- Satheesh D M
- MA24M023
- Task 2 - CNN edge detector

```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

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

```python
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)

# CNN architecture
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(16, 1, kernel_size=3, padding=1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.conv3(x)
        return x

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)
```

```python
        return loss

def train_and_validate(model, train_loader, val_loader, criterion, 
optimizer, num_epochs=100):
    # Check and set device
    if torch.backends.mps.is_available():
        device = torch.device('mps')
    elif torch.cuda.is_available():
        device = torch.device('cuda')
    else:
        device = torch.device('cpu')

    model.to(device)

    os.makedirs('checkpoints', exist_ok=True)
    csv_path = os.path.join('checkpoints', 'CNN.csv')

    # Create CSV and write headers
    with open(csv_path, mode='w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(["Epoch", "Train Loss", "Validation Loss"])

    train_losses = []
    val_losses = []

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

            optimizer.zero_grad()
            outputs = model(images)

            loss = criterion(outputs, edges)
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

        train_loss = epoch_loss / len(train_loader)
        train_losses.append(train_loss)

        # 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)

                outputs = model(images)
                loss = criterion(outputs, edges)
                val_loss += loss.item()

        val_loss /= len(val_loader)
        val_losses.append(val_loss)

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

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

    return train_losses, val_losses

import torch.optim as optim
# Initialize model, criterion, and optimizer
model = CNN()
criterion = BalancedBCEWithLogitsLoss()
lrate = 0.001
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)
```

```
Epoch [1/100], Train Loss: 0.023817852282753356, Validation Loss:
0.021473737098276616
Epoch [2/100], Train Loss: 0.019473835467719115, Validation Loss:
0.017515031285583973
Epoch [3/100], Train Loss: 0.017447291515194453, Validation Loss:
0.016923963166773318
Epoch [4/100], Train Loss: 0.01712862430856778, Validation Loss:
0.01670920003205538
Epoch [5/100], Train Loss: 0.016703496758754436, Validation Loss:
0.016500063240528107
Epoch [6/100], Train Loss: 0.01682287325652746, Validation Loss:
0.0164274213463068
Epoch [7/100], Train Loss: 0.01646937050211888, Validation Loss:
0.016339521631598474
Epoch [8/100], Train Loss: 0.016548657646546, Validation Loss:
0.016329772770404816
Epoch [9/100], Train Loss: 0.01638561594658173, Validation Loss:
0.01625573992729187
Epoch [10/100], Train Loss: 0.0162973736341183, Validation Loss:
0.01620705094188452
```

