CV Assignment 1

CSE 344 | Winter 25'
Dr. Saket Anand
Submitted By: Rahul Oberoi (2021555)

Theory Questions:

Q1

a) Since y is a one hot vector it will have 1 where it is the correct class and 0 everywhere else.

By adding label smoothing it becomes $1-\epsilon+\epsilon/K$ for the true class and $\epsilon+\epsilon/K$ for the incorrect class.

$$\hat{y}_i = \begin{cases} 1-\epsilon+\epsilon/K & i=k \quad k \text{ is true label} \\ \epsilon+\epsilon/K & i\neq k \end{cases}$$

$$H(\hat{y}, q) = -\sum_{i=1}^{K} \hat{y}_i \log q_i$$

$$H(\hat{y}, q) = -\left[(1-\epsilon+\epsilon/K) \log q_k + \sum_{i\neq k} \epsilon/K \log q_i \right]$$

Correct Class

$$H(\hat{y}, q) = -\left[(1-\epsilon+\epsilon/K) \log q_k + \epsilon/K \sum_{i\neq k} \log q_i \right]$$

$$\sum_{i=1}^{K} \log q_i = \log q_k + \sum_{i\neq k} \log q_i \quad [\text{total log probability}]$$

$$\sum_{i\neq k} \log q_i = \sum_{i=1}^{K} \log q_i - \log q_k$$

$$H(\hat{y}, q) = -\left[(1-\epsilon+\epsilon/K) \log q_k + \epsilon/K \left(\sum_{i=1}^{K} \log q_i - \log q_k \right) \right]$$

$$H(\hat{y}, q) = -\left[(1-\epsilon+\epsilon/K) \log q_k + \epsilon/K \sum_{i=1}^{K} \log q_i - \epsilon/K \log q_k \right]$$

$$H(\hat{y}, q) = -\left[(1-\epsilon+\epsilon/K-\epsilon/K)\log q_k + \epsilon/K \sum_{i=1}^{K} \log q_i\right]$$

$$H(\hat{y}, q) = -(1-\epsilon)\log q_k - \frac{\epsilon}{K}\sum_{i=1}^{K}\log q_i$$

b) Effect on loss:

When our model is prone to overfitting label smoothing can act as regularizer by assigning small probabilities to the labels with 0 probability.

It helps in reducing the model's overconfidence by assigning some probabilities to every class, even those that are not true.

Interpretation:
Label smoothing effectively reflects the uncertainity (like the real world data), we may not always be 100% certain. ∴ our model can generalize better.

Q2

a) $H(p, q) = E_p(-\log q(x))$

For the gaussian distribution: $H(p, q) = \int_{-\infty}^{\infty} p(x)\,(-\log(q(x)))\,dx$ —①

$p(x) = N(\mu_p, \sigma_p^2)$ $q(x) = N(\mu_q, \sigma_q^2)$

b) $p(x) = \frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(x-\mu_p)^2}{2\sigma_p^2}\right)$

$q(x) = \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(x-\mu_q)^2}{2\sigma_q^2}\right)$

Taking log of $q(x)$

$\log(q(x)) = -\log\left(\sqrt{2\pi\sigma_q^2}\right) - \frac{(x-\mu_q)^2}{2\sigma_q^2}$

$-\log(q(x)) = \log\left(\sqrt{2\pi\sigma_q^2}\right) + \frac{(x-\mu_q)^2}{2\sigma_q^2}$

Replacing in ①

$$H(p,q) = \int_{-\infty}^{\infty} p(x)\left[\log\left(\sqrt{2\pi\sigma_q^2}\right) + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right] dx$$

$$\Rightarrow \underbrace{\int_{-\infty}^{\infty} p(x)\log\left(\sqrt{2\pi\sigma_q^2}\right)}_{\text{Solving this first}} + \int_{-\infty}^{\infty} p(x)\frac{(x-\mu_q)^2}{2\sigma_q^2}$$

$\Rightarrow \log\left(\sqrt{2\pi\sigma_q^2}\right)\int_{-\infty}^{\infty} p(x)\,dx$ — probability ∴ integral over the whole domain = 1

$\Rightarrow \log\left(\sqrt{2\pi\sigma_q^2}\right)$

$\Rightarrow$ Solving second integral

$$\Rightarrow \int_{-\infty}^{\infty} p(x) \frac{(x-\mu_q)^2}{2\sigma_q^2} dx \qquad (x-\mu_q)^2 = (x-\mu_p+\mu_p-\mu_q)^2$$

$$\Rightarrow \underbrace{(x-\mu_p)^2}_{\sigma_p^2} + \underbrace{2(x-\mu_p)(\mu_p-\mu_q)}_{\Downarrow 0} + (\mu_p-\mu_q)$$

$$\Rightarrow \int_{-\infty}^{\infty} p(x) \frac{[\sigma_p^2 + (\mu_p-\mu_q)^2]}{2\sigma_q^2} dx$$

(integral of a deviation from the mean over a symmetric distribution)

$$\Rightarrow \frac{[\sigma_p^2 + (\mu_p-\mu_q)^2]}{2\sigma_q^2}$$

$$\Rightarrow \log\left(\sqrt{2\pi\sigma_q^2}\right) + \frac{\sigma_p^2 + (\mu_p-\mu_q)^2}{2\sigma_q^2}$$

$$\Rightarrow \frac{1}{2}\log(2\pi\sigma_q^2) + \frac{\sigma_p^2 + (\mu_p-\mu_q)^2}{2\sigma_q^2} = H(p,q)$$

c) If $\sigma_q = \sigma_p = \sigma$

$$\Rightarrow H(p,q) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{\sigma^2 + (\mu_p-\mu_q)^2}{2\sigma^2}$$

$$\Rightarrow \frac{1}{2}\log(2\pi\sigma^2) + \frac{\sigma^2}{2\sigma^2} + \frac{(\mu_p-\mu_q)^2}{2\sigma^2}$$

$$\Rightarrow \frac{1}{2}\left(\log(2\pi\sigma^2) + 1 + \frac{(\mu_p-\mu_q)^2}{\sigma^2}\right)$$

Q3 a) Kernel size = k

Dilation = n

Layers = l

Using kernel size

1D convolution receptive field = $k + (k-1)(n-1)$

$K = k + (k-1)(n-1)$ [At dilation R and layer n]

$K_{n-1} = k + (k-1)(n-1) - 1$

$= \cancel{k} + k - \cancel{k} - n + \cancel{1} - \cancel{1}$

$= n_n(k-1)$

$K = K_{n-1} + n_{n-1}(k-1)$ —①

$K_0 = 1$

$K_1 = 1 + n_0(k-1)$ using ①

$K_2 = K_1 + n_1(k-1)$

⋮

$K_L = 1 + n_0(k-1) + \ldots\ldots + n_{L-1}(k-1)$

$\Rightarrow 1 + (k-1)\underbrace{[n_0 + n_1 \ldots\ldots + n_{L-1}]}$

this is given as $[1, 2, 4, 8 \ldots.]$

$\Rightarrow 1 + (k-1)(2^L - 1)$ —②

b) For 2D we need the width and height of the kernel

If is given that kernel is $k \times k$ ∴ Receptive field = $②^2$

$\Rightarrow (1 + (k-1)(2^L - 1))^2$

c) The computational complexity should remain the same as standard convolution as dilation only affects the spacing but not

the number of operations.

∴ if the computational complexity of standard convolution
= $W \times H \times k^2$ (Assuming Cin and Cout = 1)
then it is the same for dilated convolutions.

Coding Questions

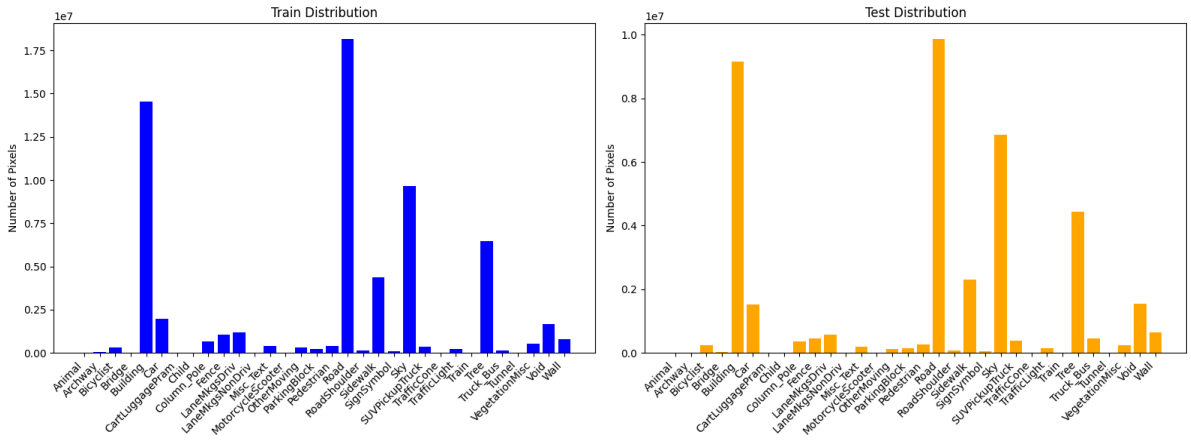
Q2.

1a.

```
class MyDataset(torch.utils.data.Dataset):
    def __init__(self, path, label_path):
        self.image_list = []
        self.label_list = []
        self.class_dict = pd.read_csv(path + "../class_dict.csv")
        self.transform_image = torchvision.transforms.Compose([
            torchvision.transforms.Resize((360, 480)),
            torchvision.transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
        ])
        self.transform_label = torchvision.transforms.Resize((360, 480), interpolation=torchvision.transforms.InterpolationMode.NEAREST)

        for i in os.listdir(path):
            image = os.path.join(path, i)
            self.image_list.append(image)

        for i in os.listdir(label_path):
            label = os.path.join(label_path, i)
            self.label_list.append(label)

        self.rgb_dict = {}

        for idx, row in self.class_dict.iterrows():
            self.rgb_dict[row["name"]] = [idx, row["r"], row["g"], row["b"]]


    def __len__(self):
        return len(self.image_list)
```

```
    def __getitem__(self, idx):
        image_path = self.image_list[idx]
        label_path = self.label_list[idx]

        image = torchvision.io.read_image(image_path, mode=torchvision.io.ImageReadMode.RGB).float()/255.0
        label = torchvision.io.read_image(label_path, mode=torchvision.io.ImageReadMode.RGB)

        image = self.transform_image(image)
        label = self.transform_label(label)

        encoded_label = torch.zeros(label.shape[1], label.shape[2], dtype=torch.long)

        for value in self.rgb_dict.values():
            class_idx, r, g, b = value
            mask = (label[0] == r) & (label[1] == g) & (label[2] == b)
            encoded_label[mask] = class_idx

        # print(encoded_label.shape)
        # print(encoded_label)
        return image, encoded_label
```

1b.

