Image Segmentation

Introduction

Image segmentation is a fundamental task in computer vision that involves partitioning an image into meaningful regions, often corresponding to objects or specific structures. In this Task, we work with the **CAMVid dataset**, a labeled dataset for semantic segmentation tasks.

1. Download the CAMVid dataset.

(a) Dataset Preparation

a.1 Dataset Description

The **CAMVid dataset** consists of urban street scenes, with each pixel labeled according to its corresponding class. The dataset includes training, validation, and test images along with corresponding segmentation masks. The original image resolution is **(960 × 720)**.

a.2 Data Loading and Preprocessing

To ensure proper model input, we apply the following preprocessing steps:

- **Resizing** images to **(480 × 360)**.
- **Normalization** using the mean `[0.485, 0.456, 0.406]` and standard deviation `[0.229, 0.224, 0.225]`.

```
# Define transforms
transform = transforms.Compose([
    transforms.Resize((360, 480)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

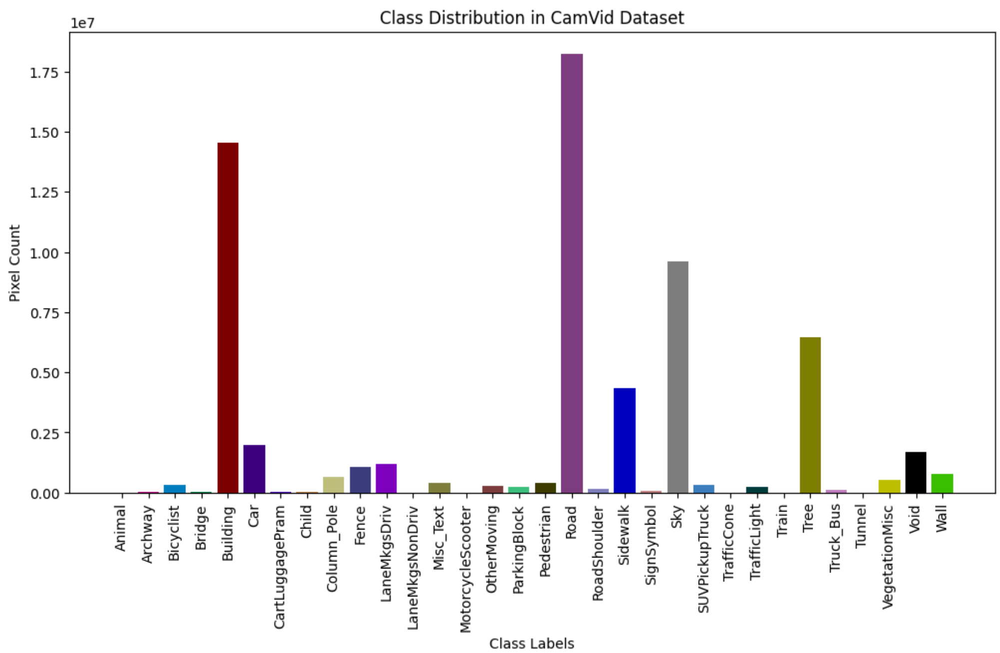
- **Encoding segmentation masks**, converting RGB masks into label indices.
- A custom PyTorch **Dataloader** was implemented to handle these transformations efficiently with **batch size: 10**

```
Train Dataset Size: 369
Test Dataset Size: 232
```

```python
class CamVidDataset(Dataset):
    def __init__(self, image_dir, mask_dir, class_dict_path, transform=None, preload=True):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.preload = preload
        self.class_dict = pd.read_csv(class_dict_path)
        self.images = os.listdir(image_dir)

        # Load class mapping
        self.color_to_label = {
            tuple(self.class_dict.iloc[i, 1:4].astype(int)): i
            for i in range(len(self.class_dict))
        }

        self.mask_transform = transforms.Compose([
            transforms.Resize((360, 480), interpolation=Image.NEAREST)
        ])

        # Preload data into RAM if preload=True
        if self.preload:
            self.preloaded_data = []
            for img_name in self.images:
                self.preloaded_data.append(self.process_image_mask(img_name))

