VARIN KALA
2022561

CV Assignment 1 Theory

Date

Q1) a) If P and Q are 2 probability distributions drawn from Random Variable X,
Cross-Entropy Loss between P & Q

$$H(P,Q) = -\sum_{i=1}^{K} P_i \log Q_i$$

where K is the number of classes.

$\because$ The true class label is One-Hot encoded,

$$P_i = \begin{cases} 0 & \text{if } i \neq \text{true class} \\ 1 & \text{otherwise} \end{cases}$$

$$\therefore H(P,Q) = -\log Q_t$$

where t is the True Class

When we apply Label Smoothing, P_i becomes

$$P_i = \begin{cases} \varepsilon/K & \text{if } i \neq \text{true class} \\ 1-\varepsilon+\varepsilon/K & \text{otherwise} \end{cases}$$

$\therefore$ Cross Entropy Now becomes

$$H(P,Q) = -\sum_{i=1}^{K} P_i \log Q_i$$

$$H(P,Q) = -\left(\left(1-\varepsilon+\varepsilon/K\right)\log Q_t + \sum_{i\neq t} \frac{\varepsilon}{K} \log Q_i\right)$$

$$H(P,Q) = -\frac{\varepsilon}{K}\sum_{i=1}^{K} \log Q_i - (1-\varepsilon)\log Q_t$$

where t is the True Class.

Spiral

Date

b) Label Smoothing ~~helps the~~ allows the model to avoid the disadvantages of hard classification to one particular class and assigns small probability ϵ/k to each class that is not the predicted one.

It also introduces ~~helps in~~ regularization and generalization making it a ~~better~~ model better fit for unseen data.

Q2) we have 2 prob distributions p & q over x

a) $p(x) = \mathcal{N}(\mu_p, \sigma_p^2) \qquad q(x) = \mathcal{N}(\mu_q, \sigma_q^2)$

Cross Entropy between $p(x)$ and $q(x)$

$$H(p,q) = -\int p(x) \log(q(x))\, dx$$

$$H(p,q) = E_{p\sim x}\left[-\log(q(x))\right]$$

b) $$H(p,q) = -\int p(x) \log(q(x))\, dx$$

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

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

$$\therefore \log(q(x)) = -\frac{1}{2}\log(2\pi\sigma_q^2) - \frac{(x-\mu_q)^2}{2\sigma_q^2}$$

Date

$$\therefore H(p,q) = \int p(x)\left[\frac{1}{2}\log(2\pi) + \log(|\sigma_q|) + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right]dx$$

$$H(p,q) = \mathbb{E}_{p\sim x}\left[\frac{1}{2}\log(2\pi) + \log(|\sigma_q|) + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right]$$

$$H(p,q) = \frac{1}{2}\log(2\pi\sigma_q^2) + \frac{1}{2\sigma_q^2}\underbrace{\int p(x)(x-\mu_q)^2\,dx}_{\mathbb{E}_{p\sim x}\left[(x-\mu_q)^2\right]}$$

we can write $\mathbb{E}_{p\sim x}\left[(x-\mu_q)^2\right]$ as

$$\mathbb{E}_{p\sim x}\left[(x-\mu_q)^2\right] = \mathbb{E}\left[(x-\mu_p+\mu_p-\mu_q)^2\right]$$

$$= \sigma_p^2 + (\mu_p-\mu_q)^2$$

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

When $\sigma_p = \sigma_q = \sigma$

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

Here, we can clearly see that as the two means grow further apart, the Cross Entropy b/w p & q will increase.

Spiral

Date

Q3) For 1-D convolution, ~~Receptive field~~

$k \rightarrow$ kernel size $\quad r \rightarrow$ Dilation, $L \rightarrow$ layers

$(k-1)(r-1) \rightarrow$ extra space due to dilation

$\therefore$ Receptive field $RF = k + (k-1)(r-1)$

For L layers,

$RF_2 = RF_1 + (k-1)r_2$

$RF_3 = RF_2 + (k-1)r_3$

$= 1 + (k-1)(r_1 + r_2 + r_3)$

For $r = 1, 2, 4 \ldots$

$\therefore RF_L = 1 + (k-1)(2^L - 1)$

$\therefore$ Receptive field is growing exponentialy.