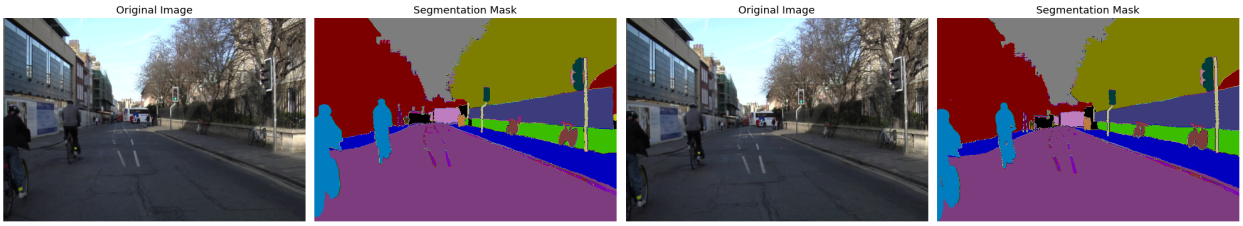


1c.

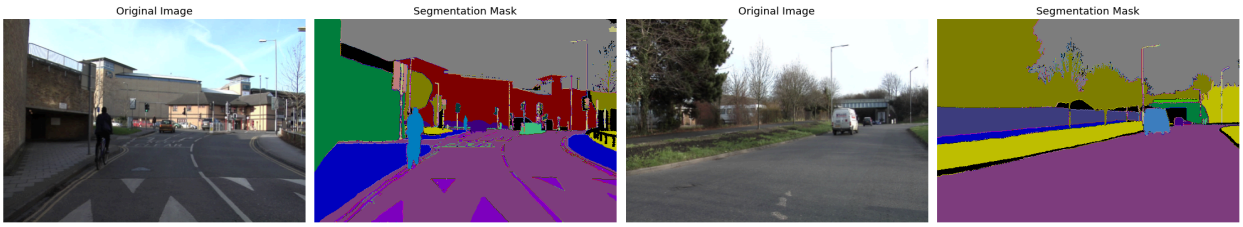
Animal:

Archway:

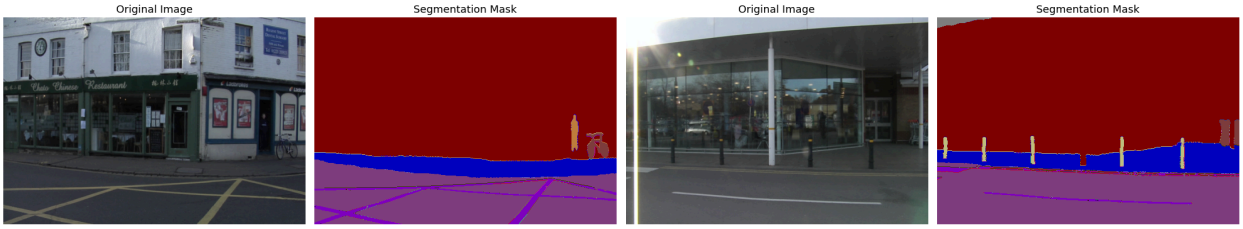
Bicyclist:



Bridge:

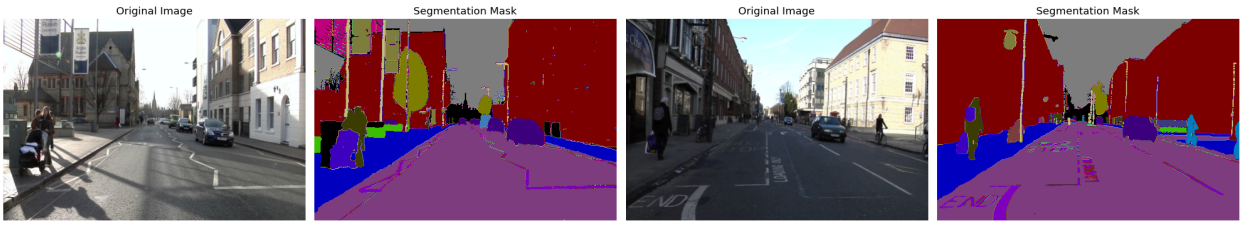


Building:



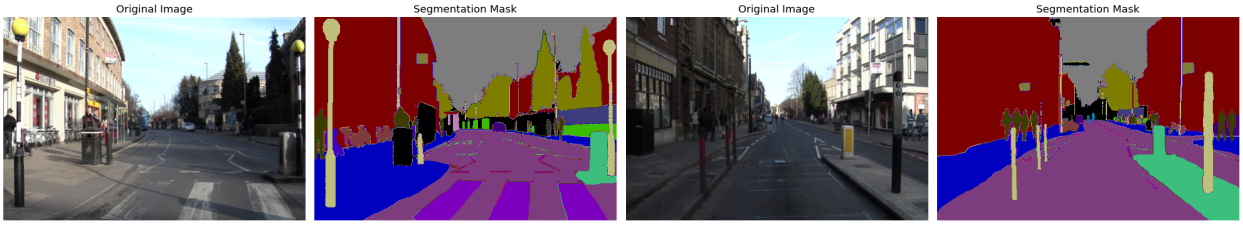
Car:

CartLuggagePram:

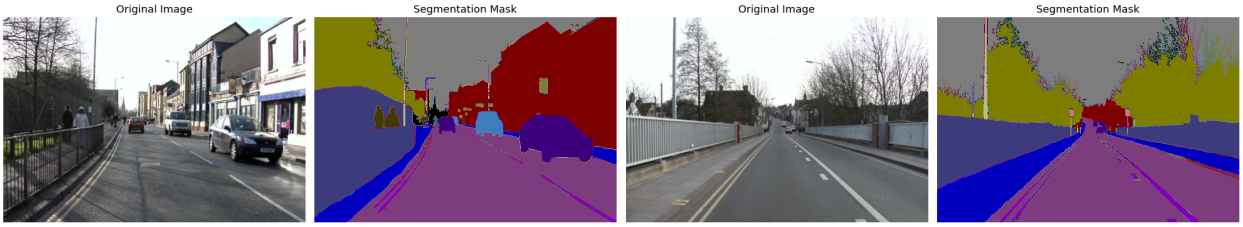


Child:

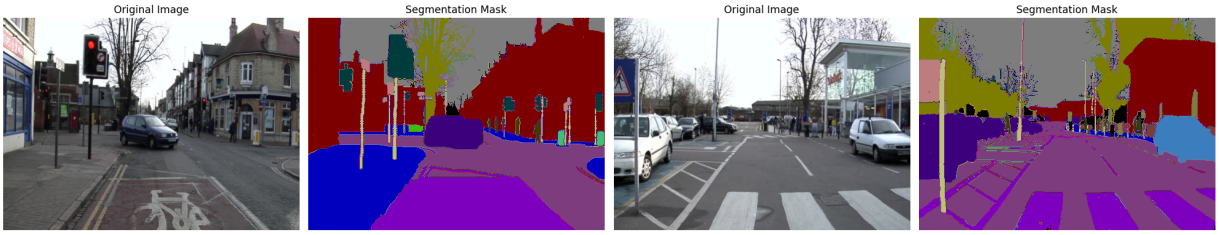
Column_Pole:



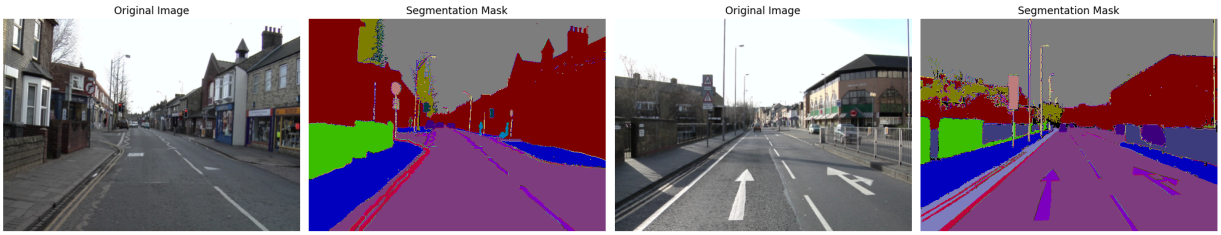
Fence:



LaneMkgsDriv:



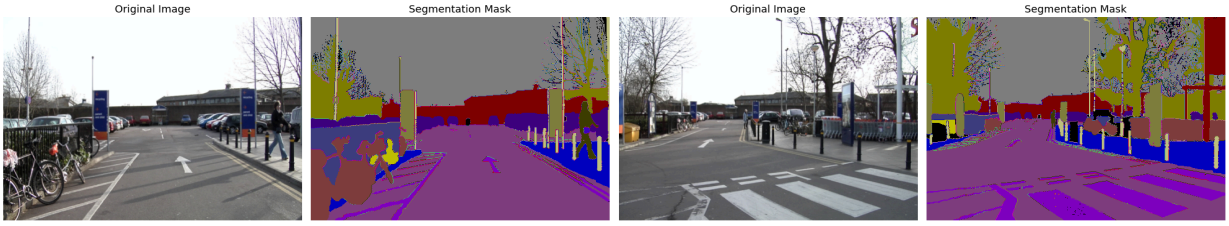
LaneMkgsNonDriv:



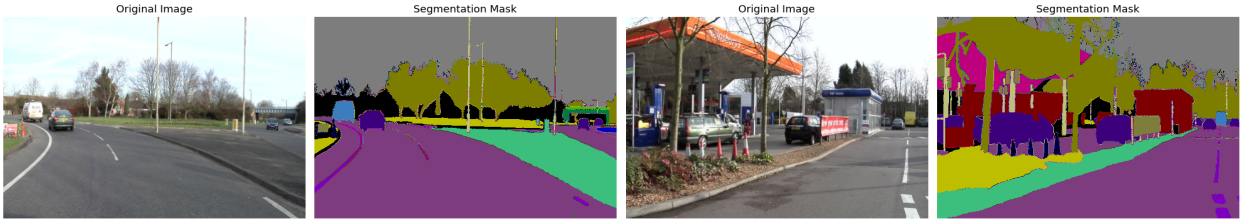
Misc_Text:

