CSCI677:HW5

Nisha Tiwari | nishatiw@usc.edu| uscid-7888495181

Results summary

Model	Mean IOU(%)	Dice(%)	#Epochs (approx)
FCN-32	45.59	36.55	15
FCN-16	48.28	38.00	25

1. FCN-16 is better able to capture the finer details compared to FCN-32
2. FCN-16 has ~3% more accuracy (for both IoU and Dice) than FCN-32

3. FCN-16 performs well (dice or IOU>40%) for 12 classes.
4. FCN-32 performs well (dice or IOU>35%) for 5 classes.
5. Both models have the worst performance in segmenting the bicycle and the chair classes.
6. The best performance for both variations is achieved in segmenting the bus, the train, and the cat classes

Hyperparameters choices and their effects

1. All training images were resized to 256*256.
2. The batch size was fixed to 5.
3. Starting from a learning rate of 0.001, it was further reduced by a factor of ⅕ as the overfitting starts to happen.
4. The image size affects the learning rate. When I resized the images to 500, the training time was slower for each epoch, but learning was faster (FCN32 achieves an accuracy of .35 dice score in just 3 epochs). For 256 sized image, it takes around 10 epochs to achieve similar accuracy.
5. Lowering the learning rate at the point of overfitting helps in boosting accuracy.

Class-wise Accuracy

Following map defines the index vs class name

(0=background, 1=aeroplane, 2=bicycle, 3=bird, 4=boat, 5=bottle, 6=bus, 7=car , 8=cat, 9=chair, 10=cow, 11=diningtable, 12=dog, 13=horse, 14=motorbike, 15=person, 16=potted plant, 17=sheep, 18=sofa, 19=train, 20=tv/monitor)

FCN32

(Classes in bold have measure>0.35 and red have measure<0.1)

Dice For each class	**Mean IOU for each class**
class 0 : 0.60 class 1 : 0.33 class 2 : 0.07 class 3 : 0.28 class 4 : 0.24 class 5 : 0.34 **class 6 : 0.42** **class 7 : 0.35** class 8 : 0.42 class 9 : 0.06 class 10 : 0.17 **class 11 : 0.37** class 12 : 0.30	class 0 : 0.77 class 1 : 0.27 class 2 : 0.04 class 3 : 0.23 class 4 : 0.20 class 5 : 0.30 class 6 : **0.42** class 7 : 0.33 class 8 : 0.42 class 9 : 0.04 class 10 : 0.13 class 11 : **0.36** class 12 : 0.26

class 13 : 0.27	class 13 : 0.22
class 14 : 0.32	class 14 : 0.27
class 15 : 0.37	class 15 : **0.35**
class 16 : 0.09	class 16 : 0.06
class 17 : 0.31	class 17 : 0.26
class 18 : 0.14	class 18 : 0.10
class 19 : 0.40	class 19 : **0.39**
class 20 : 0.33	class 20 : 0.29

FCN16

(Classes in bold have measure>0.40 and red have measure<0.15)

Dice For each class	**Mean IOU for each class**
class 0 : 0.62	class 0 : 0.83
class 1 : 0.41	**class 1 : 0.42**
class 2 : 0.14	class 2 : 0.09
class 3 : 0.41	**class 3 : 0.41**
class 4 : 0.39	class 4 : 0.38
class 5 : 0.42	**class 5 : 0.44**
class 6 : 0.54	**class 6 : 0.64**
class 7 : 0.48	**class 7 : 0.56**
class 8 : 0.51	**class 8 : 0.58**
class 9 : 0.16	class 9 : 0.12
class 10 : 0.38	**class 10 : 0.40**
class 11 : 0.36	class 11 : 0.36
class 12 : 0.46	**class 12 : 0.50**
class 13 : 0.43	**class 13 : 0.43**
class 14 : 0.44	**class 14 : 0.43**
class 15 : 0.44	**class 15 : 0.45**
class 16 : 0.31	class 16 : 0.29
class 17 : 0.40	**class 17 : 0.41**
class 18 : 0.32	class 18 : 0.31
class 19 : 0.49	**class 19 : 0.55**
class 20 : 0.39	**class 20 : 0.40**

Discussion

1. The poor performance for the bicycle and the chair class is probably due to the fine details present in the objects of both the classes. For example, spikes of the bicycle and legs of the chair are thin.

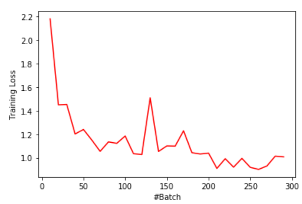
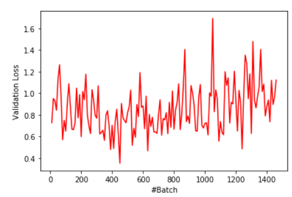
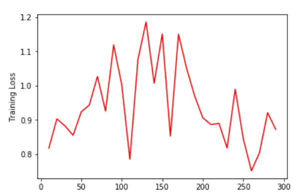
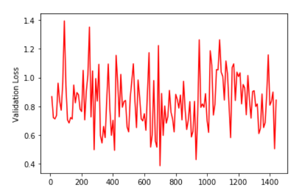
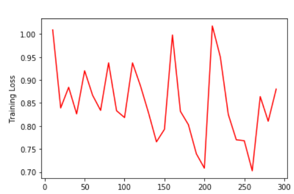
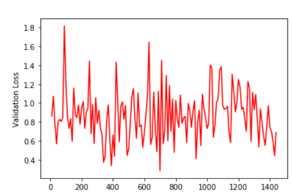
2. As per my observation, the learning is slow for this specific task. To train it to reach 75% mean IOU for training set while avoiding overfitting (by lowering down the learning rates) takes more than 25 epochs.
3. Data augmentation might help in boosting the accuracy. For this task, we have fixed the size of every image. It might be interesting to see the results if we add random crops of the same images to the data set.