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

    def encode_mask(self, mask):
        mask = np.array(mask, dtype=np.uint8)
        label_mask = np.zeros(mask.shape[:2], dtype=np.int64)
        for color, label in self.color_to_label.items():
            label_mask[(mask == color).all(axis=-1)] = label
        return torch.tensor(label_mask, dtype=torch.long)

    def process_image_mask(self, img_name):
        img_path = os.path.join(self.image_dir, img_name)
        mask_path = os.path.join(self.mask_dir, img_name.replace('.png', '_L.png'))

        image = Image.open(img_path).convert("RGB")
        mask = Image.open(mask_path).convert("RGB")

        if self.transform:
            image = self.transform(image)

        mask = self.mask_transform(mask)
        mask = self.encode_mask(mask)

        return image, mask

    def __getitem__(self, idx):
        return self.preloaded_data[idx] if self.preload else self.process_image_mask(self.images[idx])

    def visualize_sample(self, idx):
        img_path = os.path.join(self.image_dir, self.images[idx])
        mask_path = os.path.join(self.mask_dir, self.images[idx].replace('.png', '_L.png'))

        image = Image.open(img_path).convert("RGB")
        mask = Image.open(mask_path).convert("RGB")

        fig, axes = plt.subplots(1, 2, figsize=(8, 5))
        axes[0].imshow(image)
        axes[0].set_title("Image")
        axes[0].axis("off")

        axes[1].imshow(mask)
        axes[1].set_title("Mask")
        axes[1].axis("off")

        plt.show()
```

(b) Class Distribution Visualization

To analyze class frequency in the dataset, we computed the pixel-wise distribution of class labels across the training images.



(c) Image and Mask Visualization

2. [SegNet Encoder-Decoder]

(a) Implementation and Training

a.1 Architecture

The SegNet was implemented by following the architecture specified in the `model classes.py` file. The encoder consists of five stages, each corresponding to an encoding stage. The layers in the decoder mirror the encoder using transposed convolutions, batch normalization, and activation functions.

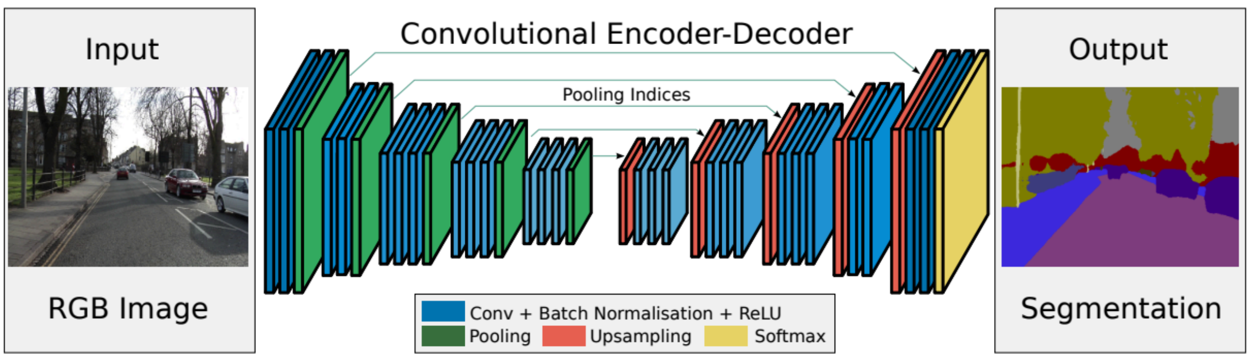
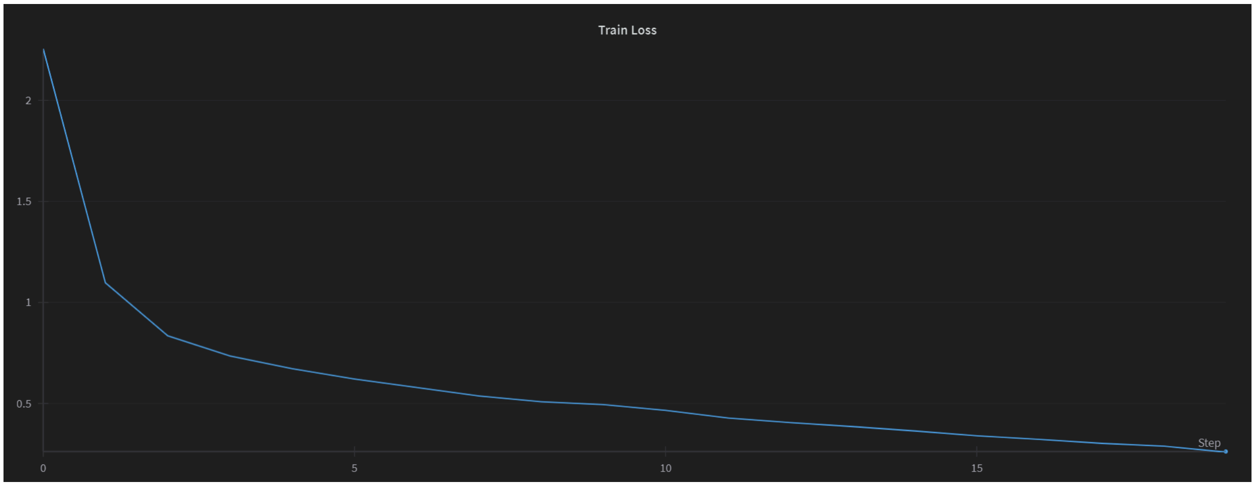


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

a.2 Training Setup

- **Loss Function:** Cross-entropy loss
- **Optimizer:** Adam optimizer
- **Batch Normalization Momentum:** 0.5
- **Learning rate:** 0.0002
- **Epoch:** 20

a.3 Training logs