MotorcycleScooter:

OtherMoving:



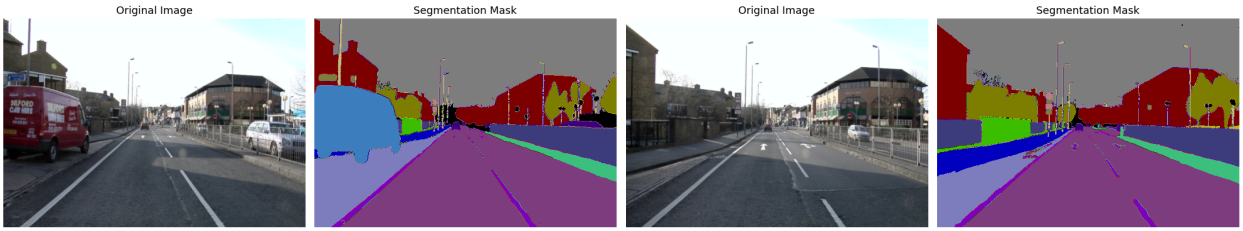
ParkingBlock:



Pedestrian:

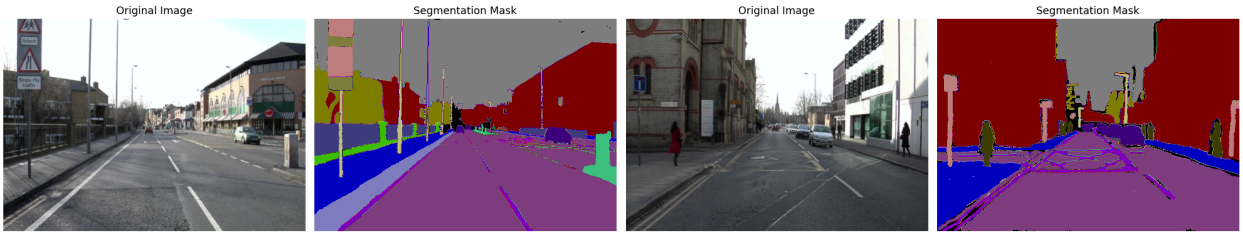
Road:

RoadShoulder:



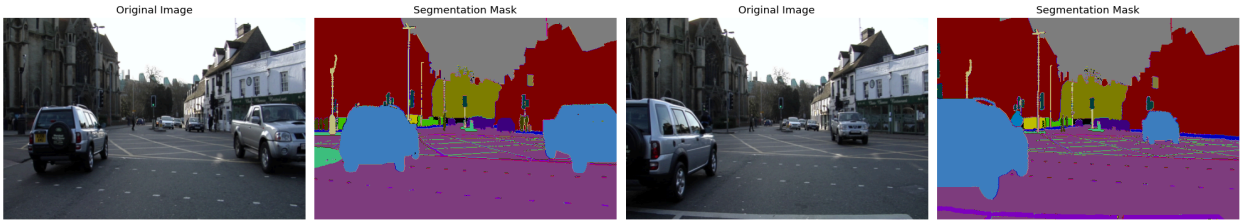
Sidewalk:

SignSymbol:



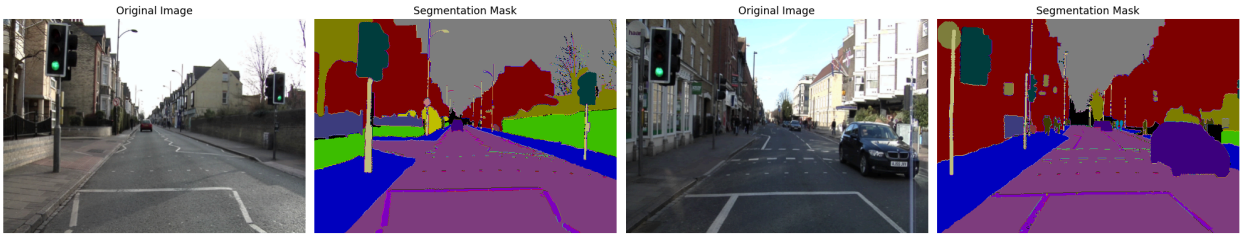
Sky:

SUVPickupTruck:

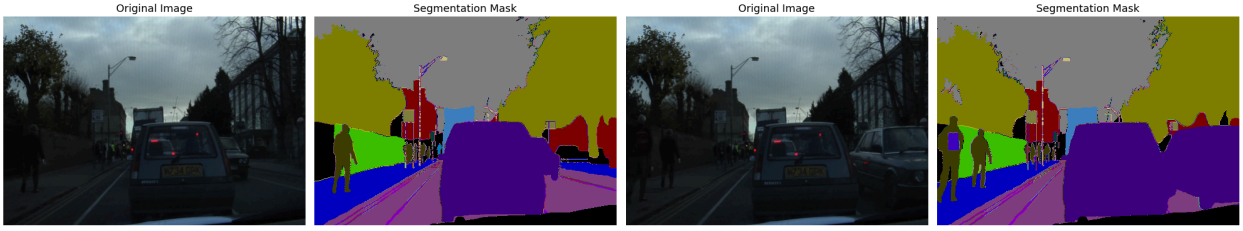


TrafficCone:

TrafficLight:

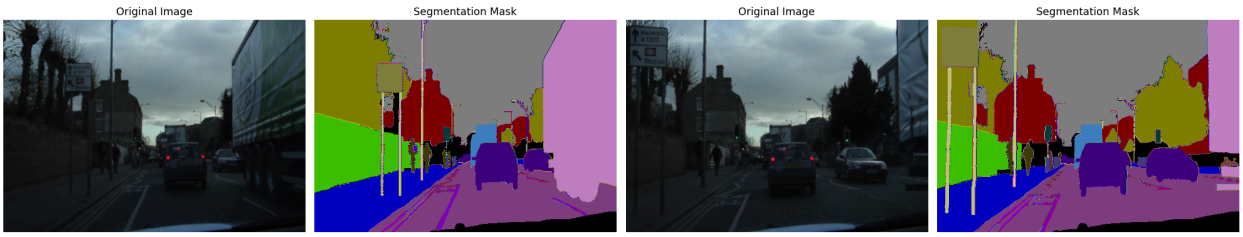


Train (There is some interpolation happening at the edges of the masks):

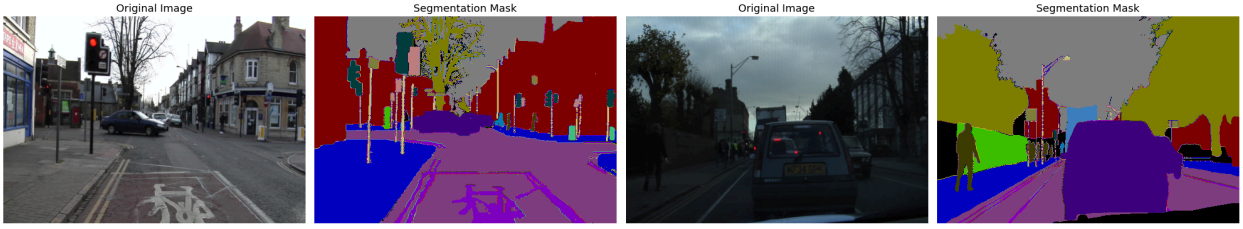


Tree:

Truck_Bus:



Tunnel:

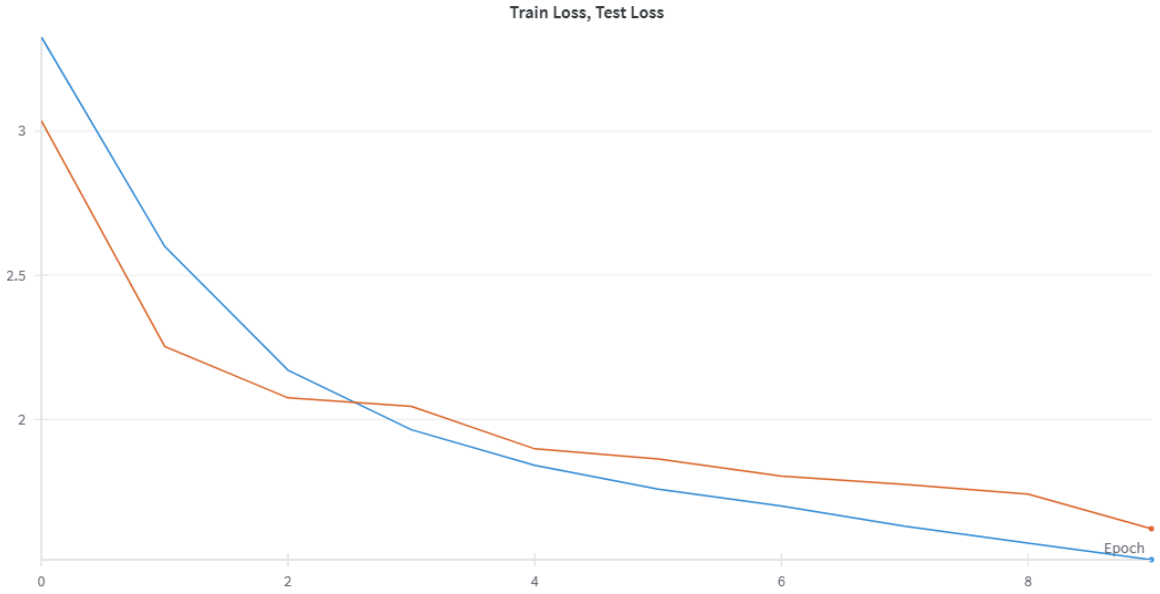


VegetationMisc:

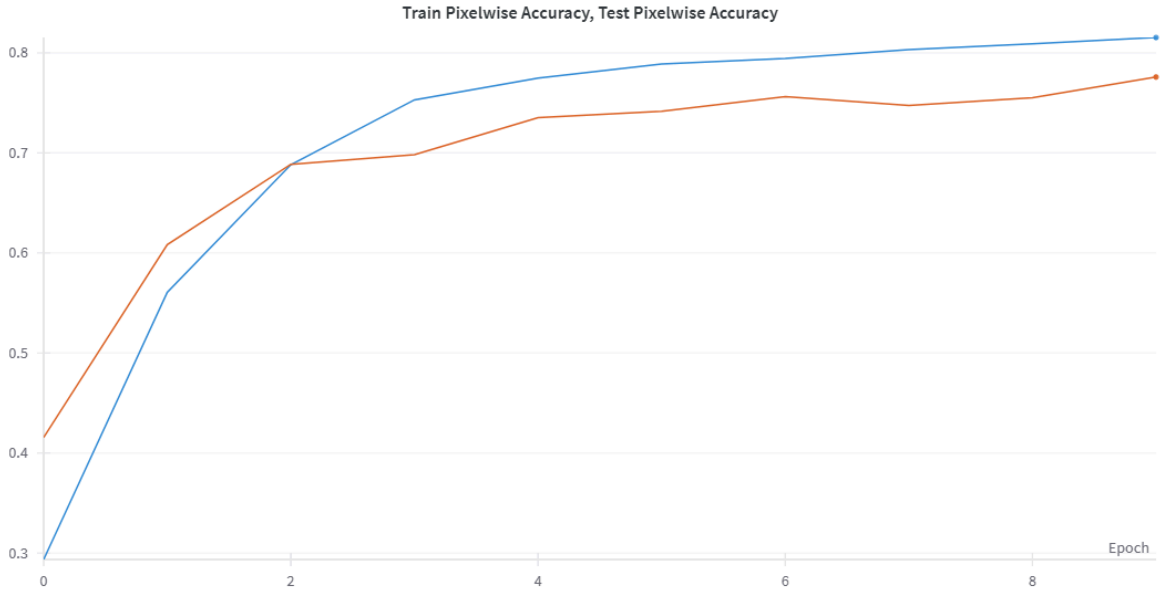
Void:

Wall:

2a.



Train Loss	1.51
Test Loss	1.62



Train Accuracy	0.81543
Test Accuracy	0.77583

```python
class SegNet_Decoder(nn.Module):
    def __init__(self, in_chn=3, out_chn=32, BN_momentum=0.5):
        super(SegNet_Decoder, self).__init__()
        self.in_chn = in_chn
        self.out_chn = out_chn
        #implement the architecture.

        # stage 5:
        # Max Unpooling: Upsample using ind5 to size4
        # Channels: 512 → 512 → 512 (3 convolutions)
        # Batch Norm: Applied after each convolution
        # Activation: ReLU after each batch norm
        self.MaxUn = nn.MaxUnpool2d(2, stride=2)

        self.Conv51 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.Bn51 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.Conv52 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.Bn52 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.Conv53 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.Bn53 = nn.BatchNorm2d(512, momentum=BN_momentum)

        #stage 4:
        # Max Unpooling: Upsample using ind4 to size3
        # Channels: 512 → 512 → 256 (3 convolutions)
        # Batch Norm: Applied after each convolution
        # Activation: ReLU after each batch norm
        self.Conv41 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.Bn41 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.Conv42 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.Bn42 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.Conv43 = nn.Conv2d(512, 256, kernel_size=3, padding=1)
        self.Bn43 = nn.BatchNorm2d(256, momentum=BN_momentum)

    # Stage 3:
    # Max Unpooling: Upsample using ind3 to size2
    # Channels: 256 → 256 → 128 (3 convolutions)
    # Batch Norm: Applied after each convolution
    # Activation: ReLU after each batch norm
    self.Conv31 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
    self.Bn31 = nn.BatchNorm2d(256, momentum=BN_momentum)
    self.Conv32 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
    self.Bn32 = nn.BatchNorm2d(256, momentum=BN_momentum)
    self.Conv33 = nn.Conv2d(256, 128, kernel_size=3, padding=1)
    self.Bn33 = nn.BatchNorm2d(128, momentum=BN_momentum)

    # Stage 2:
    # Max Unpooling: Upsample using ind2 to size1
    # Channels:  128 → 64 (2 convolutions)
    # Batch Norm: Applied after each convolution
    # Activation: ReLU after each batch norm
    self.Conv21 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
    self.Bn21 = nn.BatchNorm2d(128, momentum=BN_momentum)
    self.Conv22 = nn.Conv2d(128, 64, kernel_size=3, padding=1)
    self.Bn22 = nn.BatchNorm2d(64, momentum=BN_momentum)


    # Stage 1:
    # Max Unpooling: Upsample using ind1
    # Channels: 64 → out_chn (2 convolutions)
    # Batch Norm: Applied after each convolution
    # Activation: ReLU after the first convolution, no activation after the last one
    self.Conv11 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
    self.Bn11 = nn.BatchNorm2d(64, momentum=BN_momentum)
    self.Conv12 = nn.Conv2d(64, out_chn, kernel_size=3, padding=1)
    self.Bn12 = nn.BatchNorm2d(out_chn, momentum=BN_momentum)
```

```
    #For convolution use kernel size = 3, padding =1
    #for max unpooling use kernel size=2 ,stride=2
def forward(self,x,indexes,sizes):
    ind1,ind2,ind3,ind4,ind5=indexes[0],indexes[1],indexes[2],indexes[3],indexes[4]
    size1,size2,size3,size4,size5=sizes[0],sizes[1],sizes[2],sizes[3],sizes[4]

    x = self.MaxUn(x, ind5, output_size=size4)

    x = F.relu(self.Bn51(self.Conv51(x)))
    x = F.relu(self.Bn52(self.Conv52(x)))
    x = F.relu(self.Bn53(self.Conv53(x)))

    x = self.MaxUn(x, ind4, output_size=size3)

    x = F.relu(self.Bn41(self.Conv41(x)))
    x = F.relu(self.Bn42(self.Conv42(x)))
    x = F.relu(self.Bn43(self.Conv43(x)))

    x = self.MaxUn(x, ind3, output_size=size2)

    x = F.relu(self.Bn31(self.Conv31(x)))
    x = F.relu(self.Bn32(self.Conv32(x)))
    x = F.relu(self.Bn33(self.Conv33(x)))

    x = self.MaxUn(x, ind2, output_size=size1)

    x = F.relu(self.Bn21(self.Conv21(x)))
    x = F.relu(self.Bn22(self.Conv22(x)))

    x = self.MaxUn(x, ind1)

    x = F.relu(self.Bn11(self.Conv11(x)))
    x = self.Bn12(self.Conv12(x))

    return x
```

2b. Classwise Scores

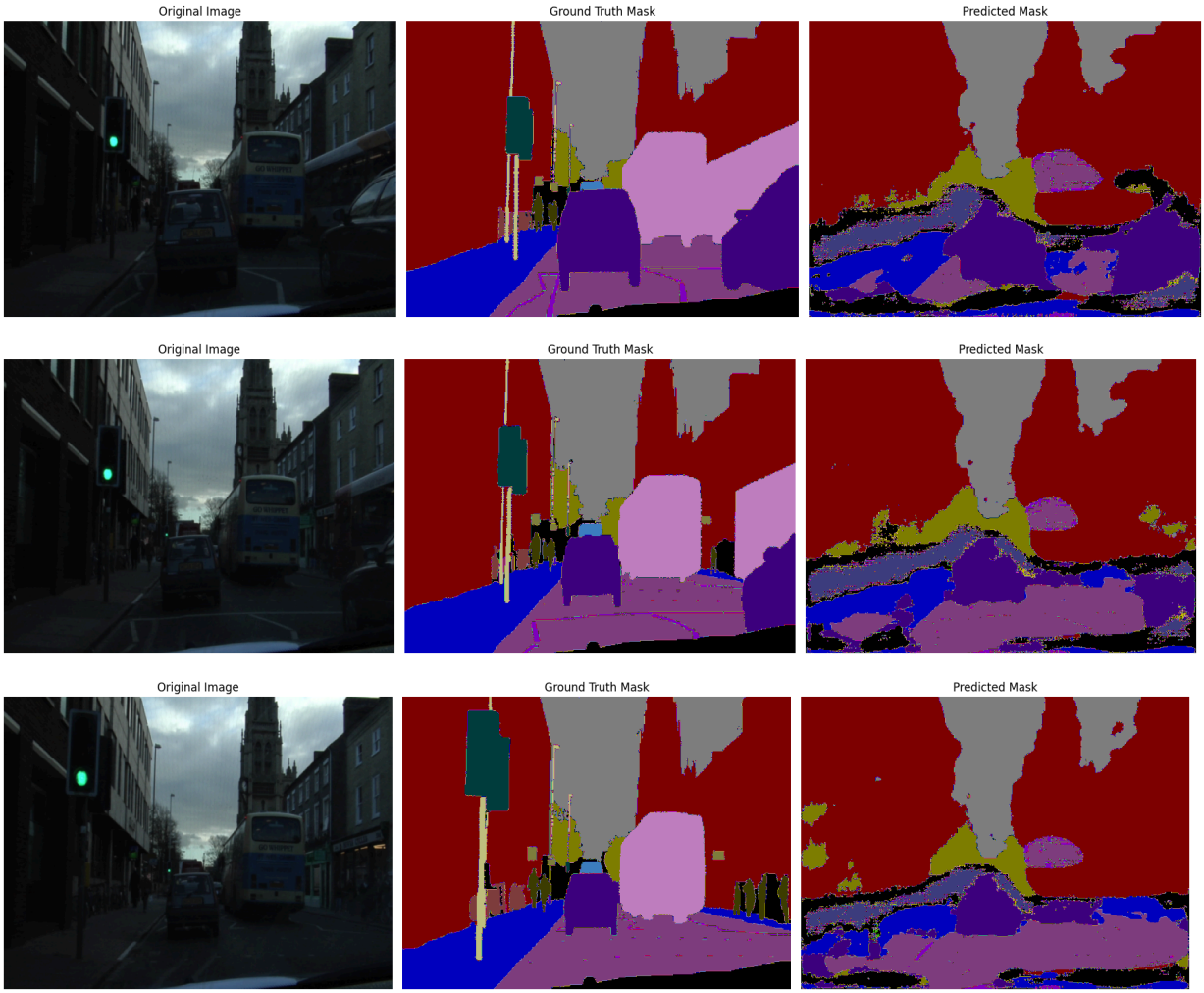
Class	Precision	Recall	Accuracy	Dice Coeff.	IOU
Animal	0	0	0.999	0	0
Archway	0	0	0.999	0	0
Bicyclist	0.11	0	0.993	0.002	0.001
Bridge	0	0	0.999	0	0
Building	0.78	0.92	0.922	0.84	0.72
Car	0.65	0.72	0.974	0.68	0.51
CartLuggagePram	0	0	0.999	0	0
Child	0	0	0.999	0	0
Column_Pole	0	0	0.990	0	0
Fence	0.11	0.36	0.961	0.17	0.09
LaneMkgsDriv	0.43	0.61	0.982	0.50	0.33

