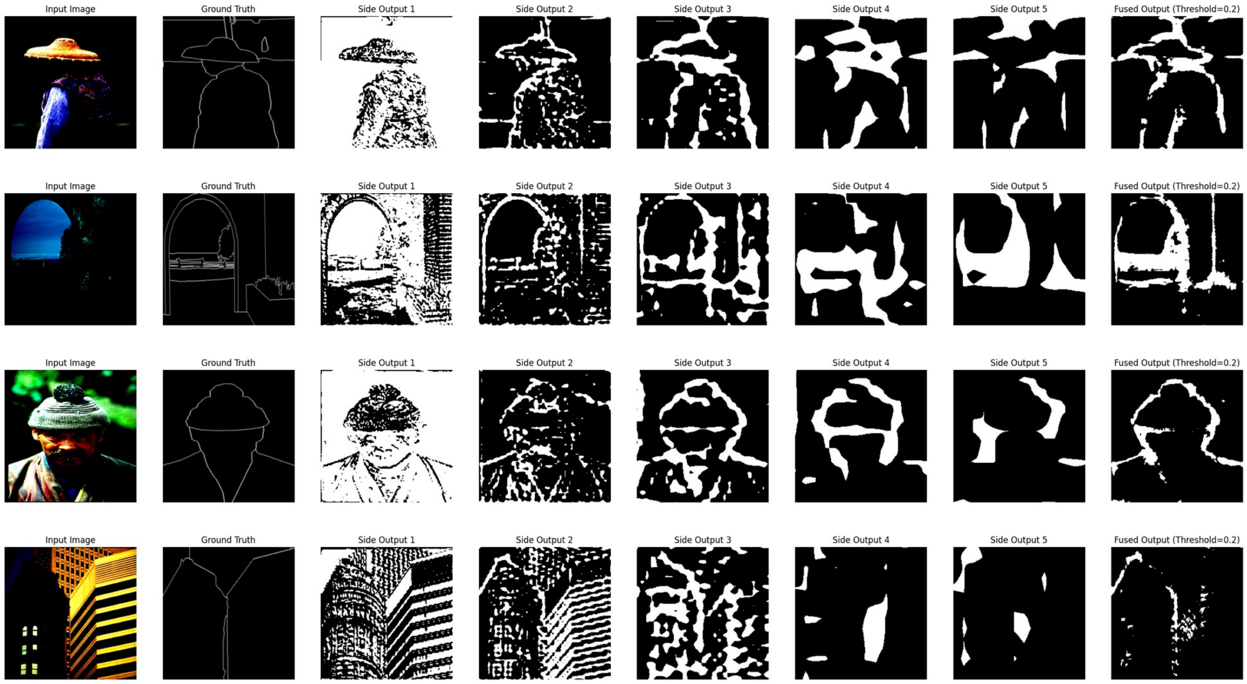
```
0.016026523895561695
Epoch [71/100], Train Loss: 0.015014579519629478, Validation Loss:
0.016044649742543698
Epoch [72/100], Train Loss: 0.015026144325160064, Validation Loss:
0.015964733846485616
Epoch [73/100], Train Loss: 0.014924157697420854, Validation Loss:
0.015988693032413723
Epoch [74/100], Train Loss: 0.014901285489591269, Validation Loss:
0.015954691916704178
Epoch [75/100], Train Loss: 0.014875972571854409, Validation Loss:
0.015970415361225607
Epoch [76/100], Train Loss: 0.015179975101580987, Validation Loss:
0.016086943298578262
Epoch [77/100], Train Loss: 0.014874403579876972, Validation Loss:
0.01620185747742653
Epoch [78/100], Train Loss: 0.014907313654055962, Validation Loss:
0.015994866825640202
Epoch [79/100], Train Loss: 0.01485085766762495, Validation Loss:
0.01599116899073124
Epoch [80/100], Train Loss: 0.015001015522732185, Validation Loss:
0.015999304577708245
Epoch [81/100], Train Loss: 0.014929772068101626, Validation Loss:
0.015986501798033716
Epoch [82/100], Train Loss: 0.01490200597506303, Validation Loss:
0.016038262061774732
Epoch [83/100], Train Loss: 0.014852609557028, Validation Loss:
0.016072744429111482
Epoch [84/100], Train Loss: 0.014885826107974235, Validation Loss:
0.015966923832893373
Epoch [85/100], Train Loss: 0.014831163347340547, Validation Loss:
0.01602835051715374
Epoch [86/100], Train Loss: 0.014833487641925994, Validation Loss:
0.016092713922262192
Epoch [87/100], Train Loss: 0.01498395691697414, Validation Loss:
0.01614954527467489
Epoch [88/100], Train Loss: 0.014648794626387266, Validation Loss:
0.01615835156291723
Epoch [89/100], Train Loss: 0.014733455430429716, Validation Loss:
0.016086763776838778
Epoch [90/100], Train Loss: 0.014906630278206788, Validation Loss:
0.016126020848751067
Epoch [91/100], Train Loss: 0.014771066749325166, Validation Loss:
0.016181897073984146
Epoch [92/100], Train Loss: 0.01488566778313655, Validation Loss:
0.016126633808016776
Epoch [93/100], Train Loss: 0.014842775721962635, Validation Loss:
0.016095777861773967
Epoch [94/100], Train Loss: 0.014773349779156538, Validation Loss:
0.016132297366857527
```

```
Epoch [95/100], Train Loss: 0.01483814067278917, Validation Loss: 
0.016180966906249525
Epoch [96/100], Train Loss: 0.014705997676803516, Validation Loss: 
0.01614089999347925
Epoch [97/100], Train Loss: 0.014643971163492937, Validation Loss: 
0.01609198532998562
Epoch [98/100], Train Loss: 0.014570225173464188, Validation Loss: 