```
================================ TRAINING segnet =================================
Epoch [1/20] -> Train Loss: 2.2558
Epoch [2/20] -> Train Loss: 1.0993
Epoch [3/20] -> Train Loss: 0.8367
Epoch [4/20] -> Train Loss: 0.7373
Epoch [5/20] -> Train Loss: 0.6747
Epoch [6/20] -> Train Loss: 0.6237
Epoch [7/20] -> Train Loss: 0.5817
Epoch [8/20] -> Train Loss: 0.5400
Epoch [9/20] -> Train Loss: 0.5114
Epoch [10/20] -> Train Loss: 0.4970
Epoch [11/20] -> Train Loss: 0.4687
Epoch [12/20] -> Train Loss: 0.4307
Epoch [13/20] -> Train Loss: 0.4084
Epoch [14/20] -> Train Loss: 0.3887
Epoch [15/20] -> Train Loss: 0.3666
Epoch [16/20] -> Train Loss: 0.3428
Epoch [17/20] -> Train Loss: 0.3251
Epoch [18/20] -> Train Loss: 0.3056
Epoch [19/20] -> Train Loss: 0.2917
Epoch [20/20] -> Train Loss: 0.2621
================================ TRAINING COMPLETED =================================
```

a.4 Model Saving

The SegNet decoder was successfully trained using a pre-trained encoder

SegNet_Encoder is saved in ***encoder_model.pth***

SegNet_Decoder is saved in ***decoder.pth***

(b) Performance Evaluation

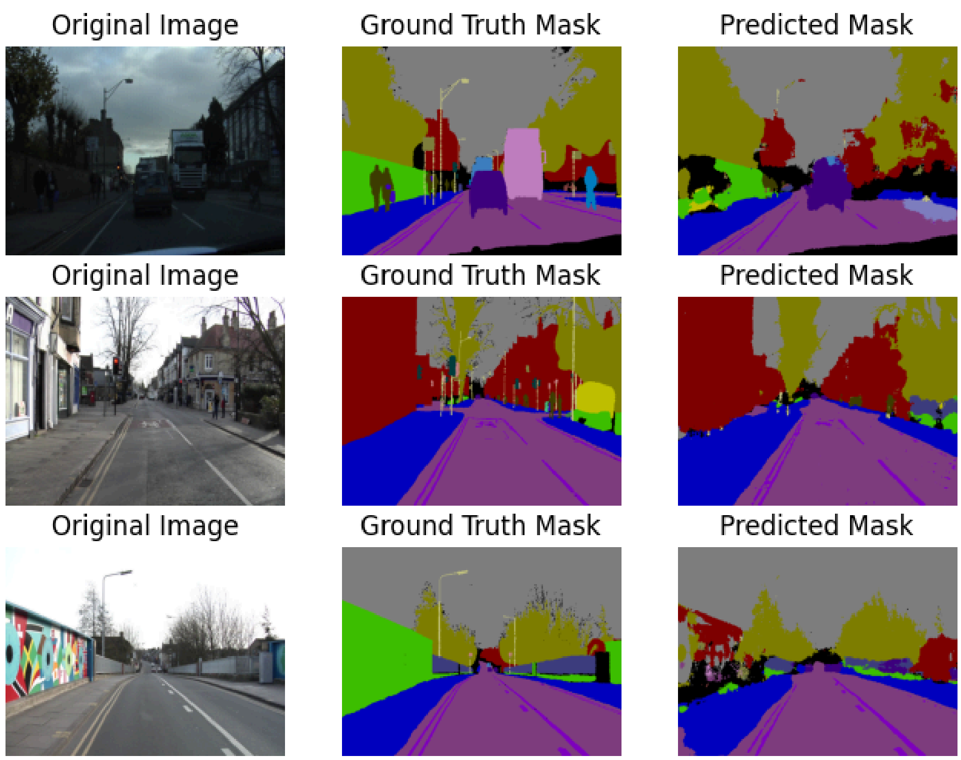
```
Pixel Accuracy: 0.8150
Mean IoU (mIoU): 0.2272