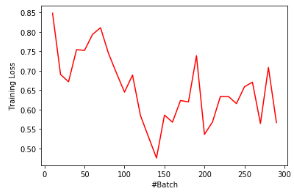
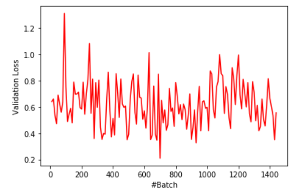
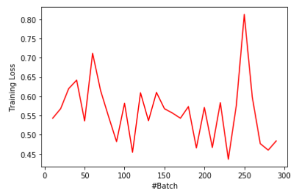
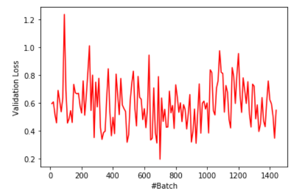
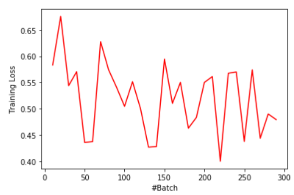
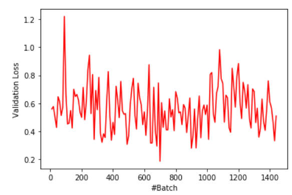
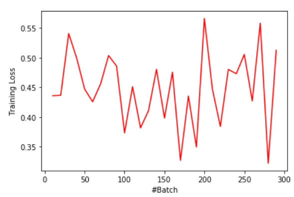
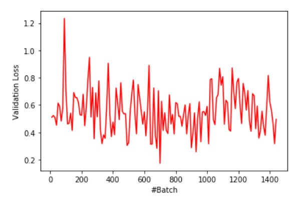
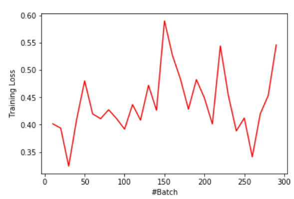
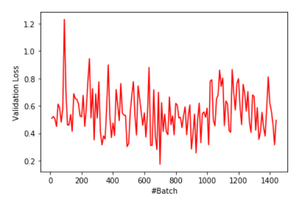
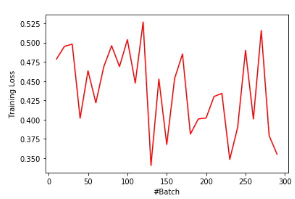
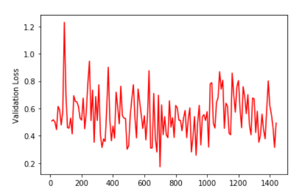
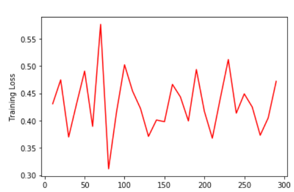
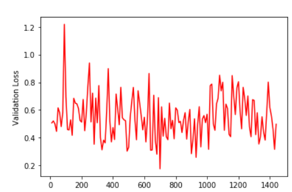
The rest of the report is organized in the following manner.

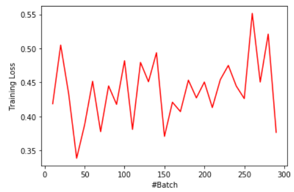
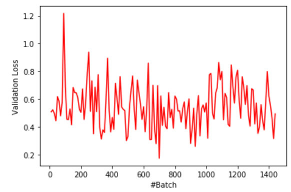
1. First section contains the tables detailing the evolution of the loss functions with accuracy metrics for each epoch for both models.
2. Example section to display the output of FCN-32 and FCN-16 on a few instances of the various classes.
3. Code description

Evolution of loss function

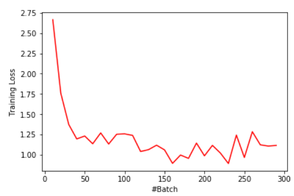
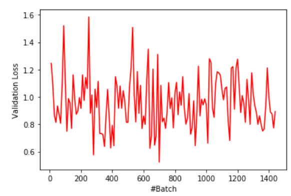
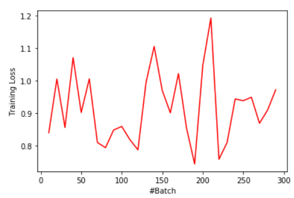
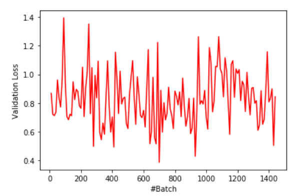
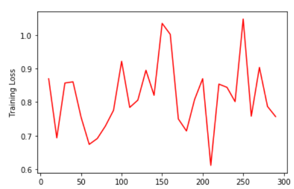
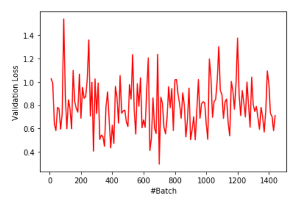
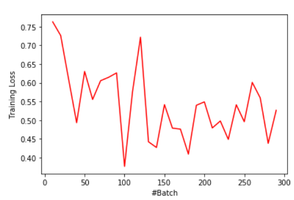
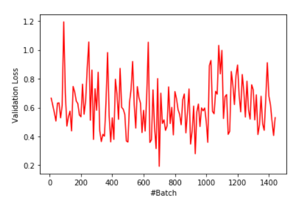
FCN-32

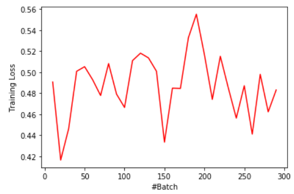
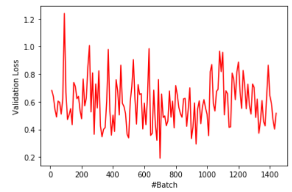
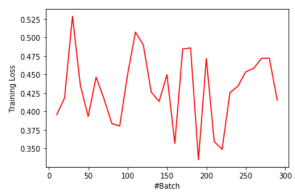
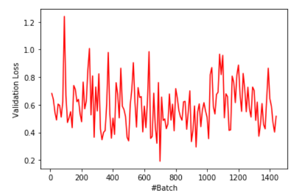
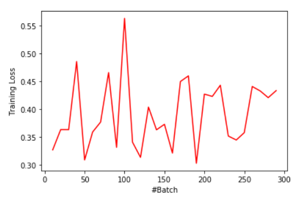
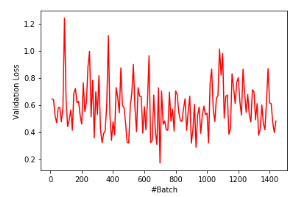
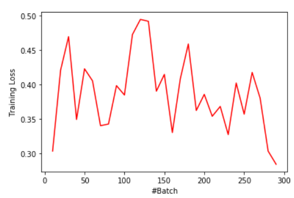
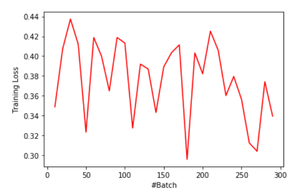
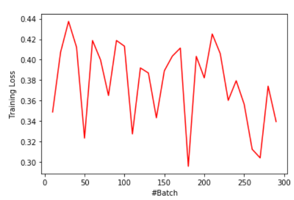
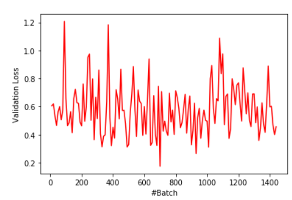
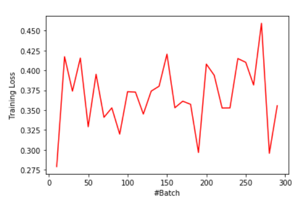
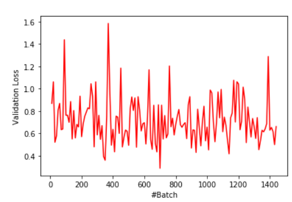
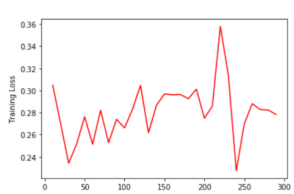
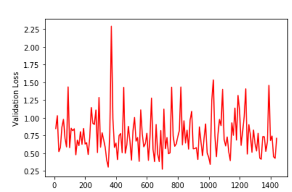
Learning rate	**Training loss**	**Validation loss**	**IOU train**	**Dice train**	**IOU valid**	**Dice valid**
0.001			35.01	27.54	33.59	26.65
0.001			37.39	29.67	34.97	28.06
0.001			38.59	30.61	36.48	29.16