0.016242030449211596
Epoch [99/100], Train Loss: 0.014760510781063484, Validation Loss: 
0.01618360314518213
Epoch [100/100], Train Loss: 0.014636520941097002, Validation Loss: 
0.016124121285974978
```

```
test_dataset = BSDS500(image_dir='archive/images/test', 
edge_dir='archive/ground_truth_boundaries/test',
                        transform=vgg_transform, 
edge_transform=edge_transform)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)

# Function to plot results

def plot_results(model, dataloader, threshold=0.25, device='cpu', 
num_batches=2):
    model.eval()
    batch_count = 0

    with torch.no_grad():
        for images, edges in dataloader:
            images = images.to(device)
            edges = edges.unsqueeze(1).to(device)

            side_outputs = model(images)  # Model returns multiple 
outputs
            probs = [torch.sigmoid(out) for out in side_outputs]  # 
Convert each output to probabilities
            predictions = [(p > threshold).float() for p in probs]  # 
Apply threshold

            for i in range(len(images)):
                num_outputs = len(predictions)

                plt.figure(figsize=(4 * (num_outputs + 2), 4))  # 
Dynamic figure size

                # Plot Input Image
                plt.subplot(1, num_outputs + 2, 1)
                plt.imshow(images[i].cpu().permute(1, 2, 0))
                plt.title("Input Image")
                plt.axis('off')
```

```
                # Plot Ground Truth
                plt.subplot(1, num_outputs + 2, 2)
                plt.imshow(edges[i].cpu().squeeze(), cmap='gray')
                plt.title("Ground Truth")
                plt.axis('off')

                # Plot Side Outputs
                for j in range(num_outputs):
                    plt.subplot(1, num_outputs + 2, j + 3)
                    plt.imshow(predictions[j][i].cpu().squeeze(),
cmap='gray')
                    if j == num_outputs - 1:
                        plt.title(f"Fused Output
(Threshold={threshold})")
                    else:
                        plt.title(f"Side Output {j + 1}")
                    plt.axis('off')

                plt.show()

            batch_count += 1
            if batch_count >= num_batches:
                break

plot_results(model, test_loader, threshold=0.2, device='mps',
num_batches=2)
```




Input Image
Ground Truth
Side Output 1
Side Output 2
Side Output 3
Side Output 4
Side Output 5
Fused Output (Threshold=0.2)