For 2D convolutions of a square kernel,

$RF = (1D\text{-}RF)^2$

$\therefore RF_L^2 = (RF_L^{1D})^2$

$\therefore RF_L^2 = (1 + (k-1)(2^L - 1))^2$

$\therefore$ RF increases more rapidly now

Space complexity depends on kernel size and feature map size, which stay the same

$\therefore$ Space complexity is same for both : $O(MNK^2)$

Spiral

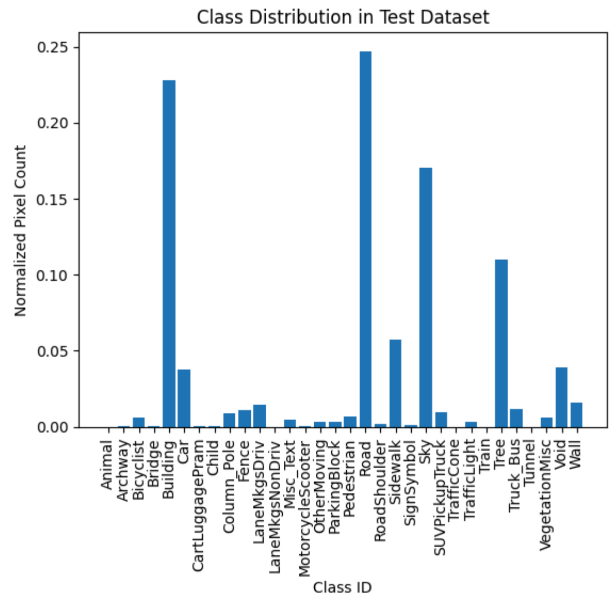
CV A1

Q3) Semantic Segmentation

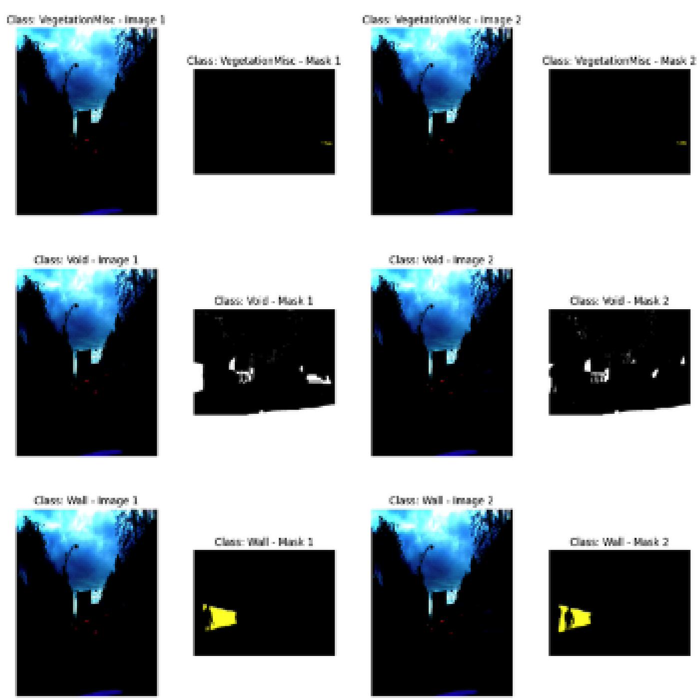
1.a. Custom Dataset Class

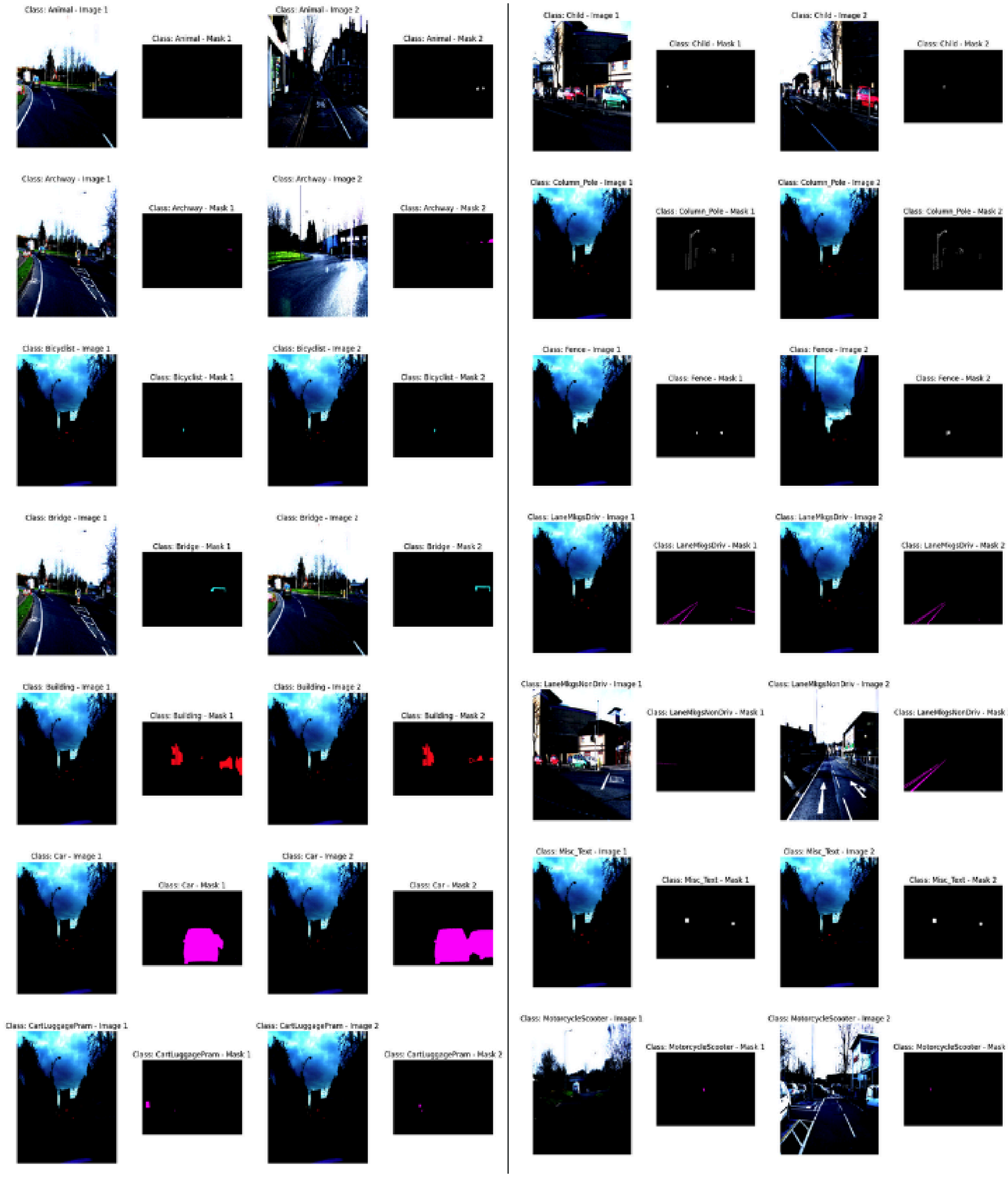
```
class CamVidDataset(Dataset):
    def __init__(self, data_paths, label_paths, transform=None):
        self.data_paths = data_paths
        self.label_paths = label_paths

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

        if transform:
            self.transform = transforms.Compose([
                *self.transform.transforms,
                *transform.transforms
            ])

    def convert_rgb_to_id(self, image):
        image = image.resize((480, 360), Image.NEAREST)
        image = np.array(image)

        new_image = np.zeros((360, 480, 32), dtype=np.float32)

        i = 0
        for rgb, class_id in rgb_to_id.items():
            image_indices = np.all(image == np.array(rgb), axis=-1)
            new_image[image_indices] = class_id
            i += 1

        return torch.tensor(new_image, dtype=torch.float32)

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

    def __getitem__(self, index):
        data = Image.open(self.data_paths[index]).convert('RGB')
        label = Image.open(self.label_paths[index]).convert('RGB')

        data = self.transform(data).permute(0, 2, 1)
        label = self.convert_rgb_to_id(label).permute(2, 0, 1)

        return data, label
```