Class-wise Metrics:
Class 0: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 1: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 2: IoU=0.3223, Dice=0.4874, Precision=0.7264, Recall=0.3668
Class 3: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 4: IoU=0.7550, Dice=0.8604, Precision=0.8510, Recall=0.8700
Class 5: IoU=0.6155, Dice=0.7620, Precision=0.7673, Recall=0.7568
Class 6: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 7: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 8: IoU=0.0523, Dice=0.0994, Precision=0.4449, Recall=0.0559
Class 9: IoU=0.2815, Dice=0.4393, Precision=0.4963, Recall=0.3941
Class 10: IoU=0.4408, Dice=0.6119, Precision=0.7384, Recall=0.5223
Class 11: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 12: IoU=0.0317, Dice=0.0615, Precision=0.1046, Recall=0.0436
Class 13: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 14: IoU=0.1922, Dice=0.3225, Precision=0.3902, Recall=0.2748
Class 15: IoU=0.1480, Dice=0.2579, Precision=0.2204, Recall=0.3106
Class 16: IoU=0.1188, Dice=0.2124, Precision=0.3126, Recall=0.1609
Class 17: IoU=0.8758, Dice=0.9338, Precision=0.9376, Recall=0.9300
Class 18: IoU=0.1430, Dice=0.2502, Precision=0.1621, Recall=0.5479
Class 19: IoU=0.6839, Dice=0.8123, Precision=0.8031, Recall=0.8217
Class 20: IoU=0.0052, Dice=0.0104, Precision=0.8651, Recall=0.0052
Class 21: IoU=0.8945, Dice=0.9443, Precision=0.9310, Recall=0.9580
Class 22: IoU=0.0585, Dice=0.1105, Precision=0.1739, Recall=0.0809
Class 23: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 24: IoU=0.1687, Dice=0.2887, Precision=0.6612, Recall=0.1847
Class 25: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 26: IoU=0.6588, Dice=0.7943, Precision=0.7279, Recall=0.8740
Class 27: IoU=0.0624, Dice=0.1175, Precision=0.5183, Recall=0.0663