```
Epoch [11/100], Train Loss: 0.016316692631405134, Validation Loss:
0.016208656951785087
Epoch [12/100], Train Loss: 0.016288446835600413, Validation Loss:
0.01614197015762329
Epoch [13/100], Train Loss: 0.016226999963132236, Validation Loss:
0.01612869095057249
Epoch [14/100], Train Loss: 0.016429825614278134, Validation Loss:
0.016108295954763888
Epoch [15/100], Train Loss: 0.01638103197686947, Validation Loss:
0.016081939190626143
Epoch [16/100], Train Loss: 0.016332405261122264, Validation Loss:
0.016048835515975954
Epoch [17/100], Train Loss: 0.01614827920611088, Validation Loss:
0.016019237525761128
Epoch [18/100], Train Loss: 0.016064670510016955, Validation Loss:
0.0159551751986146
Epoch [19/100], Train Loss: 0.016004170004564982, Validation Loss:
0.015925597958266736
Epoch [20/100], Train Loss: 0.015909874310287144, Validation Loss:
0.01584805816411972
Epoch [21/100], Train Loss: 0.01598334699296034, Validation Loss:
0.015752094723284246
Epoch [22/100], Train Loss: 0.015840746605625518, Validation Loss:
0.01564296890050173
Epoch [23/100], Train Loss: 0.0157358986683763, Validation Loss:
0.015566463544964791
Epoch [24/100], Train Loss: 0.015630358256972753, Validation Loss:
0.015474905259907246
Epoch [25/100], Train Loss: 0.015715579932125714, Validation Loss:
0.015452549420297145
Epoch [26/100], Train Loss: 0.015539485173156628, Validation Loss:
0.015443294681608677
Epoch [27/100], Train Loss: 0.01534146500321535, Validation Loss:
0.015391964875161647
Epoch [28/100], Train Loss: 0.015484591946005821, Validation Loss:
0.015379902981221676
Epoch [29/100], Train Loss: 0.01544639733261787, Validation Loss:
0.015336212627589703
Epoch [30/100], Train Loss: 0.015492403091719517, Validation Loss:
0.01531127255409956
Epoch [31/100], Train Loss: 0.015419084077271132, Validation Loss:
0.015297607518732548
Epoch [32/100], Train Loss: 0.015488662542058872, Validation Loss:
0.01534581396728754
Epoch [33/100], Train Loss: 0.015524831528847035, Validation Loss:
0.015354042872786522
Epoch [34/100], Train Loss: 0.015388890312841305, Validation Loss:
0.015341678149998188
Epoch [35/100], Train Loss: 0.015393484742022477, Validation Loss:
```

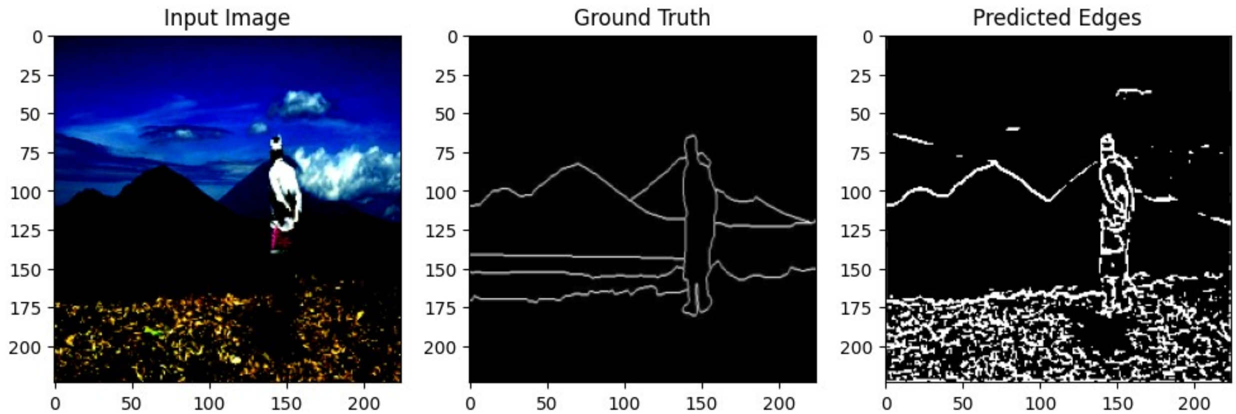
```
0.015225663110613822
Epoch [36/100], Train Loss: 0.015172508903420888, Validation Loss:
0.01522511500865221
Epoch [37/100], Train Loss: 0.015131439153964702, Validation Loss:
0.015238400921225548
Epoch [38/100], Train Loss: 0.015361961980278675, Validation Loss:
0.015211970694363117
Epoch [39/100], Train Loss: 0.015400938044946928, Validation Loss:
0.01519808392971754
Epoch [40/100], Train Loss: 0.015214467851015238, Validation Loss:
0.015156145319342612
Epoch [41/100], Train Loss: 0.015343423717870163, Validation Loss:
0.015146382190287112
Epoch [42/100], Train Loss: 0.015052875074056478, Validation Loss:
0.015153623074293137
Epoch [43/100], Train Loss: 0.015269979261434995, Validation Loss:
0.015134221520274877
Epoch [44/100], Train Loss: 0.015143345611599775, Validation Loss:
0.015169027335941792
Epoch [45/100], Train Loss: 0.015262635281452766, Validation Loss:
0.015116762220859527
Epoch [46/100], Train Loss: 0.015216608746693684, Validation Loss:
0.015131330527365207
Epoch [47/100], Train Loss: 0.015091107943310188, Validation Loss:
0.015116516537964343
Epoch [48/100], Train Loss: 0.015092464020619025, Validation Loss:
0.015060413591563702
Epoch [49/100], Train Loss: 0.015265634999825405, Validation Loss:
0.015117870215326547
Epoch [50/100], Train Loss: 0.015250524434332665, Validation Loss:
0.015060717277228831
Epoch [51/100], Train Loss: 0.015135121245223742, Validation Loss:
0.015042576640844345
Epoch [52/100], Train Loss: 0.015014139792093864, Validation Loss:
0.015052511524409055
Epoch [53/100], Train Loss: 0.0151136159323729, Validation Loss:
0.015041941367089749
Epoch [54/100], Train Loss: 0.015254656139474649, Validation Loss:
0.015031515061855317
Epoch [55/100], Train Loss: 0.0149655260432225, Validation Loss:
0.015063378028571606
Epoch [56/100], Train Loss: 0.015003190519144902, Validation Loss:
0.015056046806275845
Epoch [57/100], Train Loss: 0.015119812451303005, Validation Loss:
0.015009011328220367
Epoch [58/100], Train Loss: 0.014985738465419183, Validation Loss:
0.015071835219860077
Epoch [59/100], Train Loss: 0.015087751814952264, Validation Loss:
0.01522370781749487
```

