HOMEWORK 5 REPORT

CSCI 677

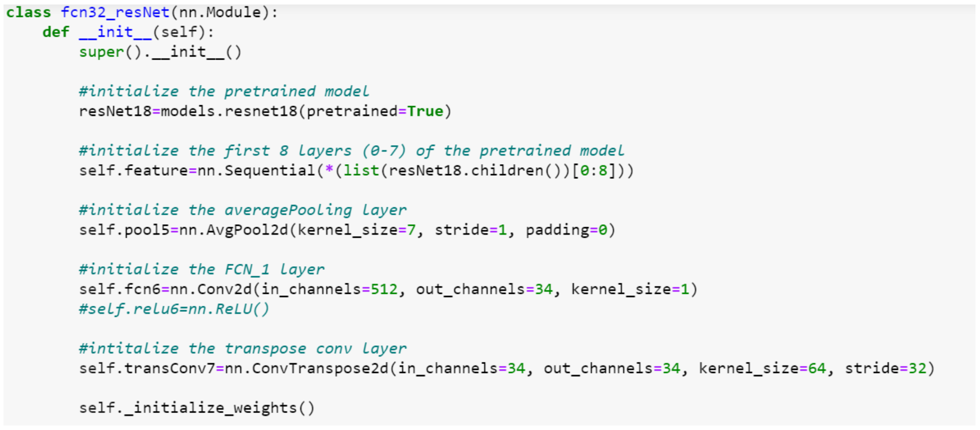
Semantic Segmentation, ResNet, FCN32, FCN16

Hardik Prajapati
USC ID: 2678294168

Parameter	FCN32	FCN16
Epochs	20	20
Initial learning rate	0.001	0.001
Batch size	1	1
Optimizer	Adam()	Adam()
Train+Validation time	8822 sec	9862 sec
mIoU	0.1821	0.2769

1) A brief description of the programs you write, including the source listing.

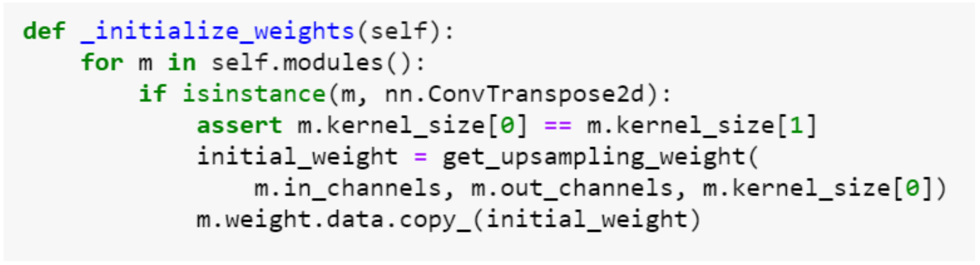
a) Architecture: FCN32



```python
class fcn32_resNet(nn.Module):
    def __init__(self):
        super().__init__()

        #initialize the pretrained model
        resNet18=models.resnet18(pretrained=True)

        #initialize the first 8 layers (0-7) of the pretrained model
        self.feature=nn.Sequential(*(list(resNet18.children())[0:8]))

        #initialize the averagePooling layer
        self.pool5=nn.AvgPool2d(kernel_size=7, stride=1, padding=0)

        #initialize the FCN_1 layer
        self.fcn6=nn.Conv2d(in_channels=512, out_channels=34, kernel_size=1)
        #self.relu6=nn.ReLU()

        #intitialize the transpose conv layer
        self.transConv7=nn.ConvTranspose2d(in_channels=34, out_channels=34, kernel_size=64, stride=32)

        self._initialize_weights()
```