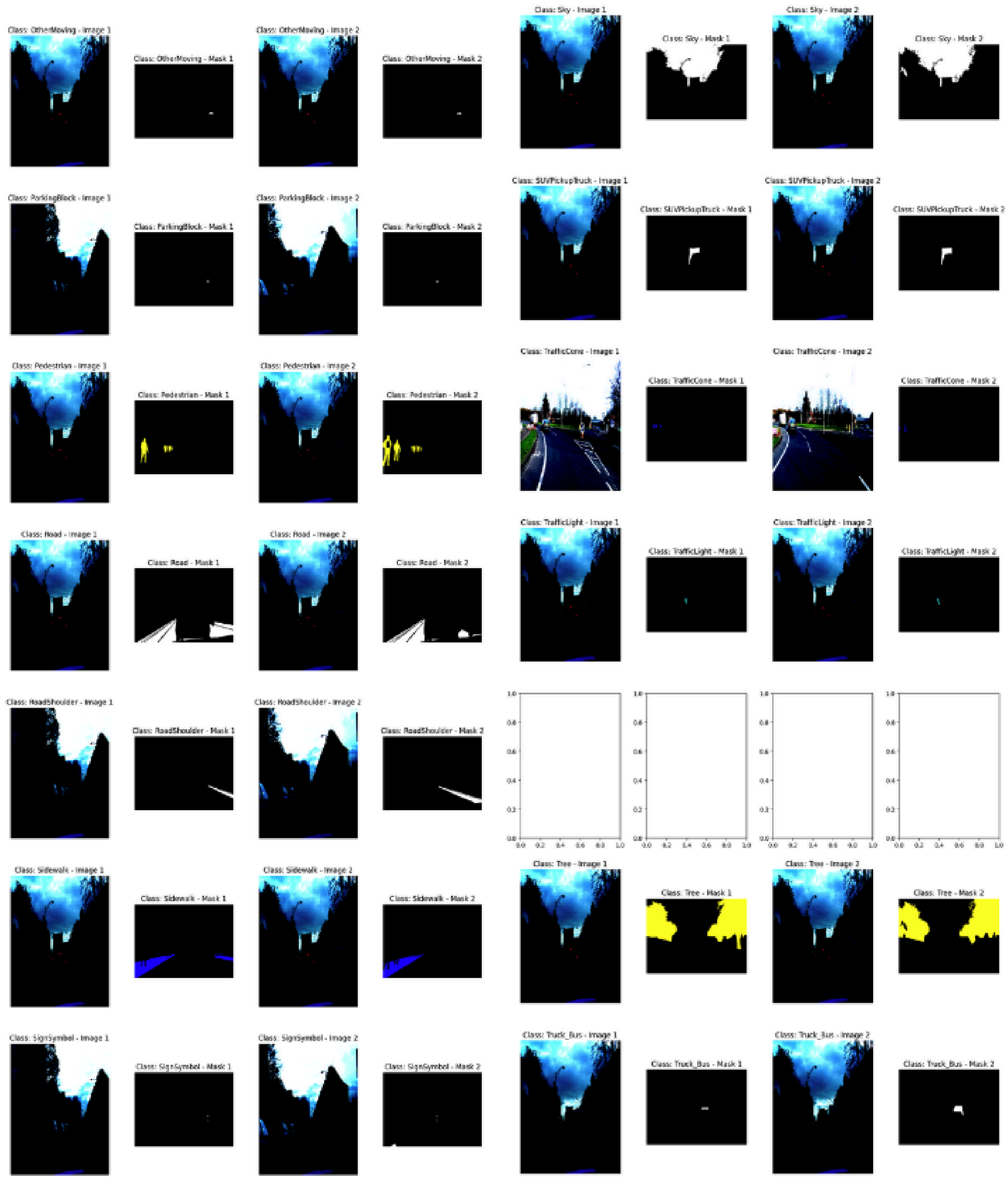
1.b. Class Distribution



1.c. Images with their masks



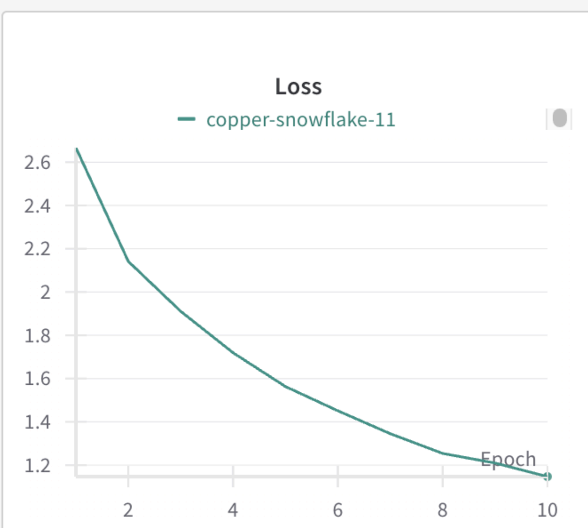

Class: Animal - Image 1
Class: Animal - Mask 1
Class: Animal - Image 2
Class: Animal - Mask 2
Class: Archway - Image 1
Class: Archway - Mask 1
Class: Archway - Image 2
Class: Archway - Mask 2
Class: Bicyclist - Image 1
Class: Bicyclist - Mask 1
Class: Bicyclist - Image 2
Class: Bicyclist - Mask 2
Class: Bridge - Image 1
Class: Bridge - Mask 1
Class: Bridge - Image 2
Class: Bridge - Mask 2
Class: Building - Image 1
Class: Building - Mask 1
Class: Building - Image 2
Class: Building - Mask 2
Class: Car - Image 1
Class: Car - Mask 1
Class: Car - Image 2
Class: Car - Mask 2
Class: CartLuggagePram - Image 1
Class: CartLuggagePram - Mask 1
Class: CartLuggagePram - Image 2
Class: CartLuggagePram - Mask 2
Class: Child - Image 1
Class: Child - Mask 1
Class: Child - Image 2
Class: Child - Mask 2
Class: Column_Pole - Image 1
Class: Column_Pole - Mask 1
Class: Column_Pole - Image 2
Class: Column_Pole - Mask 2
Class: Fence - Image 1
Class: Fence - Mask 1
Class: Fence - Image 2
Class: Fence - Mask 2
Class: LaneMkgsDriv - Image 1
Class: LaneMkgsDriv - Mask 1
Class: LaneMkgsDriv - Image 2
Class: LaneMkgsDriv - Mask 2
Class: LaneMkgsNonDriv - Image 1
Class: LaneMkgsNonDriv - Mask 1
Class: LaneMkgsNonDriv - Image 2
Class: LaneMkgsNonDriv - Mask 2
Class: Misc_Text - Image 1
Class: Misc_Text - Mask 1
Class: Misc_Text - Image 2
Class: Misc_Text - Mask 2
Class: MotorcycleScooter - Image 1
Class: MotorcycleScooter - Mask 1
Class: MotorcycleScooter - Image 2
Class: MotorcycleScooter - Mask 2