```
Epoch [60/100], Train Loss: 0.015158127635144271, Validation Loss:
0.015009035989642143
Epoch [61/100], Train Loss: 0.01494905173491973, Validation Loss:
0.015016407109797
Epoch [62/100], Train Loss: 0.015116598958579393, Validation Loss:
0.014990765489637852
Epoch [63/100], Train Loss: 0.015190984982137497, Validation Loss:
0.014966818131506443
Epoch [64/100], Train Loss: 0.015018194555663146, Validation Loss:
0.015055712275207044
Epoch [65/100], Train Loss: 0.015110536765020628, Validation Loss:
0.015074099712073803
Epoch [66/100], Train Loss: 0.015064169366199236, Validation Loss:
0.015185823775827884
Epoch [67/100], Train Loss: 0.01515358378394292, Validation Loss:
0.01503177685663104
Epoch [68/100], Train Loss: 0.014850425605590526, Validation Loss:
0.015057947896420955
Epoch [69/100], Train Loss: 0.015078212421100873, Validation Loss:
0.015052193477749824
Epoch [70/100], Train Loss: 0.014985169355685894, Validation Loss:
0.014998784437775612
Epoch [71/100], Train Loss: 0.01500808447599411, Validation Loss:
0.014985953867435455
Epoch [72/100], Train Loss: 0.01498177031484934, Validation Loss:
0.01495614916086197
Epoch [73/100], Train Loss: 0.014911653402333077, Validation Loss:
0.015000936705619097
Epoch [74/100], Train Loss: 0.01492662656192596, Validation Loss:
0.014976143091917037
Epoch [75/100], Train Loss: 0.014887638533344636, Validation Loss:
0.014975239410996438
Epoch [76/100], Train Loss: 0.015157792072456617, Validation Loss:
0.014954611994326114
Epoch [77/100], Train Loss: 0.014898489659222273, Validation Loss:
0.014960029497742652
Epoch [78/100], Train Loss: 0.014941228768573357, Validation Loss:
0.014960724376142025
Epoch [79/100], Train Loss: 0.014920461206482006, Validation Loss:
0.014966689832508564
Epoch [80/100], Train Loss: 0.015059074028753318, Validation Loss:
0.014934652969241142
Epoch [81/100], Train Loss: 0.014998350005883437, Validation Loss:
0.01495396412909031
Epoch [82/100], Train Loss: 0.014997432151666054, Validation Loss:
0.014949525371193886
Epoch [83/100], Train Loss: 0.01495068484487442, Validation Loss:
0.014948838204145432
Epoch [84/100], Train Loss: 0.014980746074937858, Validation Loss:
```

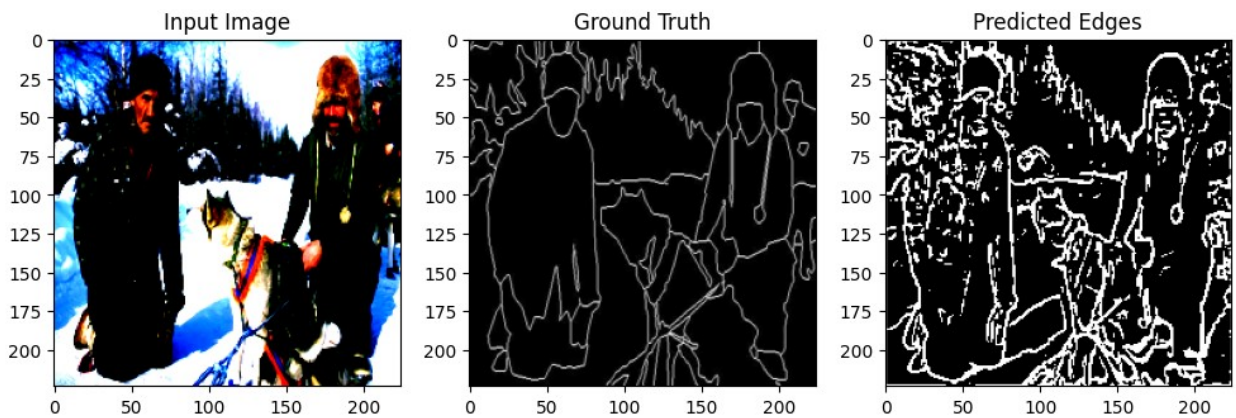
```
0.014943993873894215
Epoch [85/100], Train Loss: 0.01492649202163403, Validation Loss:
0.01493320770561695
Epoch [86/100], Train Loss: 0.0149602061853959, Validation Loss:
0.01494730968028307
Epoch [87/100], Train Loss: 0.015077161459395519, Validation Loss:
0.014904944375157356
Epoch [88/100], Train Loss: 0.014745652460708069, Validation Loss:
0.014915563575923442
Epoch [89/100], Train Loss: 0.01487395905244809, Validation Loss:
0.01494040623307228
Epoch [90/100], Train Loss: 0.015048407161465058, Validation Loss:
0.014906357638537884
Epoch [91/100], Train Loss: 0.014939183870760294, Validation Loss:
0.014951617419719697
Epoch [92/100], Train Loss: 0.015062963590025902, Validation Loss:
0.014915199354290962
Epoch [93/100], Train Loss: 0.015021162680708446, Validation Loss:
0.014941221214830875
Epoch [94/100], Train Loss: 0.014964375931483049, Validation Loss:
0.0149446789175272
Epoch [95/100], Train Loss: 0.015051183577340383, Validation Loss:
0.014924528449773789
Epoch [96/100], Train Loss: 0.014910211213506185, Validation Loss:
0.014991236068308354
Epoch [97/100], Train Loss: 0.014853426900047522, Validation Loss:
0.014893348999321461
Epoch [98/100], Train Loss: 0.014782966616062017, Validation Loss:
0.014935400113463402
Epoch [99/100], Train Loss: 0.014997906266496731, Validation Loss:
0.014943513311445713
Epoch [100/100], Train Loss: 0.014873391733719753, Validation Loss:
0.014898669607937336