LaneMkgsNonDriv	0	0	0.999	0	0
Misc_Text	0	0	0.995	0	0
MotorcycleScooter	0	0	0.999	0	0
OtherMoving	0.02	0	0.996	0.001	0.0008
ParkingBlock	0	0	0.996	0	0
Pedestrian	0.05	0	0.993	0.00005	0.00002
Road	0.94	0.9	0.960	0.91	0.84
RoadShoulder	0	0	0.997	0	0
Sidewalk	0.65	0.86	0.965	0.73	0.58
SignSymbol	0	0	0.998	0	0
Sky	0.96	0.91	0.977	0.93	0.87
SUVPickupTruck	0.1	0	0.990	0.001	0.0005
TrafficCone	0	0	0.999	0	0
TrafficLight	0	0	0.999	0	0
Train	nan	nan	1	nan	nan
Tree	0.76	0.81	0.950	0.78	0.64
Truck_Bus	0	0	0.988	0	0
Tunnel	nan	nan	1	nan	nan
VegetationMisc	0.02	0	0.993	0.003	0.001
Void	0.27	0.28	0.942	0.27	0.15
Wall	0.11	0.01	0.98	0.009	0.004
Mean IOU					0.15982

Classwise Threshold Scores have been submitted in decoder_thresh_scores.csv

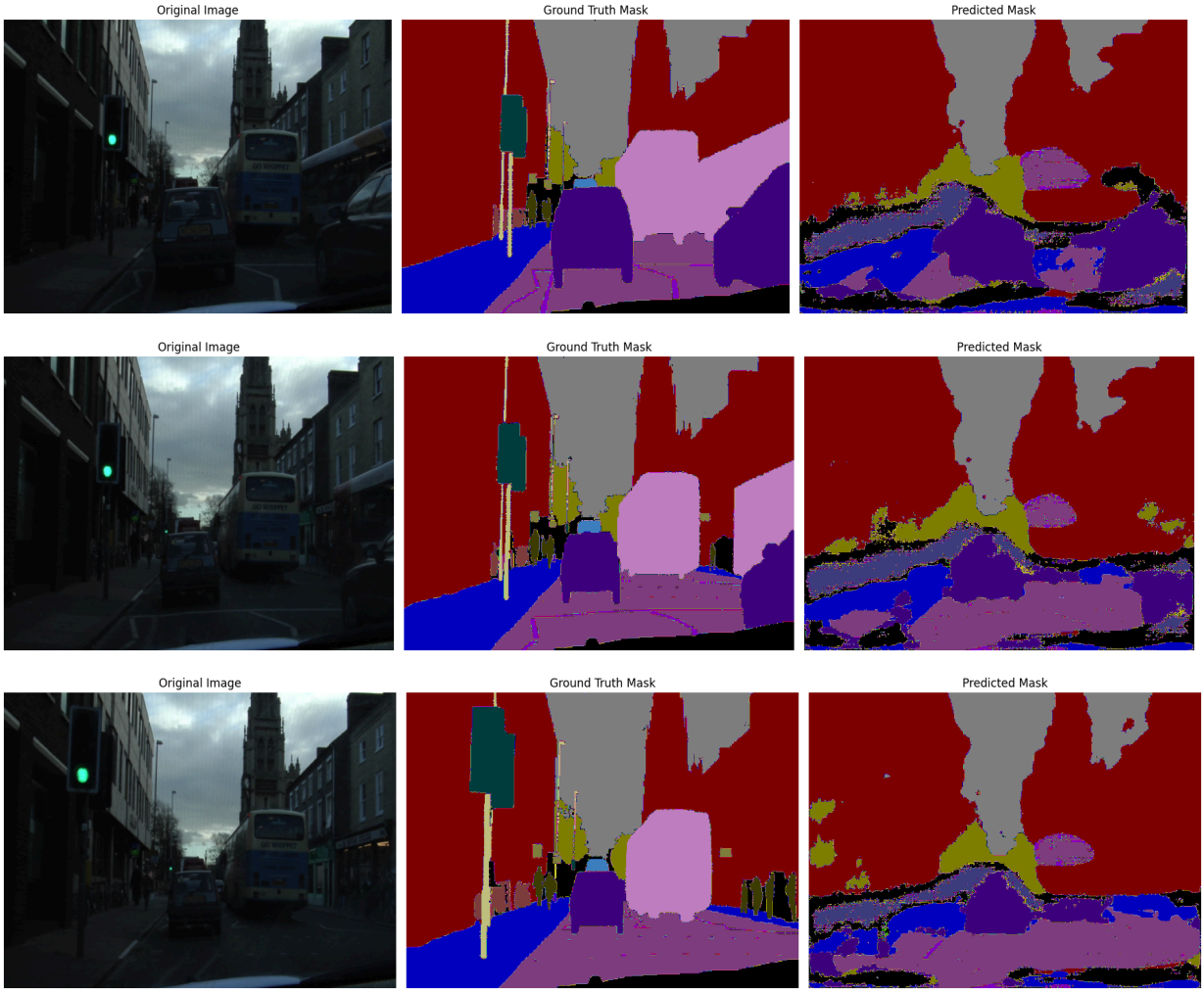
2c.

Class: VegetationMisc



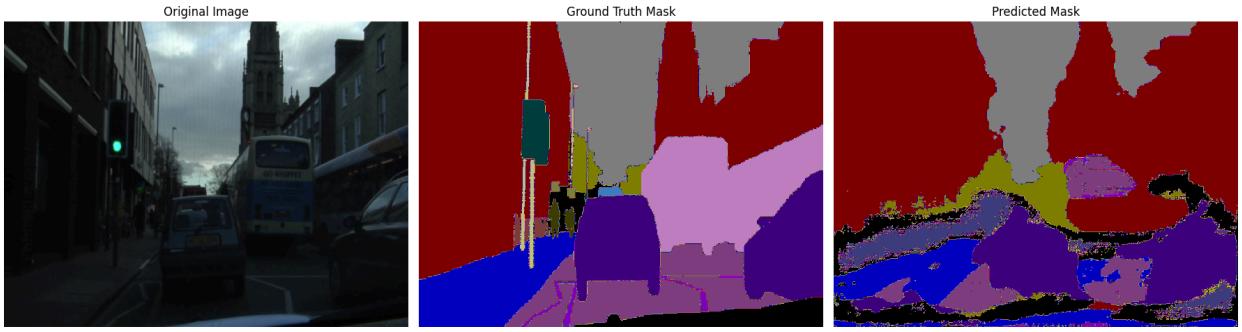
The model scores a low IOU on the VegetationMisc class, it could be due to occlusion as trees are usually far away and get covered by buildings, poles etc. Additionally, vegetation is not just in the form of trees, it is also in the form of grass which makes it a bit harder as grass is only green whereas trees have trunks which are brown. Also, these images are very dark which make it difficult to identify the trees.

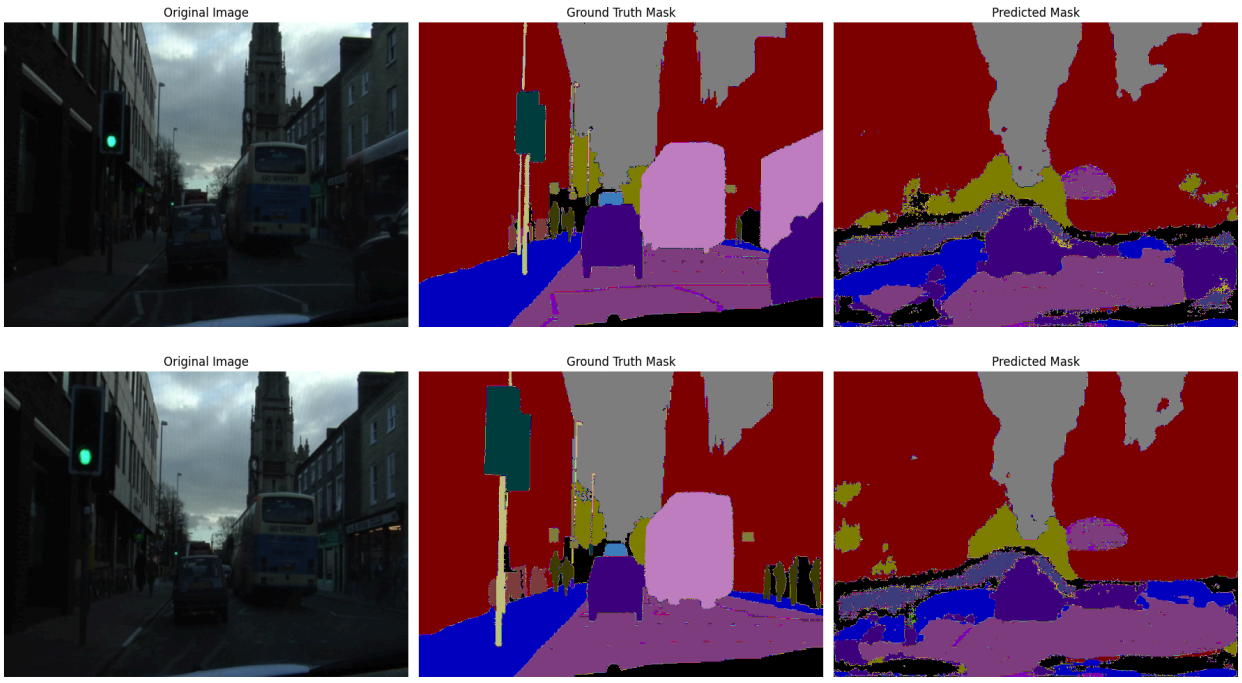
Class: Void



Similar to the tree class, void class gets occluded and is usually far away, surrounded by multiple classes which makes it hard for the model to detect the edges of the void class. Also, it is very difficult to understand when something could belong to the void class, as the objects that are in the void class are either very far away or very close and the darker images make it harder to identify what a void could be.