Class 28: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 29: IoU=0.1131, Dice=0.2033, Precision=0.3519, Recall=0.1429
Class 30: IoU=0.3189, Dice=0.4836, Precision=0.4197, Recall=0.5704
Class 31: IoU=0.3309, Dice=0.4972, Precision=0.5536, Recall=0.4513
```

(c) Visualization and Failure Analysis

c.1 Visualizing IoU ≤ 0.5 Cases

For each class, three images where the $0 < \text{IoU} \leq 0.5$ were selected but for some classes there are no examples which might be because class objects are not present in the test dataset. These images show the predicted masks compared to ground truth.

Example Figures:



c.2 Failure Analysis

The model struggles in the following cases:

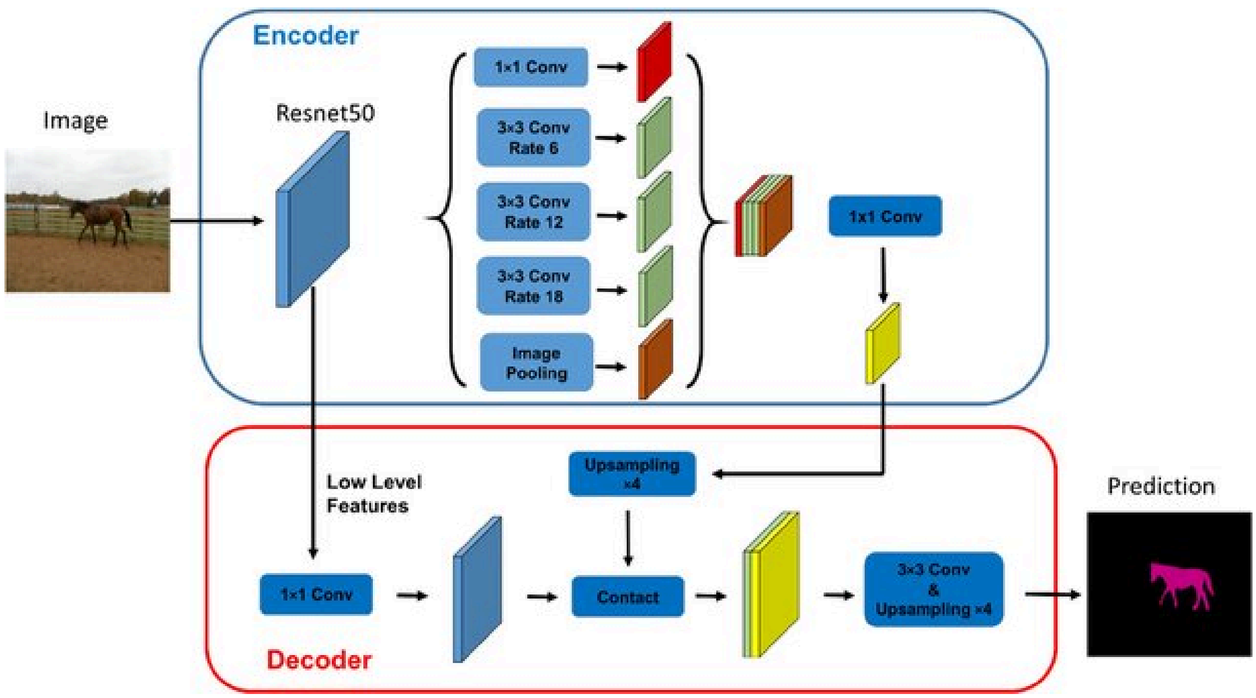
- **Occlusion:** Objects are partially visible.
- **Misclassification:** The model confuses similar-looking objects like bicyclist and pedestrians
- **Environmental Challenges:** Low lighting or complex backgrounds.
- **Small Object Detection:** CamVid contains small, thin objects like poles and signs, which are hard for convolutional layers to segment accurately.
- **Ambiguous Boundaries:** Blurred or unclear boundaries between objects (e.g., road and sidewalk) can confuse the model.
- **SegNet** lacks advanced boundary refinement techniques.

3. DeepLabv3 (ResNet50)

(a) Implementation and Training

a.1 Architecture

DeepLabV3, pre-trained on the Pascal VOC dataset, was fine-tuned for CamVid segmentation. The classifier was modified to predict 32 classes.

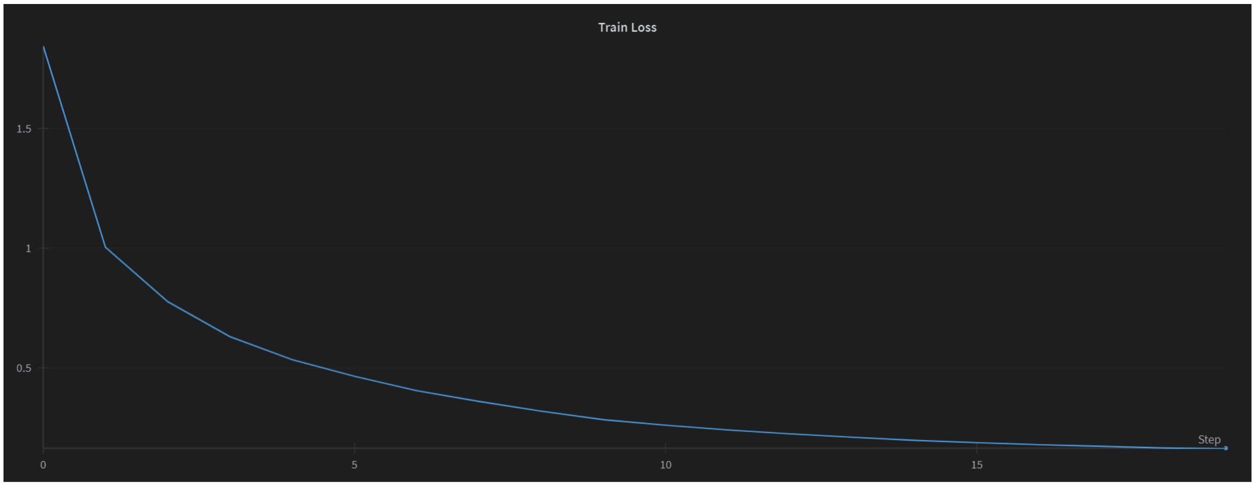


a.2 Training Setup

- **Loss Function:** Cross-entropy loss
- **Optimizer:** Adam optimizer
- **Batch Normalization Momentum:** 0.5
- **Learning rate:** 0.0002
- **Epoch:** 20

deeplabv3	Finished	Add notes	harshu	25m ago	14m 24s	-	20	0.0002	0.16492

a.3 Training logs