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=True)

def plot_results(model, dataloader, threshold=0.25, device='mps',
num_batches=2):
    model.eval()  # Set the model to evaluation mode
    batch_count = 0

    with torch.no_grad():  # Disable gradient calculation for
inference
        for images, edges in dataloader:
            images = images.to(device)
            edges = edges.unsqueeze(1).to(device)  # Make sure the
```

```
ground truth has the correct shape

            # Get the model's output and apply Sigmoid activation
            outputs = model(images)
            outputs = torch.sigmoid(outputs)  # Apply Sigmoid

            # Apply thresholding to get binary predictions
            predictions = (outputs > threshold).float()

            # Visualize the results for the current batch
            for i in range(len(images)):
                plt.figure(figsize=(12, 4))

                # Display input image
                plt.subplot(1, 3, 1)
                plt.imshow(images[i].cpu().permute(1, 2, 0))  #
Convert to HxWxC format for displaying
                plt.title("Input Image")

                # Display ground truth
                plt.subplot(1, 3, 2)
                plt.imshow(edges[i].cpu().squeeze(), cmap='gray')  #
Remove channel dimension for grayscale image
                plt.title("Ground Truth")

                # Display predicted edges
                plt.subplot(1, 3, 3)
                plt.imshow(predictions[i].cpu().squeeze(),
cmap='gray')  # Remove channel dimension
                plt.title("Predicted Edges")

                plt.show()

            batch_count += 1
            if batch_count >= num_batches:
                break