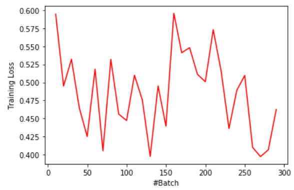
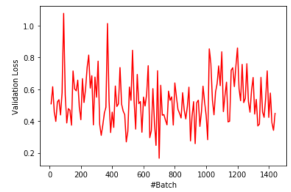
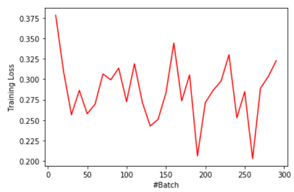
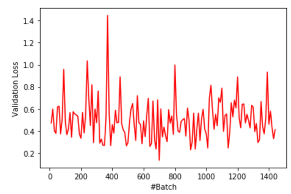
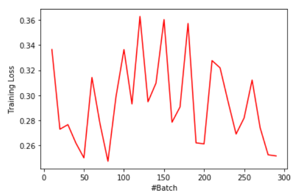
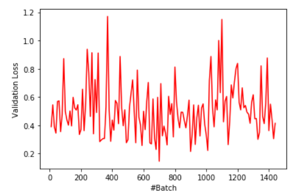
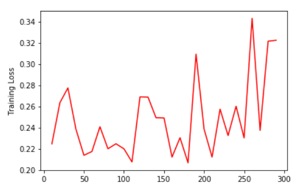
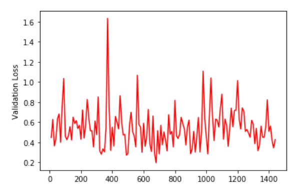
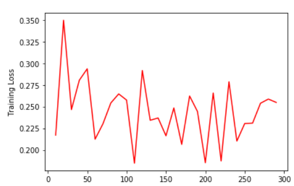
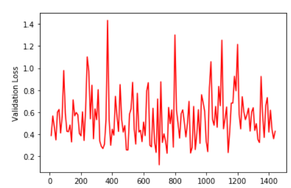
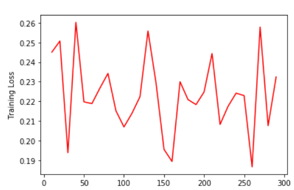
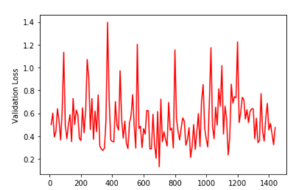
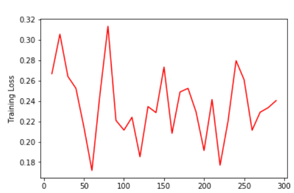
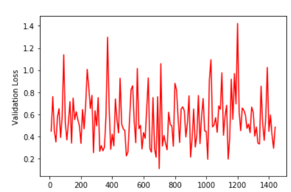
0.0001			48.06	34.58	42.97	34.59
0.0001			49.43	39.14	43.32	34.92
0.0001			51.87	40.75	44.95	36.08
0.0001			53.66	41.85	45.35	36.32
0.0001			53.85	41.98	45.42	36.38
0.0001			53.99	42.11	45.46	36.42
0.0001			54.08	42.16	45.49	36.46

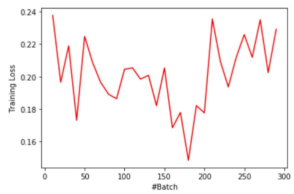
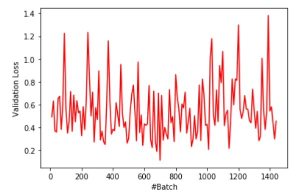
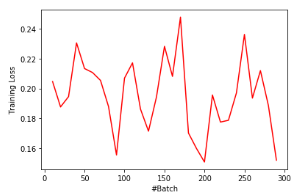
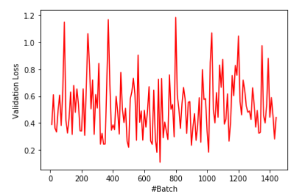
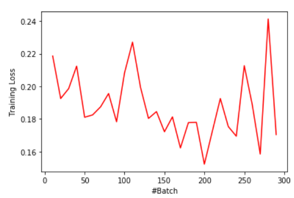
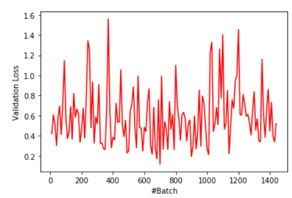
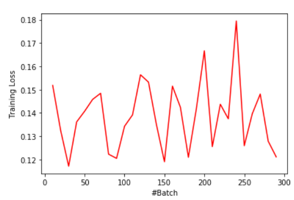
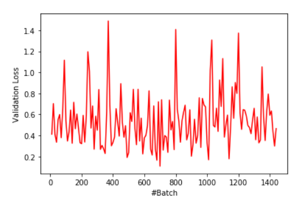
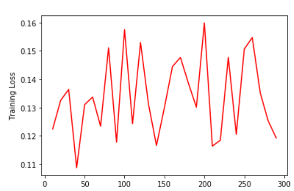
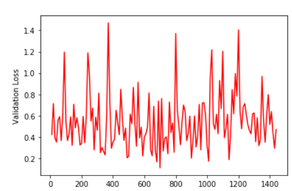
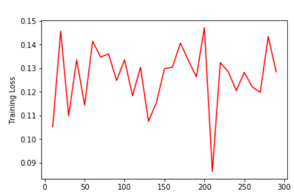
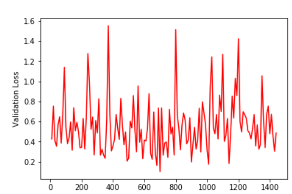
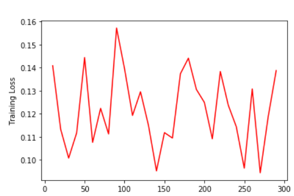
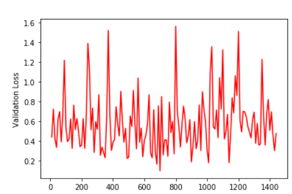
0.0001			54.14	42.23	45.59	36.55