```
================================ TRAINING deeplabv3 ================================
Epoch [1/20] -> Train Loss: 1.8444
Epoch [2/20] -> Train Loss: 1.0057
Epoch [3/20] -> Train Loss: 0.7784
Epoch [4/20] -> Train Loss: 0.6327
Epoch [5/20] -> Train Loss: 0.5366
Epoch [6/20] -> Train Loss: 0.4672
Epoch [7/20] -> Train Loss: 0.4074
Epoch [8/20] -> Train Loss: 0.3625
Epoch [9/20] -> Train Loss: 0.3218
Epoch [10/20] -> Train Loss: 0.2863
Epoch [11/20] -> Train Loss: 0.2632
Epoch [12/20] -> Train Loss: 0.2437
Epoch [13/20] -> Train Loss: 0.2273
Epoch [14/20] -> Train Loss: 0.2134
Epoch [15/20] -> Train Loss: 0.2003
Epoch [16/20] -> Train Loss: 0.1907
Epoch [17/20] -> Train Loss: 0.1823
Epoch [18/20] -> Train Loss: 0.1756
Epoch [19/20] -> Train Loss: 0.1684
Epoch [20/20] -> Train Loss: 0.1649
================================ TRAINING COMPLETED ================================
```

a.4 Model Saving

The DeepLabv3 decoder was successfully fine tuned on CamVid

Model's state dict saved as ***deeplabv3.pth***

(b) Performance Evaluation

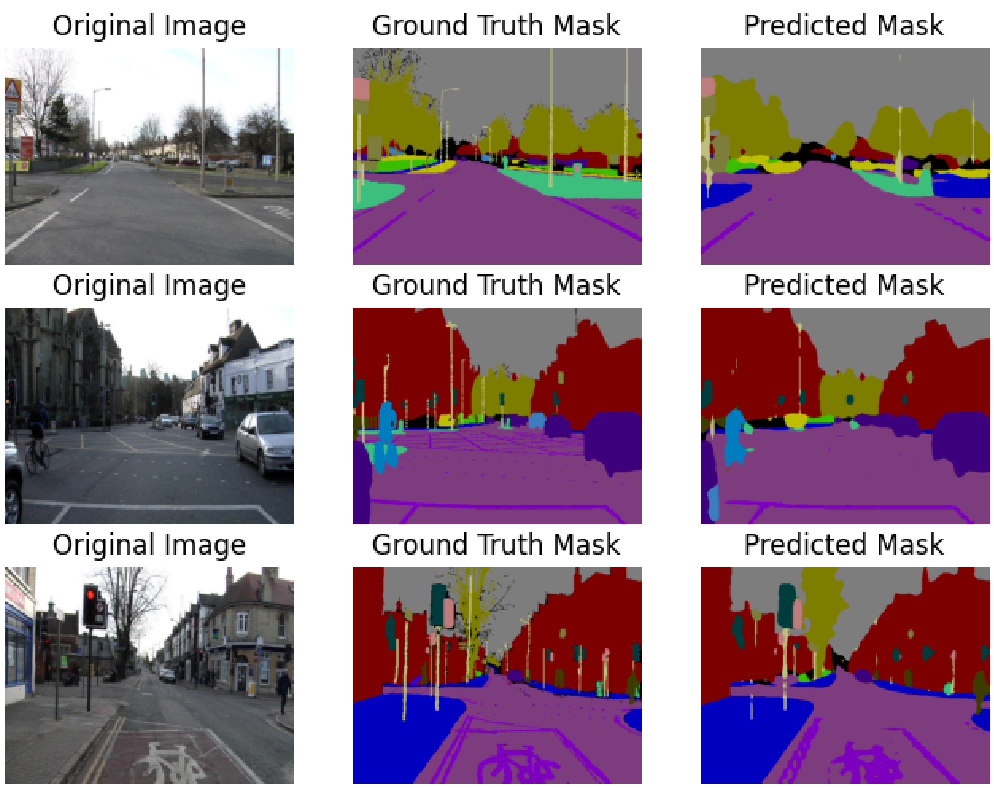
```
Pixel Accuracy: 0.8817
Mean IoU (mIoU): 0.3659