plot_results(model, test_loader, threshold=0.22, device='mps',
num_batches=2)
```

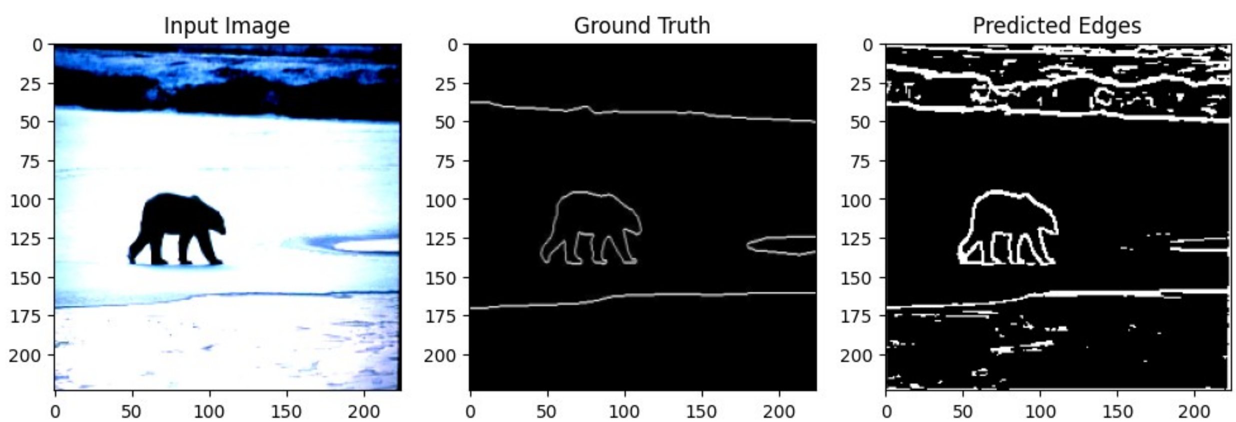
```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
1.9466565..2.4285715].
```



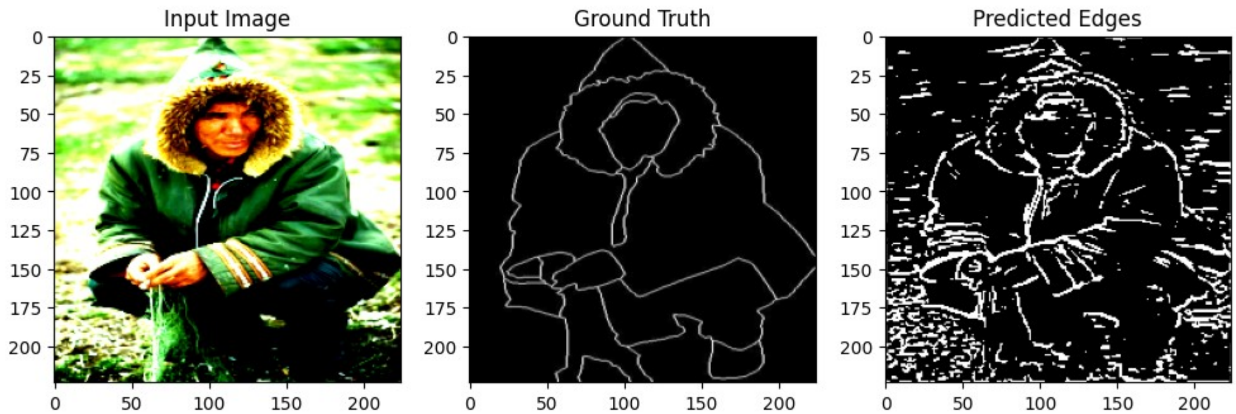

```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
1.8952821..2.64].
```



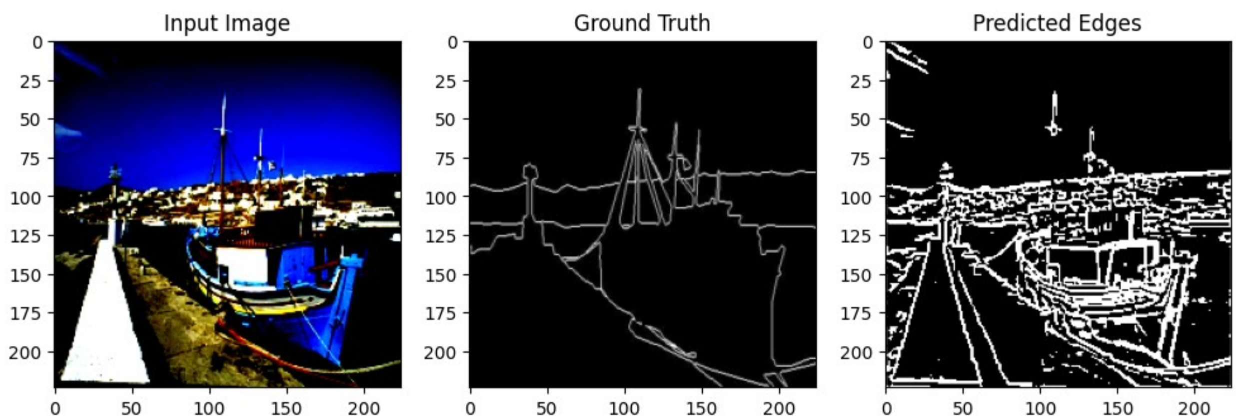