FCN-16

Learning rate	Training loss	Validation loss	IOU train	Dice train	IOU valid	Dice valid
0.001			29.22	23.25	28.27	22.64
0.001			32.17	25.99	30.05	24.44
0.001			37.27	29.82	34.76	28.09
0.0001			48.61	38.22	41.30	33.30

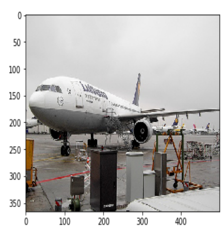
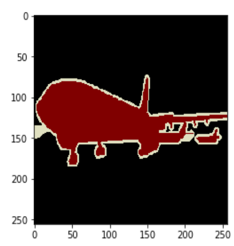
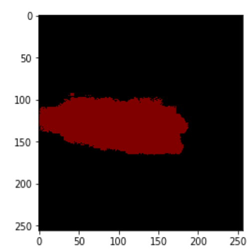
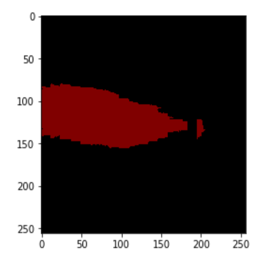
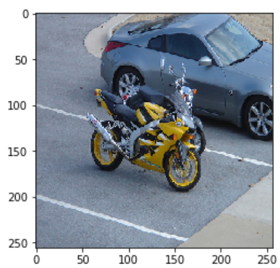
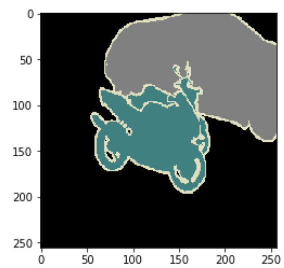
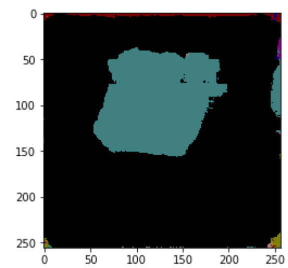
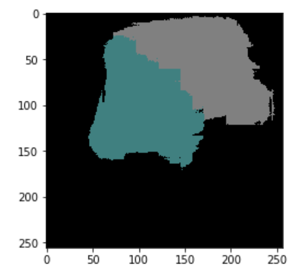
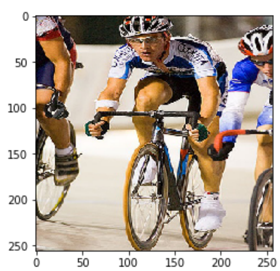
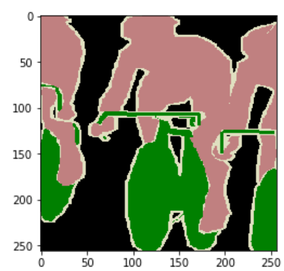
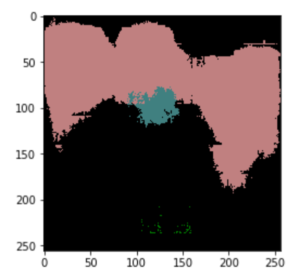
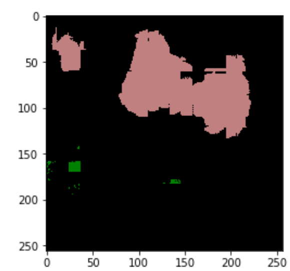
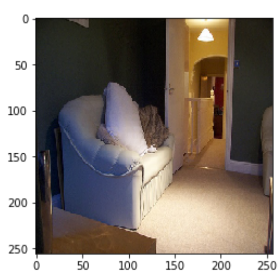
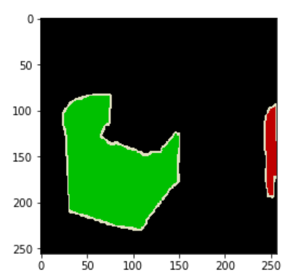
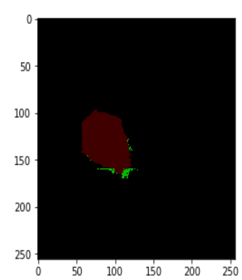
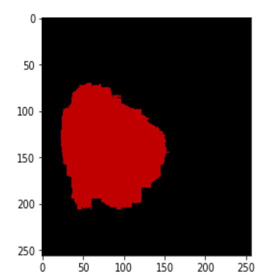
0.0001			50.73	39.73	41.91	33.71
0.0001			52.92	41.13	42.10	33.87
0.0001			54.02	41.91	42.27	34.10
0.00005			55.06	42.58	42.55	34.27
0.00005			55.97	43.19	42.61	34.32
0.00005			56.04	43.78	42.78	34.62
0.00001			66.63	50.48	43.48	35.33