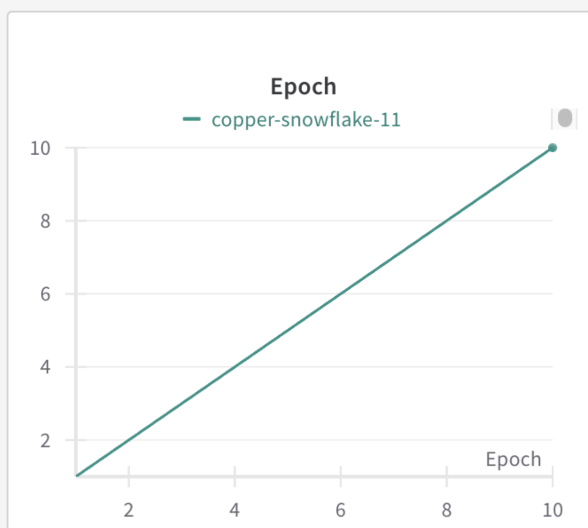


2.a. Segnet Encoder-Decoder

```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

        self.max_unpool = nn.MaxUnpool2d(2, stride=2)

        # 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.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
```

```python
def forward(self, x, indexes, sizes):
    # print(indexes[4].shape, sizes[4])

    # Stage 5
    x = self.max_unpool(x, indexes[4], sizes[3])
    x = F.relu(self.bn51(self.conv51(x)))
    x = F.relu(self.bn52(self.conv52(x)))
    x = F.relu(self.bn53(self.conv53(x)))

    # Stage 4
    x = self.max_unpool(x, indexes[3], sizes[2])
    x = F.relu(self.bn41(self.conv41(x)))
    x = F.relu(self.bn42(self.conv42(x)))
    x = F.relu(self.bn43(self.conv43(x)))

    # Stage 3
    x = self.max_unpool(x, indexes[2], sizes[1])
    x = F.relu(self.bn31(self.conv31(x)))
    x = F.relu(self.bn32(self.conv32(x)))
    x = F.relu(self.bn33(self.conv33(x)))

    # Stage 2
    x = self.max_unpool(x, indexes[1], sizes[0])
    x = F.relu(self.bn21(self.conv21(x)))
    x = F.relu(self.bn22(self.conv22(x)))

    # Stage 1
    x = self.max_unpool(x, indexes[0])
    x = F.relu(self.bn11(self.conv11(x)))
    x = self.bn12(self.conv12(x))
    return x
```