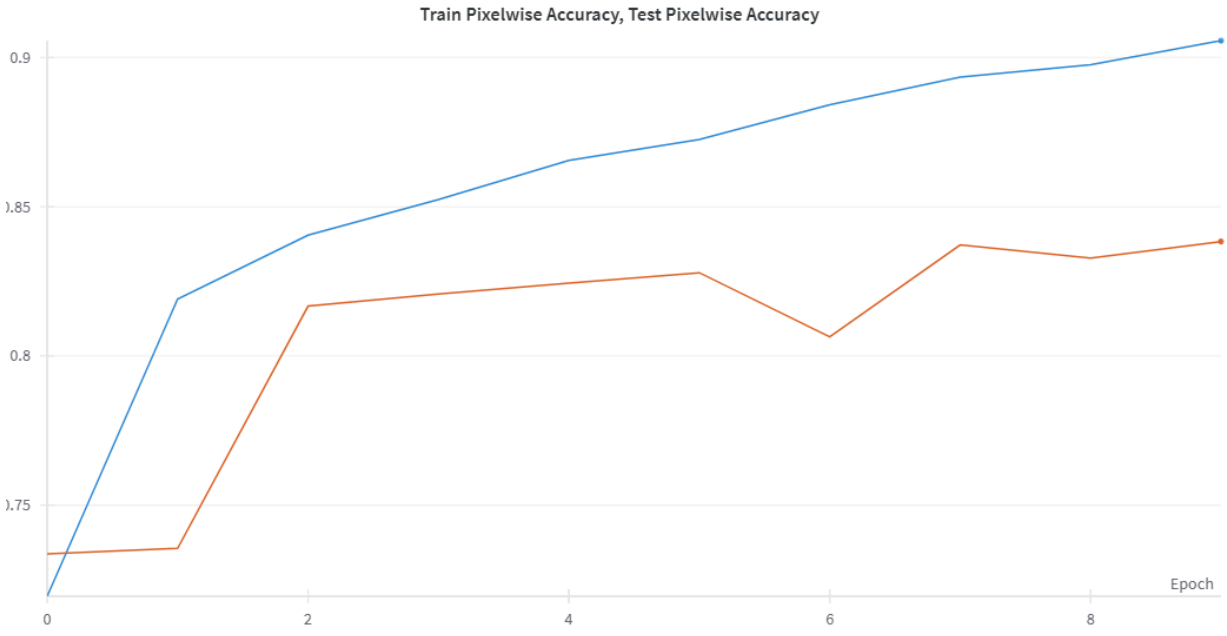
Class: Wall



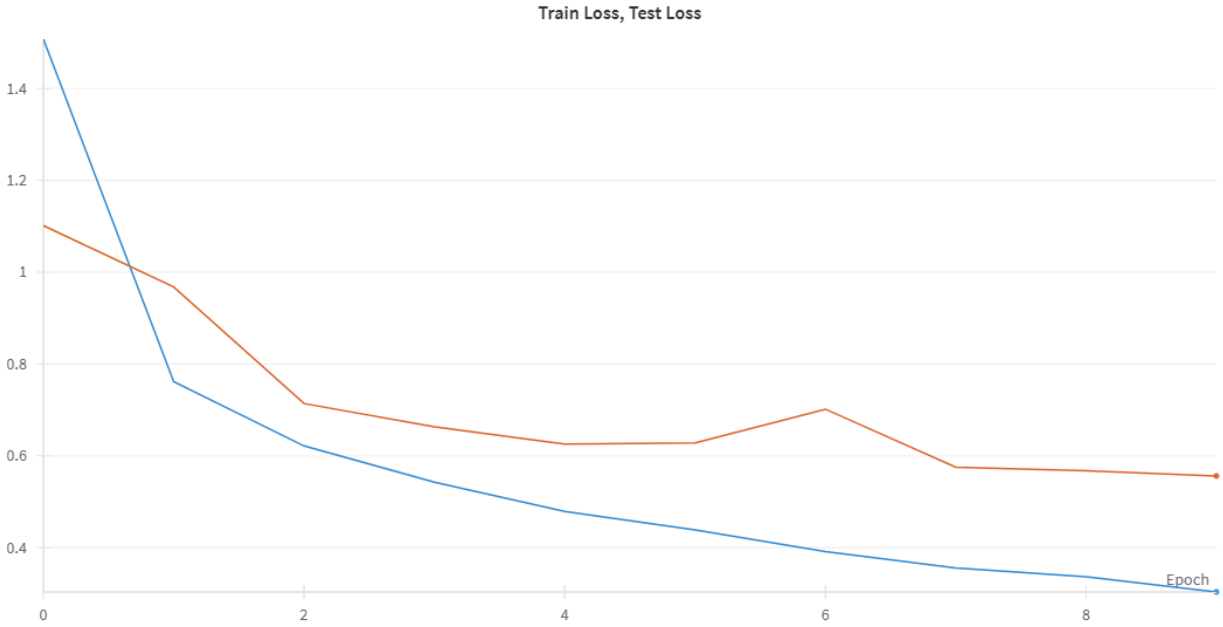


The model gets confused between the sides of the building which are technically walls, but the true label is building. The similarities between the building sides and wall are probably the reasons because of which the model misclassified buildings as walls. Darker images make it harder to identify.

3a.



Train Accuracy	0.90564
Test Accuracy	0.83834



Train Loss	0.30
Test Loss	0.55

```python
class DeepLabV3(nn.Module):
    def __init__(self, num_classes=32):
        super(DeepLabV3, self).__init__()
        self.model = torchvision.models.segmentation.deeplabv3_resnet50(pretrained=True)  # Initialize DeepLabV3 model here using pretrained=True
        self.model.classifier[4] = nn.Conv2d(in_channels=256, out_channels=num_classes, kernel_size=1, stride=1)
        #  should be a Conv2D layer with input channels as 256 and output channel as num_classes using a stride of 1, and kernel size of 1.

    def forward(self, x):
        return self.model(x)['out']
```

3b.

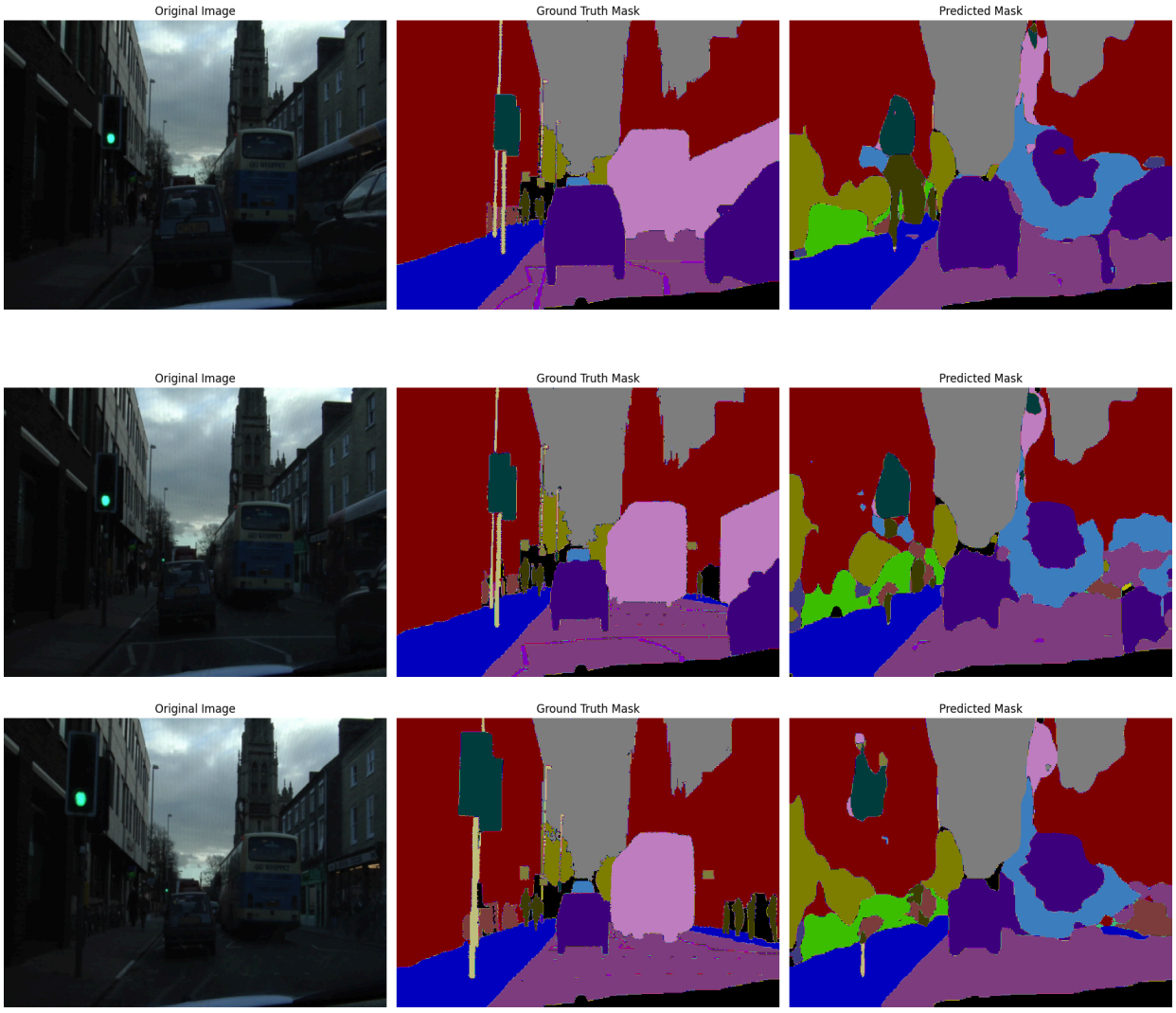
Class	Precision	Recall	Accuracy	Dice Coeff.	IOU
Animal	0	0	0.999	0	0
Archway	0	0	0.999	0	0