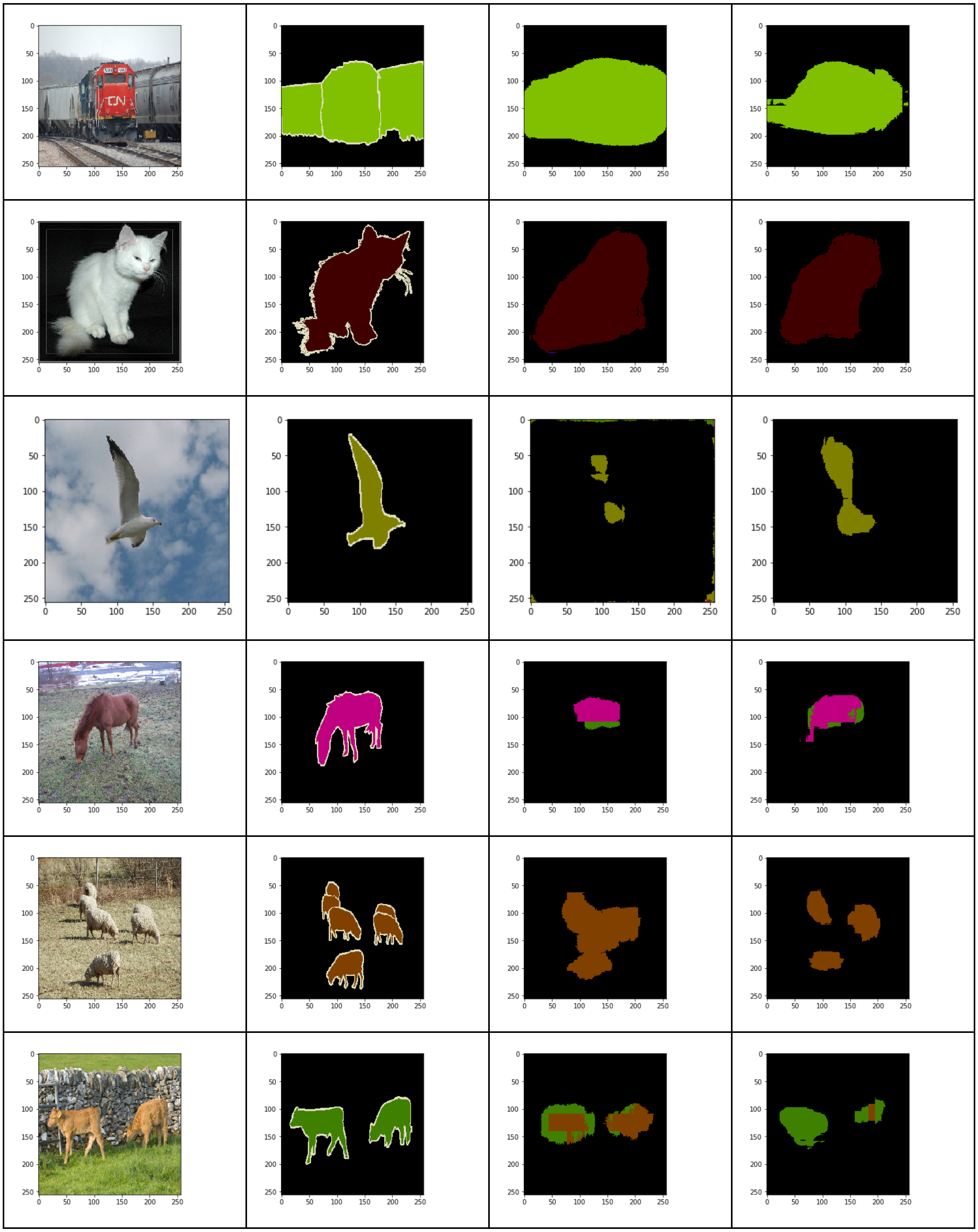
0.00005			52.26	40.89	44.00	35.32
0.00005			60.46	48.08	46.82	37.25
0.00005			62.69	47.79	46.44	36.81
0.0005			61.47	46.77	44.09	35.19
0.0005			65.82	49.91	46.69	37.08
0.0005			66.37	50.43	44.85	35.82
0.0005			65.31	49.45	46.56	36.90

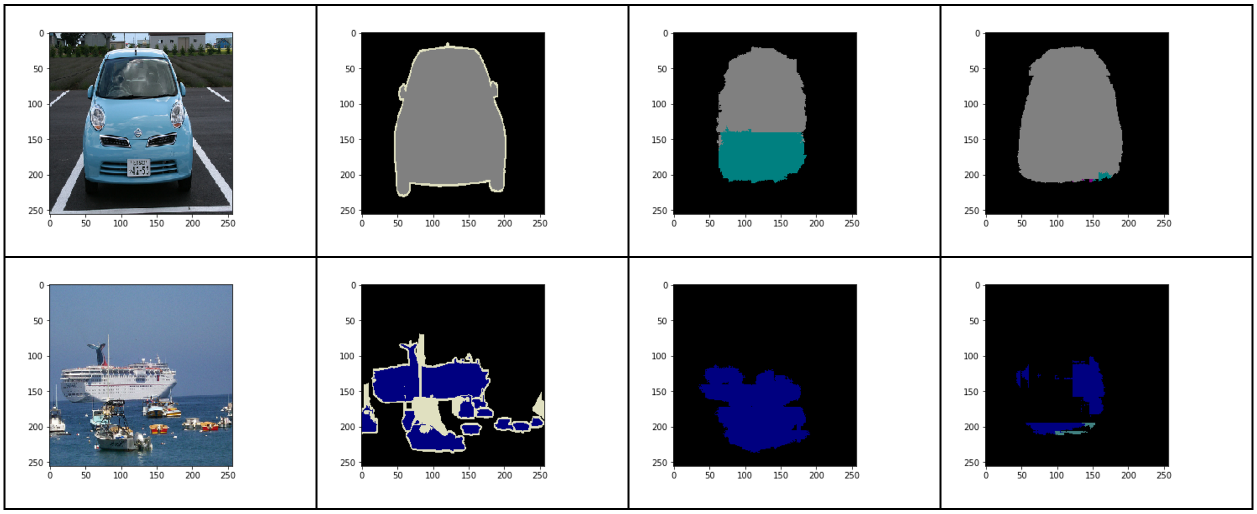
0.0005			69.76	52.37	46.64	37.02
0.0005			71.53	53.51	47.02	37.21
0.0005			72.09	53.74	48.19	38.01
0.00001			77.15	56.82	48.13	37.91
0.00001			77.56	57.05	48.20	37.98
0.000005			78.27	57.33	48.26	37.98
0.000005			78.79	57.61	48.28	38.00

Examples

Following table contains a few randomly selected examples showing both the failure and successful cases.