```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
1.7069099..2.5179958].
```



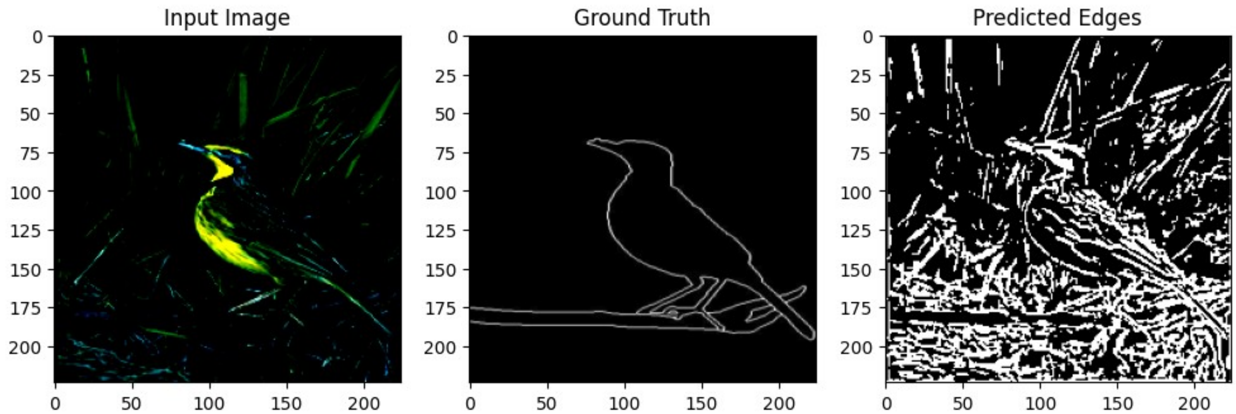
```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
1.4329139..2.6051416].
```



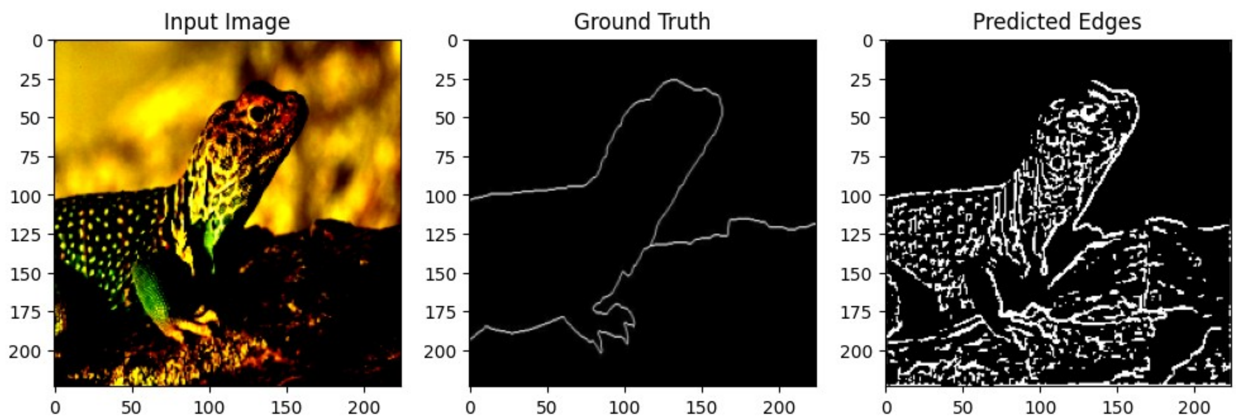

```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
2.0665298..2.4110641].
```




```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
2.117904..2.129035].
```




```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
1.9295317..2.2146587].
```




```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
2.0322802..2.64].
```