Bicyclist	0.7	0.54	0.995	0.61	0.44
Bridge	0	0	0.999	0	0
Building	0.91	0.84	0.945	0.87	0.77
Car	0.7	0.92	0.982	0.79	0.66
CartLuggagePram	0	0	0.999	0	0
Child	0	0	0.999	0	0
Column_Pole	0.42	0.07	0.990	0.11	0.06
Fence	0.57	0.55	0.990	0.55	0.38
LaneMkgsDriv	0.67	0.46	0.988	0.54	0.37
LaneMkgsNonDriv	0	0	0.999	0	0
Misc_Text	0.43	0.25	0.994	0.31	0.18
MotorcycleScooter	0	0	0.999	0	0
OtherMoving	0.15	0.65	0.987	0.24	0.13
ParkingBlock	0.59	0.47	0.997	0.52	0.35
Pedestrian	0.3	0.3	0.990	0.30	0.17
Road	0.94	0.95	0.972	0.94	0.89
RoadShoulder	0.59	0.76	0.998	0.66	0.49
Sidewalk	0.85	0.9	0.984	0.87	0.77
SignSymbol	0	0	0.998	0	0
Sky	0.93	0.97	0.982	0.94	0.90
SUVPickupTruck	0.15	0.34	0.975	0.21	0.11
TrafficCone	0	0	0.999	0	0
TrafficLight	0.63	0.55	0.997	0.59	0.41
Train	nan	nan	1	nan	nan
Tree	0.83	0.86	0.965	0.84	0.73

Truck_Bus	0.27	0.14	0.985	0.18	0.09
Tunnel	nan	nan	1	nan	nan
VegetationMisc	0.63	0.49	0.995	0.55	0.38
Void	0.76	0.45	0.973	0.56	0.39
Wall	0.53	0.71	0.985	0.60	0.43
Mean IOU					0.30707

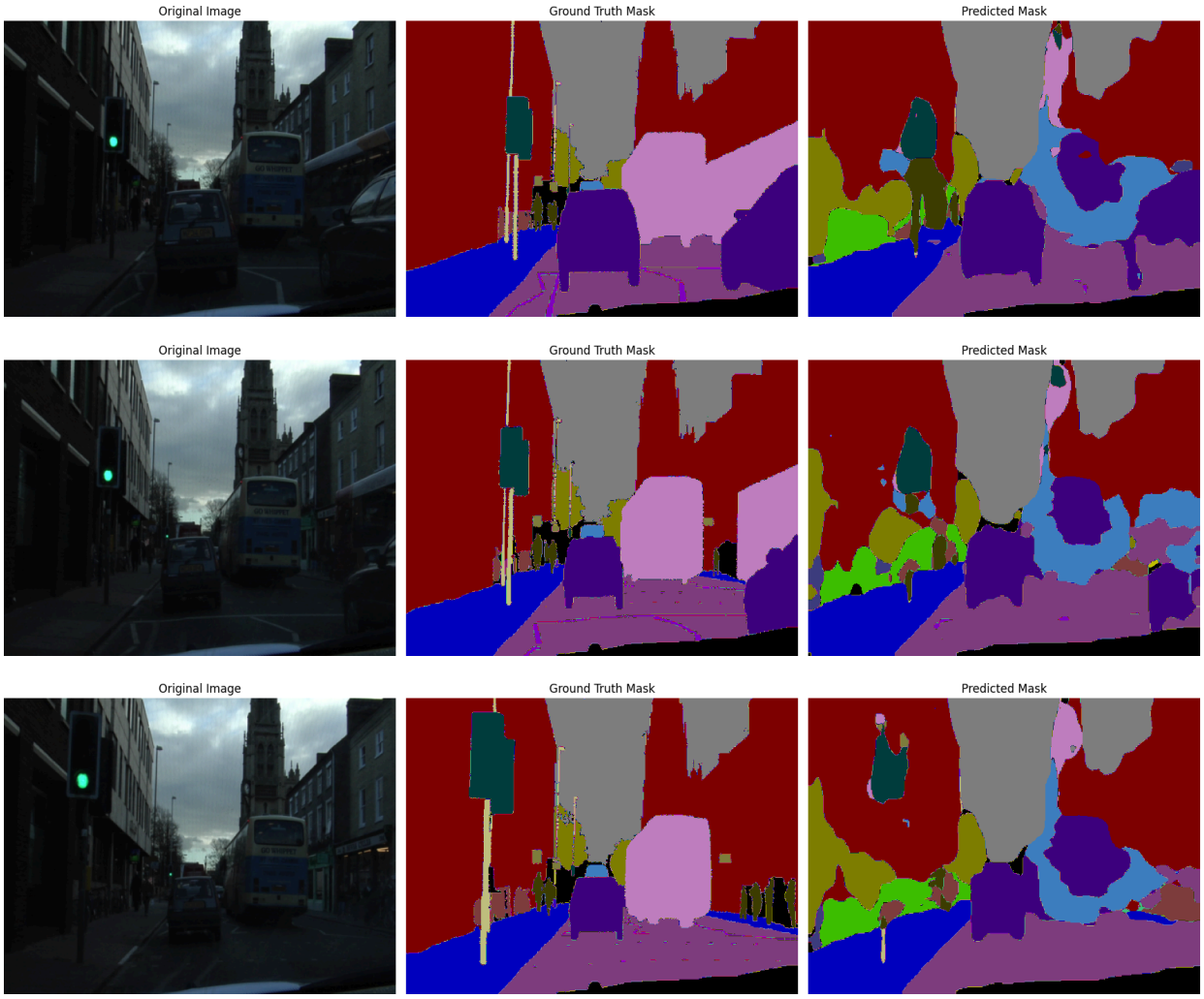
Classwise Threshold Scores have been submitted in deeplabv3_thresh_scores.csv

3c.
Class: VegetationMisc



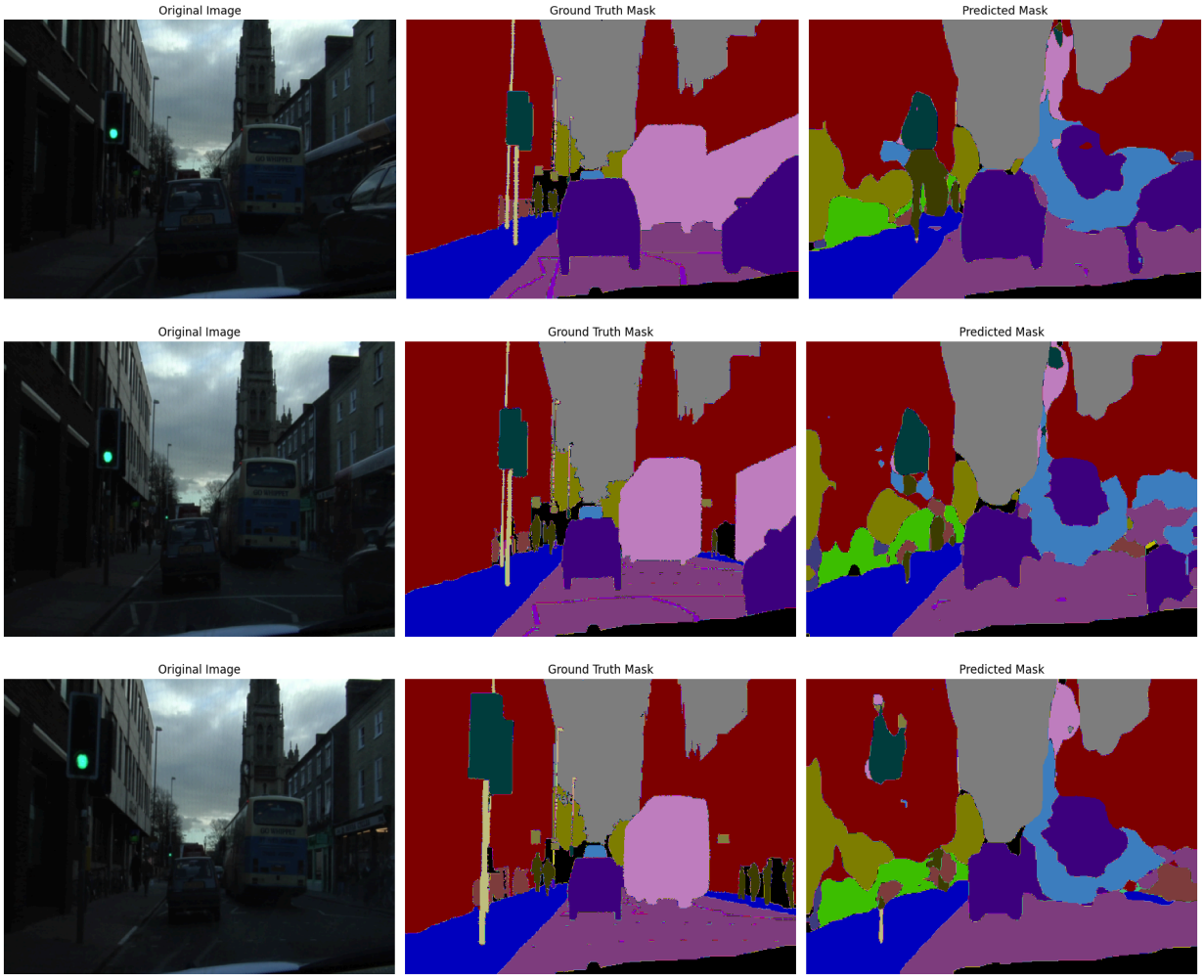
DeepLab performs better for these 3 classes, it could be due to the fact that it is a deeper model so it is better at capturing the finer details in the masks.

Class: Void



Similarly, it is able to capture the voids much better.

Class: Wall



There is a little bit of improvement in the Wall IOU as the model has not predicted the building as wall which the previous model was doing. The model performs better than the SegNet and captures the finer details much better.