Original Image	Original Mask	FCN-32 (Predicted Mask)	FCN-16 (Predicted Mask)
			
			
			
			





Code Description

Dataset Creation

The following code creates a Dataset class to split the data into train and valid sets using the ‘split’ parameter passed to the init method. It also applies the provided transformations. The dataset created out of this class are later used to create data loaders.

```
import numpy as np
import os
from PIL import Image
from torch.utils.data import Dataset

def load_image(file):
    return Image.open(file)

def image_path(root, basename, extension):
    return os.path.join(root, f'{basename}{extension}')

class VOC12(Dataset):

    def __init__(self, split, input_transform=None, target_transform=None):
        self.root='/content/drive/My Drive/Colab Notebooks/VOCdevkit/VOC2012'
        self.file_list = os.path.join(self.root, "ImageSets/Segmentation", split + ".txt")
        self.filenames = [line.rstrip() for line in list(open(self.file_list, "r"))]

        self.images_root = os.path.join(self.root, 'JPEGImages')
        self.labels_root = os.path.join(self.root, 'SegmentationClass')

        self.filenames.sort()
        self.input_transform = input_transform
        self.target_transform = target_transform
```

```python
    def __getitem__(self, index):
        filename = self.filenames[index]

        with open(image_path(self.images_root, filename, '.jpg'), 'rb') as f:
            image = load_image(f).convert('RGB')
        with open(image_path(self.labels_root, filename, '.png'), 'rb') as f:
            label = load_image(f).convert('P')

        if self.input_transform is not None:
            image = self.input_transform(image)
        if self.target_transform is not None:
            label = self.target_transform(label)

        return image, label

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

Data Transformations

Relabel class is to change the pixels with value 255 to 0 so that the boundary of the masks is changed to the background class.

```python
class Relabel:

    def __init__(self, olabel, nlabel):
        self.olabel = olabel
        self.nlabel = nlabel

    def __call__(self, tensor):
        tensor[tensor == self.olabel] = self.nlabel
        return tensor
```

ToLabel class changes the mask to a tensor of size h*w to 1*h*w

```python
class ToLabel:

    def __call__(self, image):
        return torch.from_numpy(np.array(image)).long().unsqueeze(0)
```

Dataloader creation

The following code defines the transforms for both image and mask. These transforms are used while creating data loaders. For training, I have used a data loader with batch size 5. For evaluation of the training and validation set, the batch size is 1.

```python
from torchvision.transforms import Compose, CenterCrop, Normalize
from torchvision.transforms import ToTensor, ToPILImage
NUM_CHANNELS = 3
NUM_CLASSES = 21

image_transform = ToPILImage()
input_transform = Compose([
    #CenterCrop(256),
    transforms.Resize((300,500)),
    ToTensor(),
])
target_transform = Compose([
      #CenterCrop(256),
      transforms.Resize((300,500)),
      ToLabel(),
      Relabel(255, 0),
])
voc =VOC12('train',input_transform, target_transform)
valid=VOC12('val',input_transform, target_transform)
train_loader = torch.utils.data.DataLoader(voc,
        num_workers=10, batch_size=5, shuffle=True)

train_eval_loader = torch.utils.data.DataLoader(voc,
        num_workers=10, batch_size=1, shuffle=False)

valid_eval_loader = torch.utils.data.DataLoader(valid,
        num_workers=10, batch_size=1, shuffle=False)
```

Metrics Computation

The following code takes the inputs of predicted pixel labels and true pixel labels for an image and returns mean IOU and Dice value.

```python
from sklearn.metrics import confusion_matrix
import numpy as np

def compute_iou_dice(y_pred, y_true):
    y_pred = y_pred.flatten()
    y_true = y_true.flatten()
    current = confusion_matrix(y_true, y_pred, labels=list(range(0, 21)))
    intersection = np.diag(current)
    ground_truth_set = current.sum(axis=1)
    predicted_set = current.sum(axis=0)
    union = ground_truth_set + predicted_set - intersection
    dice = np.nan_to_num(np.divide(2*intersection,(ground_truth_set +
predicted_set+intersection)))
    IoU = np.nan_to_num(np.divide(intersection , union.astype(np.float32)))

    return np.sum(IoU)/len(np.nonzero(union)[0]), np.sum(dice)/len(np.nonzero(union)[0])
```

Model Creation

FCN-16

I present the following code for FCN16. Apart from the basic init and forward method, it also has separate methods to copy parameters either from pretrained FCN32 and VGG 16. In this report I have presented results with the backbone as VGG16

```
class FCN16s(nn.Module):

    def __init__(self, n_class=21):
        super(FCN16s, self).__init__()
        # conv1
        self.conv1_1 = nn.Conv2d(3, 64, 3, padding=100)
        self.relu1_1 = nn.ReLU(inplace=True)
        self.conv1_2 = nn.Conv2d(64, 64, 3, padding=1)
        self.relu1_2 = nn.ReLU(inplace=True)
        self.pool1 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/2

        # conv2
        self.conv2_1 = nn.Conv2d(64, 128, 3, padding=1)
        self.relu2_1 = nn.ReLU(inplace=True)
        self.conv2_2 = nn.Conv2d(128, 128, 3, padding=1)
        self.relu2_2 = nn.ReLU(inplace=True)
        self.pool2 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/4

        # conv3
        self.conv3_1 = nn.Conv2d(128, 256, 3, padding=1)
        self.relu3_1 = nn.ReLU(inplace=True)
        self.conv3_2 = nn.Conv2d(256, 256, 3, padding=1)
        self.relu3_2 = nn.ReLU(inplace=True)
        self.conv3_3 = nn.Conv2d(256, 256, 3, padding=1)
        self.relu3_3 = nn.ReLU(inplace=True)
        self.pool3 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/8

        # conv4
        self.conv4_1 = nn.Conv2d(256, 512, 3, padding=1)
        self.relu4_1 = nn.ReLU(inplace=True)
        self.conv4_2 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu4_2 = nn.ReLU(inplace=True)
        self.conv4_3 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu4_3 = nn.ReLU(inplace=True)
        self.pool4 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/16

        # conv5
        self.conv5_1 = nn.Conv2d(512, 512, 3, padding=1)
```

```
        self.relu5_1 = nn.ReLU(inplace=True)
        self.conv5_2 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu5_2 = nn.ReLU(inplace=True)
        self.conv5_3 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu5_3 = nn.ReLU(inplace=True)
        self.pool5 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/32

        self.adapLayer = nn.AdaptiveAvgPool2d(output_size=(7, 7))

        # fc6
        self.fc6 = nn.Conv2d(512, 4096, 7)
        self.relu6 = nn.ReLU(inplace=True)
        self.drop6 = nn.Dropout2d()