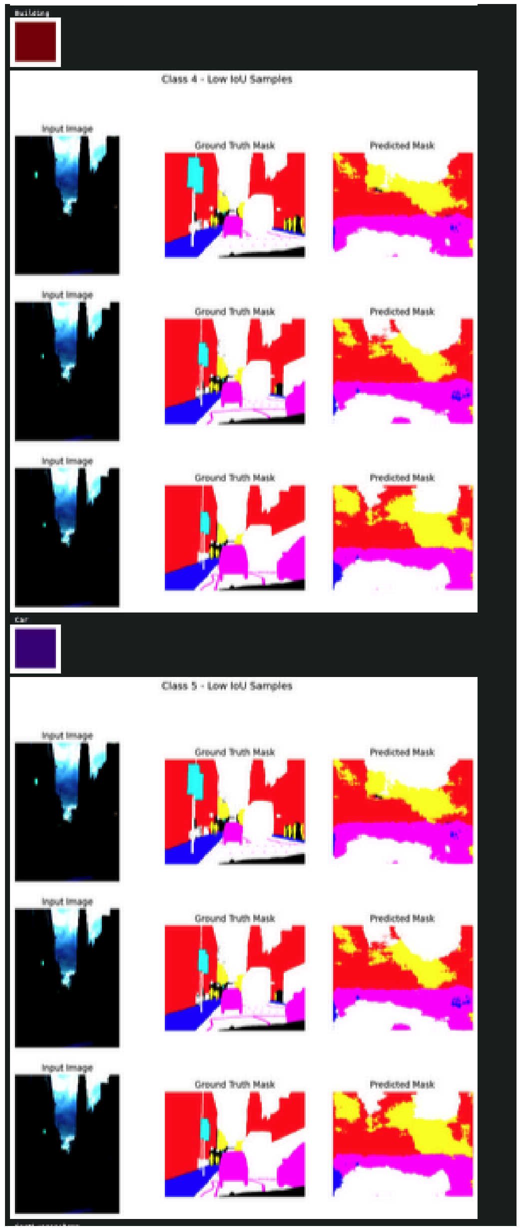




2.b. Class-wise Metrics:

```
Class-wise Metrics:

Pixel Accuracy:
 [0.99994258 0.999789   0.99376579 0.99943686 0.84859203 0.90231407
 0.99966952 0.99973829 0.99095973 0.98462023 0.98568277 0.99997281
 0.99529424 0.99981207 0.9969863  0.99654791 0.99316207 0.91249718
 0.99787885 0.94148335 0.99895761 0.90861947 0.99042173 0.99999042
 0.99634743 1.         0.8727563  0.98854466 1.         0.99406716
 0.96090687 0.98206897]

Dice Coefficient:
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 6.01064571e-01 2.23116690e-01 0.00000000e+00 0.00000000e+00
 0.00000000e+00 8.86412539e-02 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 2.25646664e-05 8.03041374e-01 0.00000000e+00 4.02360001e-01
 0.00000000e+00 7.21694005e-01 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 4.15866269e-01 0.00000000e+00
 0.00000000e+00 0.00000000e+00 4.73607309e-03 4.80038269e-02]

IoU:
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 4.63974580e-01 1.36038030e-01 0.00000000e+00 0.00000000e+00
 0.00000000e+00 5.77230299e-02 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 1.12877694e-05 6.85680427e-01 0.00000000e+00 2.90545638e-01
 0.00000000e+00 5.79187948e-01 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 3.01099223e-01 0.00000000e+00
 0.00000000e+00 0.00000000e+00 2.40425754e-03 2.84312118e-02]

Mean IoU: 0.07953423856729973
```

2.c. Visualisation of Low IoU Predictions



The prediction from the model is very inaccurate as visible in the above image. This is probably because the data to train on was very limited, with only around 350 images. The model possibly could not have given amazing results for small number of epochs. Other reasons might be because of blending with the environment, occlusion, illumination differences etc.

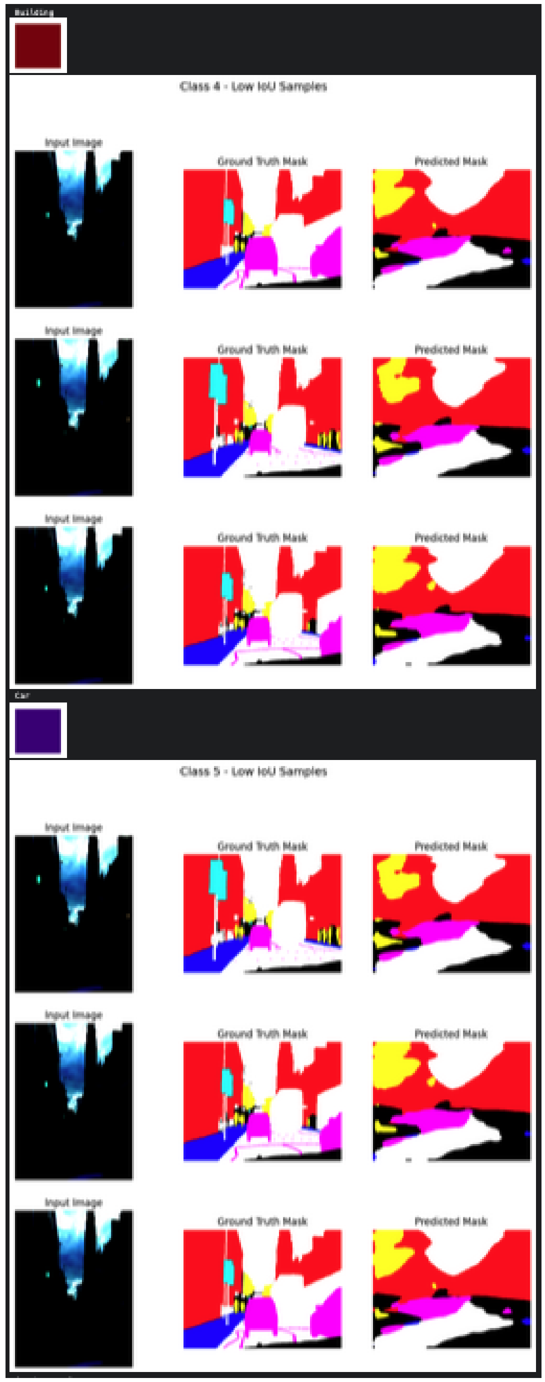
3.a. DeepLabV3 Model

```
class DeepLabV3(nn.Module):
    def __init__(self, num_classes=32):
        super(DeepLabV3, self).__init__()
        self.model = models.segmentation.deeplabv3_resnet50(pretrained=True) # TODO: Initialize
        self.model.classifier[4] = nn.Conv2d(256, num_classes, kernel_size=1, stride=1) #  shou

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

3.b. Class-wise Metrics:

```
Class-wise Metrics:

Pixel Accuracy:
 [0.99993821 0.99973664 0.99484794 0.99912154 0.90938917 0.966486
 0.9996538  0.99974831 0.98760823 0.98673731 0.98509788 0.99997281
 0.99462464 0.99981207 0.99678874 0.99615643 0.98723098 0.94412793
 0.99764904 0.96463819 0.99909131 0.95638931 0.98881683 0.99999042
 0.99695684 1.         0.92709194 0.98742799 1.         0.98842231
 0.95671745 0.98219072]

Dice Coefficient:
 [0.         0.00645222 0.08337753 0.00779683 0.71740387 0.46088294
 0.         0.05954339 0.03092904 0.16449333 0.08870939 0.
 0.09002102 0.         0.07536765 0.07327604 0.10761089 0.87756635
 0.03300915 0.53687947 0.05292777 0.86943698 0.08499903 0.
 0.12861831 0.         0.56176247 0.06892371 0.         0.10317588
 0.34297399 0.23134094]

IoU:
 [0.         0.00358847 0.06342907 0.00503741 0.60011898 0.34147477
 0.         0.04302742 0.01699828 0.1244417  0.0558664  0.
 0.0640624  0.         0.05257994 0.04899171 0.06702823 0.78741336
 0.02513299 0.41947599 0.03859929 0.77542307 0.05473002 0.
 0.09515911 0.         0.45183181 0.05194746 0.         0.06725656
 0.22239188 0.15811525]

Mean IoU: 0.1448162996292732
```

3.c. Visualization of Low IoU Predictions



The prediction from this model is better than before, but not really any good. Here again it probably is because of the small training data. Other reasons might be because of blending with the environment, occlusion, illumination differences etc.

Q4) Object Detection

1.b. mAP values

```
mAP@50: 0.07126384992975199
mAP@50-95: 0.05254300924462508 6
```

1.c. TIDE predictions

```
-- predictions --

bbox AP @ 50: 6.41

                         Main Errors
=============================================================
  Type      Cls      Loc     Both     Dupe      Bkg     Miss
-------------------------------------------------------------
   dAP    35.16     0.51     0.02     0.04     0.08     3.89
=============================================================

        Special Error
=============================
  Type   FalsePos   FalseNeg
-----------------------------
   dAP       0.79       6.95
=============================
```

Average prediction is very low, model is not able to correctly label detected objects, but is able to place bounding boxes with good estimate. But the model is missing a lot of objects