Class-wise Metrics:
Class 0: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 1: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 2: IoU=0.4604, Dice=0.6305, Precision=0.9345, Recall=0.4758
Class 3: IoU=0.2485, Dice=0.3981, Precision=0.9782, Recall=0.2499
Class 4: IoU=0.8431, Dice=0.9149, Precision=0.8773, Recall=0.9559
Class 5: IoU=0.8085, Dice=0.8941, Precision=0.8466, Recall=0.9473
Class 6: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 7: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 8: IoU=0.1434, Dice=0.2508, Precision=0.5185, Recall=0.1654
Class 9: IoU=0.5299, Dice=0.6928, Precision=0.6934, Recall=0.6921
Class 10: IoU=0.3887, Dice=0.5598, Precision=0.7211, Recall=0.4575
Class 11: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 12: IoU=0.3540, Dice=0.5229, Precision=0.5947, Recall=0.4667
Class 13: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 14: IoU=0.4276, Dice=0.5990, Precision=0.5526, Recall=0.6540
Class 15: IoU=0.3900, Dice=0.5611, Precision=0.8549, Recall=0.4176
Class 16: IoU=0.4044, Dice=0.5760, Precision=0.5887, Recall=0.5637
Class 17: IoU=0.9145, Dice=0.9553, Precision=0.9463, Recall=0.9646
Class 18: IoU=0.5162, Dice=0.6809, Precision=0.9262, Recall=0.5384
Class 19: IoU=0.7920, Dice=0.8839, Precision=0.8349, Recall=0.9390
Class 20: IoU=0.2989, Dice=0.4602, Precision=0.8748, Recall=0.3123
Class 21: IoU=0.9137, Dice=0.9549, Precision=0.9401, Recall=0.9702
Class 22: IoU=0.2391, Dice=0.3860, Precision=0.4827, Recall=0.3215
Class 23: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 24: IoU=0.5217, Dice=0.6857, Precision=0.7886, Recall=0.6065
Class 25: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 26: IoU=0.7705, Dice=0.8704, Precision=0.8720, Recall=0.8688
Class 27: IoU=0.1457, Dice=0.2544, Precision=0.7303, Recall=0.1540
Class 28: IoU=0.0000, Dice=0.0000, Precision=0.0000, Recall=0.0000
Class 29: IoU=0.5709, Dice=0.7268, Precision=0.7200, Recall=0.7338
Class 30: IoU=0.4720, Dice=0.6413, Precision=0.6936, Recall=0.5964
Class 31: IoU=0.5560, Dice=0.7147, Precision=0.8131, Recall=0.6375
```

(c) Visualization and Failure Analysis

c.1 Visualizing IoU ≤ 0.5 Cases

For each class, three images where the $0 < IoU \leq 0.5$ were selected but for some classes there are no examples which might be because class objects are not present in the test dataset. These images show the predicted masks compared to ground truth.

Example Figures:



c.2 Failure Analysis

Overall DeepLabv3 is performing better than segnet but it also struggles in some cases:

- **Occlusion:** Objects are partially visible.
- **Misclassification:** The model confuses similar-looking objects like bicyclist and pedestrians
- **Environmental Challenges:** Low lighting or complex backgrounds.
- **DeepLabV3** handles boundaries better but may still struggle with overlapping objects.