       # fc7
        self.fc7 = nn.Conv2d(4096, 4096, 1)
        self.relu7 = nn.ReLU(inplace=True)
        self.drop7 = nn.Dropout2d()

        self.score_fr = nn.Conv2d(4096, n_class, 1)
        self.score_pool4 = nn.Conv2d(512, n_class, 1)

        self.upscore2 = nn.ConvTranspose2d(
            n_class, n_class, 4, stride=2, bias=False)
        self.upscore16 = nn.ConvTranspose2d(
            n_class, n_class, 32, stride=16, bias=False)


    def forward(self, x):
        h = x
        h = self.relu1_1(self.conv1_1(h))
        h = self.relu1_2(self.conv1_2(h))
        h = self.pool1(h)

        h = self.relu2_1(self.conv2_1(h))
        h = self.relu2_2(self.conv2_2(h))
        h = self.pool2(h)

        h = self.relu3_1(self.conv3_1(h))
        h = self.relu3_2(self.conv3_2(h))
        h = self.relu3_3(self.conv3_3(h))
        h = self.pool3(h)

        h = self.relu4_1(self.conv4_1(h))
        h = self.relu4_2(self.conv4_2(h))
        h = self.relu4_3(self.conv4_3(h))
        h = self.pool4(h)
        pool4 = h

        h = self.relu5_1(self.conv5_1(h))
```

```
    h = self.relu5_2(self.conv5_2(h))
    h = self.relu5_3(self.conv5_3(h))
    h = self.pool5(h)

    h = self.relu6(self.fc6(h))
    h = self.drop6(h)

    h = self.relu7(self.fc7(h))
    h = self.drop7(h)

    h = self.score_fr(h)
    h = self.upscore2(h)
    upscore2 = h  # 1/16

    h = self.score_pool4(pool4)

    h=F.upsample_bilinear(h, upscore2.size()[2:])

    h = upscore2 + h

    h = self.upscore16(h)
    h=F.upsample_bilinear(h, x.size()[2:])

    return h

def copy_params_from_fcn32s(self, fcn32s):
    for name, l1 in fcn32s.named_children():
        try:
            l2 = getattr(self, name)
            l2.weight  # skip ReLU / Dropout
        except Exception:
            continue
        assert l1.weight.size() == l2.weight.size()
        assert l1.bias.size() == l2.bias.size()
        l2.weight.data.copy_(l1.weight.data)
        l2.bias.data.copy_(l1.bias.data)

def copy_params_from_vgg16(self, vgg16):
  features = [
      self.conv1_1, self.relu1_1,
      self.conv1_2, self.relu1_2,
      self.pool1,
      self.conv2_1, self.relu2_1,
      self.conv2_2, self.relu2_2,
      self.pool2,
      self.conv3_1, self.relu3_1,
      self.conv3_2, self.relu3_2,
      self.conv3_3, self.relu3_3,
      self.pool3,
      self.conv4_1, self.relu4_1,
```

```
            self.conv4_2, self.relu4_2,
            self.conv4_3, self.relu4_3,
            self.pool4,
            self.conv5_1, self.relu5_1,
            self.conv5_2, self.relu5_2,
            self.conv5_3, self.relu5_3,
            self.pool5,
        ]
        for l1, l2 in zip(vgg16.features, features):
            if isinstance(l1, nn.Conv2d) and isinstance(l2, nn.Conv2d):
                assert l1.weight.size() == l2.weight.size()
                assert l1.bias.size() == l2.bias.size()
                l2.weight.data = l1.weight.data
                l2.bias.data = l1.bias.data
```

FCN-32

```
class FCN32s(nn.Module):

    def __init__(self, n_class=21):
        super(FCN32s, self).__init__()
        # conv1
        self.conv1_1 = nn.Conv2d(3, 64, 3, padding=100)
        self.relu1_1 = nn.ReLU(inplace=True)
        self.conv1_2 = nn.Conv2d(64, 64, 3, padding=1)
        self.relu1_2 = nn.ReLU(inplace=True)
        self.pool1 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/2

        # conv2
        self.conv2_1 = nn.Conv2d(64, 128, 3, padding=1)
        self.relu2_1 = nn.ReLU(inplace=True)
        self.conv2_2 = nn.Conv2d(128, 128, 3, padding=1)
        self.relu2_2 = nn.ReLU(inplace=True)
        self.pool2 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/4

        # conv3
        self.conv3_1 = nn.Conv2d(128, 256, 3, padding=1)
        self.relu3_1 = nn.ReLU(inplace=True)
        self.conv3_2 = nn.Conv2d(256, 256, 3, padding=1)
      self.relu3_2 = nn.ReLU(inplace=True)
        self.conv3_3 = nn.Conv2d(256, 256, 3, padding=1)
        self.relu3_3 = nn.ReLU(inplace=True)
        self.pool3 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/8

        # conv4
        self.conv4_1 = nn.Conv2d(256, 512, 3, padding=1)
        self.relu4_1 = nn.ReLU(inplace=True)
        self.conv4_2 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu4_2 = nn.ReLU(inplace=True)
```

```
        self.conv4_3 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu4_3 = nn.ReLU(inplace=True)
        self.pool4 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/16

        # conv5
        self.conv5_1 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu5_1 = nn.ReLU(inplace=True)
        self.conv5_2 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu5_2 = nn.ReLU(inplace=True)
        self.conv5_3 = nn.Conv2d(512, 512, 3, padding=1)
        self.relu5_3 = nn.ReLU(inplace=True)
        self.pool5 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  # 1/32

        #self.adapLayer = nn.AdaptiveAvgPool2d(output_size=(7, 7))

        # fc6
        self.fc6 = nn.Conv2d(512, 4096, 7)
        self.relu6 = nn.ReLU(inplace=True)
        self.drop6 = nn.Dropout2d()

        # fc7
        self.fc7 = nn.Conv2d(4096, 4096, 1)
        self.relu7 = nn.ReLU(inplace=True)
        self.drop7 = nn.Dropout2d()

        self.score_fr = nn.Conv2d(4096, n_class, 1)
        self.upscore = nn.ConvTranspose2d(n_class, n_class, 64, stride=32,
                                          bias=False)

    def forward(self, x):
        h = x
        h = self.relu1_1(self.conv1_1(h))
        h = self.relu1_2(self.conv1_2(h))
        h = self.pool1(h)

        h = self.relu2_1(self.conv2_1(h))
        h = self.relu2_2(self.conv2_2(h))
        h = self.pool2(h)

        h = self.relu3_1(self.conv3_1(h))
        h = self.relu3_2(self.conv3_2(h))
        h = self.relu3_3(self.conv3_3(h))
        h = self.pool3(h)

        h = self.relu4_1(self.conv4_1(h))
        h = self.relu4_2(self.conv4_2(h))
        h = self.relu4_3(self.conv4_3(h))
        h = self.pool4(h)

        h = self.relu5_1(self.conv5_1(h))
```

```
        h = self.relu5_2(self.conv5_2(h))
        h = self.relu5_3(self.conv5_3(h))
        h = self.pool5(h)