Q3. NOTE: yolov8x performance and plots are available in the Detection-Starter-Notebook-v8x.ipynb

1a.

```
from ultralytics.utils.downloads import download
from pathlib import Path
import yaml

with open("data/coco.yaml", "r", encoding='utf-8') as file:
    data = yaml.safe_load(file)

segments = True
dir = Path(data['path'])
url = 'https://github.com/ultralytics/assets/releases/download/v0.0.0/'
urls = [url + ('coco2017labels-segments.zip' if segments else 'coco2017labels.zip')]
download(urls, dir=dir.parent)

urls = ['http://images.cocodataset.org/zips/val2017.zip']

download(urls, dir=dir / 'images', threads=3)
```

1b.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.315
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.423
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.346
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.113
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.345
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.480
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.261
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.365
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.368
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.128
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.400
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.564
```

“coco_predictions.json” is saved in the Detection folder (yolov8n model). yolov8x‘s predictions are saved inside *“Detection\data\datasets\coco\annotations\yolov8x_predictions.json”*

1c.

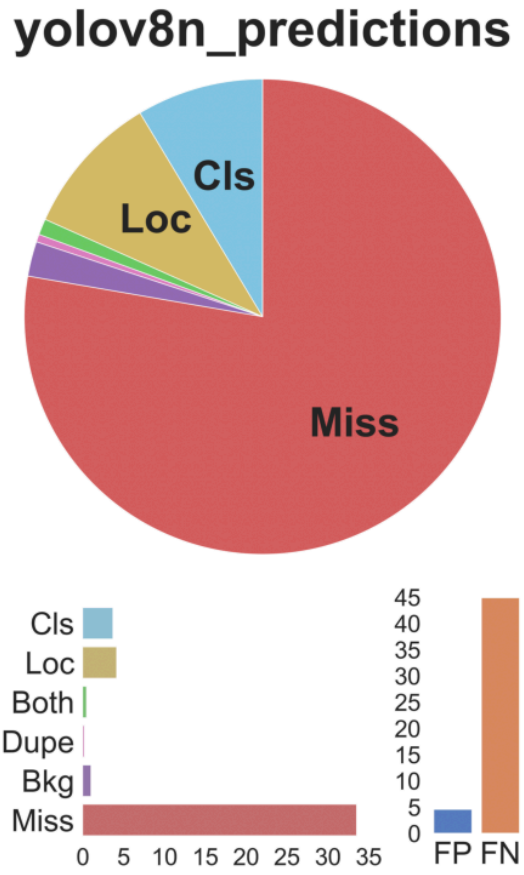
bbox AP @ 50: 42.35

Main Errors

Type	Cls	Loc	Both	Dupe	Bkg	Miss
dAP	3.71	4.17	0.47	0.22	1.02	33.52

Special Error

Type	FalsePos	FalseNeg
dAP	4.59	44.89



Analysis:
Most of the pie chart is “miss” that is the model fails to identify the object, hence the large number of false negatives. Localization and Classification errors are very small. Therefore, it makes small incorrect identifications and misses identifications more frequently.

1d.
ECE Score: 0.0991687659844248
The low ECE score indicates that the model is well calibrated. This was achieved by creating 10 bins of size 0.1 each. If the image id of the detected bounded box matches with the ground truth's image id then I compare the category id of the two, if it matches then I compare the iou (I select the image with the highest iou and above the threshold which is 0.5). Once this ground truth image has been chosen then we do not use it again.

After this I create bins using the obtained ground truth values for each prediction and add the accuracy and confidence scores in each bin.

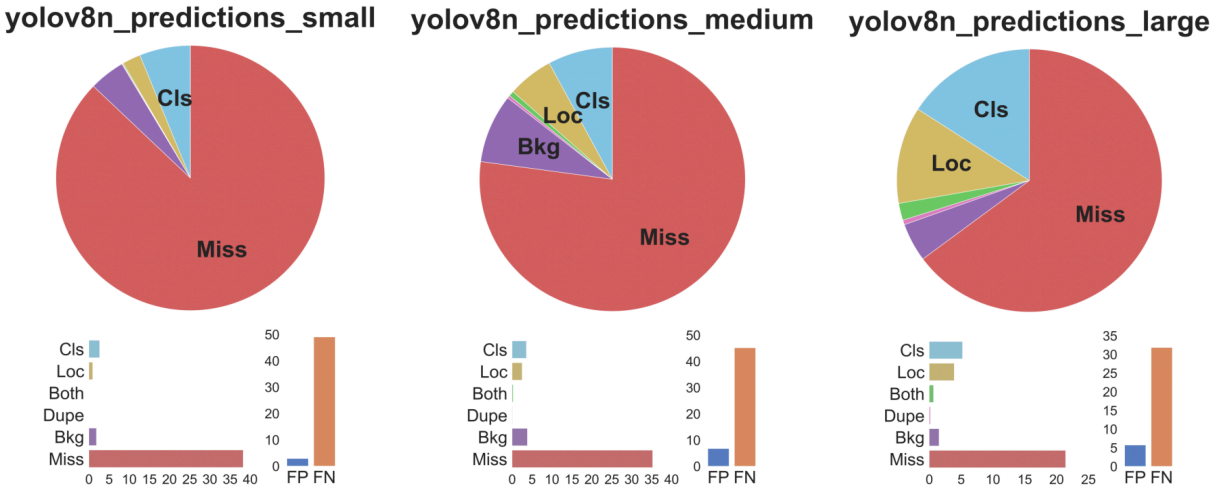
The code is available in the *"Detection-Starter-Notebook"*.

1e.

Small	0.12355794647264547
Medium	0.08868020863418118
Large	0.06691469561411154

The model tends to perform worse on the smaller objects, this could be due to the limited information that is available from those pixels. As the size of the object and bounding box increases the performance of the model improves slightly.

The code is available in the *"Detection-Starter-Notebook"*.



-- yolov8n_predictions_small --

bbox AP @ 50: 10.63

Main Errors

Type	Cls	Loc	Both	Dupe	Bkg	Miss
dAP	2.69	0.96	0.05	0.04	1.89	38.27

Special Error

Type	FalsePos	FalseNeg
dAP	2.92	49.05

-- yolov8n_predictions_medium --

bbox AP @ 50: 35.35

Main Errors

Type	Cls	Loc	Both	Dupe	Bkg	Miss
dAP	3.60	2.55	0.29	0.15	3.86	35.27

Special Error

Type	FalsePos	FalseNeg
dAP	6.78	45.19

-- yolov8n_predictions_large --

bbox AP @ 50: 57.24

Main Errors

Type	Cls	Loc	Both	Dupe	Bkg	Miss
dAP	5.25	3.94	0.69	0.19	1.56	21.44

Special Error

Type	FalsePos	FalseNeg
dAP	5.73	31.86

1f.

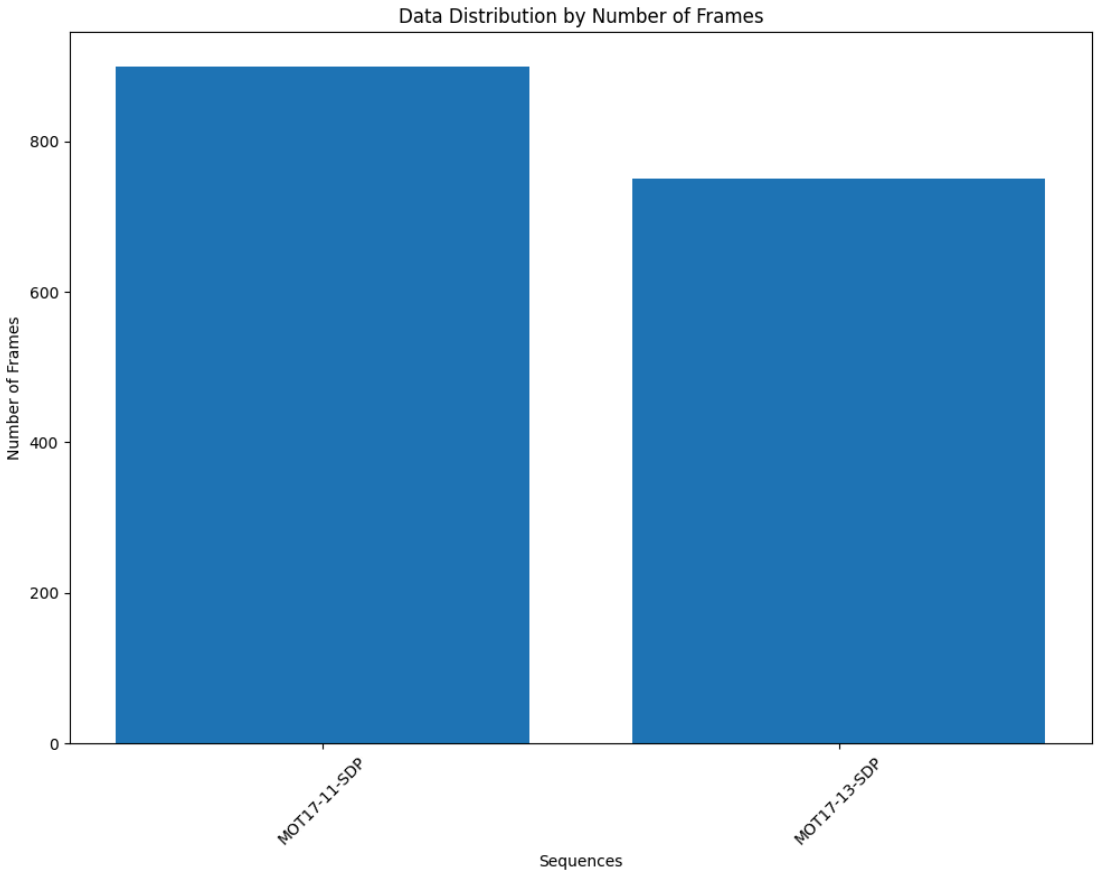
a. The model struggles the most with small objects, achieving a very low AP. Performance improves for medium-sized objects and large objects are detected the best as they have the least miss rate and the highest AP out of the three. This could be possibly due to the fact that there is more information available for larger objects.
b. **Small -** model has extremely high miss rate and a higher ECE compared to medium and large, this probably means that the model is underconfident in the predictions and misses them. **Medium -** The performance improves slightly but still remains suboptimal. **Large -** It performs the best evident from the low ECE score and low miss rate, though it has higher false positives than small boxes possibly due to overlapping bounding boxes or incorrect classifications.
c. The smaller objects have a little worse performance than the scores achieved in 3c and 3d, medium performs similar but still a little worse and large performs better than them.

2a.

First frame of the first video:

First frame of the second video:

Data distribution:



Citations:

I have included citations in the notebook itself however some other things which have helped me in this assignment are:

1. DL Course (Winter 25') taught by Dr. Vinayak Abrol
2. CV Lecture Notes
3. Convolution Arithmetic (on GC)
4. Resources provided in the assignment