        #h=self.adapLayer(h)

        h = self.relu6(self.fc6(h))
        h = self.drop6(h)

        h = self.relu7(self.fc7(h))
        h = self.drop7(h)

        h = self.score_fr(h)
        h = self.upscore(h)
        h = h[:, :, 19:19 + x.size()[2], 19:19 + x.size()[3]].contiguous()
        return h

    def copy_params_from_vgg16(self, vgg16):
        features = [
            self.conv1_1, self.relu1_1,
            self.conv1_2, self.relu1_2,
            self.pool1,
            self.conv2_1, self.relu2_1,
            self.conv2_2, self.relu2_2,
            self.pool2,
            self.conv3_1, self.relu3_1,
            self.conv3_2, self.relu3_2,
            self.conv3_3, self.relu3_3,
            self.pool3,
            self.conv4_1, self.relu4_1,
            self.conv4_2, self.relu4_2,
            self.conv4_3, self.relu4_3,
            self.pool4,
            self.conv5_1, self.relu5_1,
            self.conv5_2, self.relu5_2,
            self.conv5_3, self.relu5_3,
            self.pool5,
        ]
        for l1, l2 in zip(vgg16.features, features):
            if isinstance(l1, nn.Conv2d) and isinstance(l2, nn.Conv2d):
                assert l1.weight.size() == l2.weight.size()
                assert l1.bias.size() == l2.bias.size()
                l2.weight.data = l1.weight.data
                l2.bias.data = l1.bias.data
                l2.weight.requires_grad = False
                l2.bias.requires_grad = False
```

Results visualization

Following method takes a labeled mask as input and uses label_to_color_image method (from the link provided in assignment pdf) to map each label to the corresponding color. After this mapping, it plots the image.

```
def printLabeledImage(lbl):
  lbl=lbl.numpy()
  np.squeeze(lbl, axis=0)
  img=label_to_color_image(lbl)
  pyplot.imshow(img)
```

Training and Validation

```
FCN32=FCN32.cuda()
max_epochs=5
criterion = CrossEntropyLoss2d()
learning_rate=0.001
momentum = 0.99
optimizer = optim.Adam(FCN32.parameters(), lr=learning_rate)
#optimizer = optim.SGD(FCN16.parameters(), lr=1.0e-5, momentum=0.99)

epoch_count=0
IOU=0
for epoch in range(max_epochs):
    epoch_count+=1
    # Training
    #print(epoch)
    loss_list=[]
    num_batch_list=[]
    num_batches=0
    final_loss=[]
    FCN32.train()
    for local_batch, local_labels in loader:
      # Transfer to GPU
      optimizer.zero_grad()
      local_batch, local_labels =  torch.autograd.Variable(local_batch.cuda()),
torch.autograd.Variable(local_labels.cuda())
      output = FCN32(local_batch)
      loss = criterion(output, local_labels[:,0])
      loss_list.append(loss.item())
      loss.backward()
      optimizer.step()
      num_batches+=1
      if num_batches % 10 == 0:
            num_batch_list.append(num_batches)
            final_loss.append(np.mean(loss_list))
            loss_list.clear()
```

```
# plotting training and validation accuracies
fig2 = pyplot.figure()
pyplot.plot(num_batch_list, final_loss, 'r')
pyplot.xlabel("#Batch")
pyplot.ylabel("Training Loss")
pyplot.show(fig2)

FCN32.eval()
IOU=0
Dice=0
num_batches=0
loss_list=[]
num_batch_list=[]
final_loss=[]

for local_batch, local_labels in train_loader:
  # Transfer to GPU
  local_batch=  torch.autograd.Variable(local_batch.cuda())
  local_labels=  torch.autograd.Variable(local_labels.cuda())
  # Model computations
  predicted_val = FCN32(local_batch)
  loss = criterion(predicted_val, local_labels[:,0])
  predicted_val=nn.functional.log_softmax(predicted_val, dim=1)
  predicted_val=torch.argmax(predicted_val, dim=1)
  predicted_val = predicted_val.cpu().data.numpy()
  loss_list.append(loss.item())
  num_batches+=1
  if num_batches % 10 == 0:
        num_batch_list.append(num_batches)
        final_loss.append(np.mean(loss_list))
        loss_list.clear()
  currentIou, currentDice=compute_iou(predicted_val, local_labels.cpu().numpy())
  IOU=IOU+currentIou
  Dice=Dice+currentDice
  training_accuracy.append(IOU/num_batches)
print("Training:::for epoch", epoch, "IOU is:",IOU/num_batches)
print("Training:::for epoch", epoch, "Dice is:",Dice/num_batches)
fig2 = pyplot.figure()
pyplot.plot(num_batch_list, final_loss, 'r')
pyplot.xlabel("#Batch")
pyplot.ylabel("Validation Loss")
pyplot.show(fig2)

FCN32.eval()
IOU=0
Dice=0
num_batches=0
loss_list=[]
num_batch_list=[]
final_loss=[]
```

```python
for local_batch, local_labels in valid_loader:
  # Transfer to GPU
  local_batch=  torch.autograd.Variable(local_batch.cuda())
  local_labels=  torch.autograd.Variable(local_labels.cuda())
  # Model computations
  predicted_val = FCN32(local_batch)
  loss = criterion(predicted_val, local_labels[:,0])
  predicted_val=nn.functional.log_softmax(predicted_val, dim=1)
  predicted_val=torch.argmax(predicted_val, dim=1)
  predicted_val = predicted_val.cpu().data.numpy()
  loss_list.append(loss.item())
  num_batches+=1
  if num_batches % 10 == 0:
        num_batch_list.append(num_batches)
        final_loss.append(np.mean(loss_list))
        loss_list.clear()
  currentIou, currentDice=compute_iou(predicted_val, local_labels.cpu().numpy())
  IOU=IOU+currentIou
  Dice=Dice+currentDice
  training_accuracy.append(IOU/num_batches)
print("Validation:::for epoch", epoch, "IOU is:",IOU/num_batches)
print("Validation:::for epoch", epoch, "Dice is:",Dice/num_batches)
fig2 = pyplot.figure()
pyplot.plot(num_batch_list, final_loss, 'r')
pyplot.xlabel("#Batch")
pyplot.ylabel("Validation Loss")
pyplot.show(fig2)
```