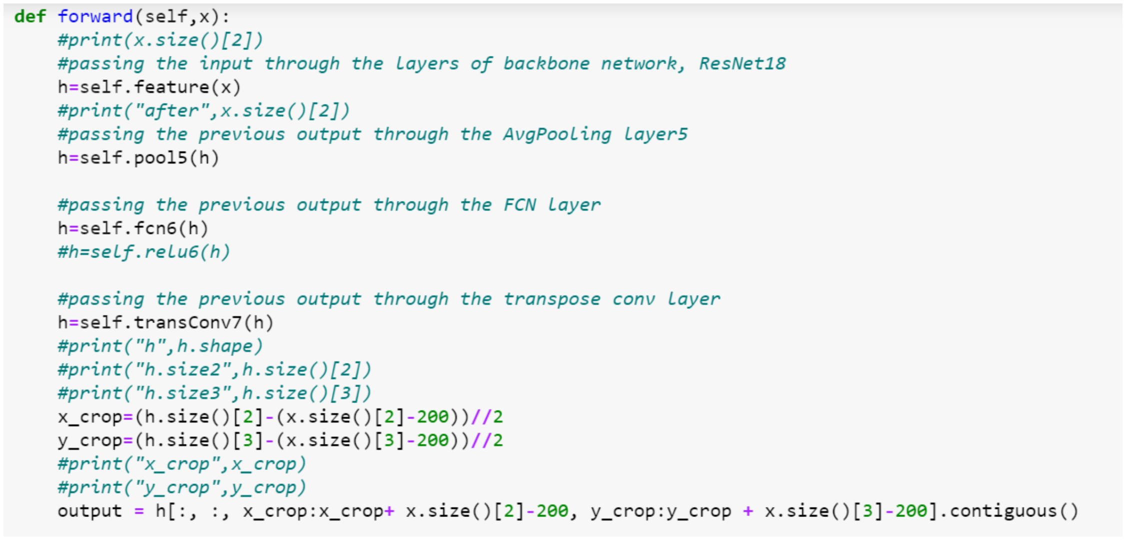
Download the pretrained ResNet Network. Take the first 8 layers(till conv5_x block)from this network building our backbone Network.

Add avgPool layer on top of this and then add FCN (fully convolutional layer). On top of this a transpose convolutional layer is added.



```python
def _initialize_weights(self):
    for m in self.modules():
        if isinstance(m, nn.ConvTranspose2d):
            assert m.kernel_size[0] == m.kernel_size[1]
            initial_weight = get_upsampling_weight(
                m.in_channels, m.out_channels, m.kernel_size[0])
            m.weight.data.copy_(initial_weight)
```

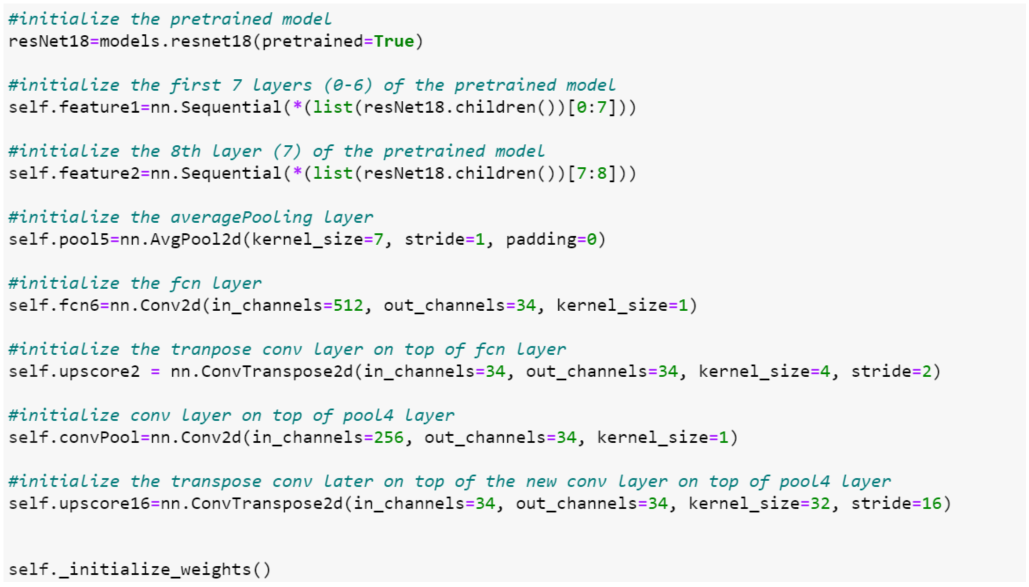
This code is referred from: https://github.com/wkentaro/pytorch-fcn/tree/main/torchfcn/models. I have removed the first ‘if’ loop from the source in order to use the pre-trained weights of the ResNet as our initial weights.



```python
def forward(self,x):
    #print(x.size()[2])
    #passing the input through the layers of backbone network, ResNet18
    h=self.feature(x)
    #print("after",x.size()[2])
    #passing the previous output through the AvgPooling layer5
    h=self.pool5(h)

    #passing the previous output through the FCN layer
    h=self.fcn6(h)
    #h=self.relu6(h)

    #passing the previous output through the transpose conv layer
    h=self.transConv7(h)
    #print("h",h.shape)
    #print("h.size2",h.size()[2])
    #print("h.size3",h.size()[3])
    x_crop=(h.size()[2]-(x.size()[2]-200))//2
    y_crop=(h.size()[3]-(x.size()[3]-200))//2
    #print("x_crop",x_crop)
    #print("y_crop",y_crop)
    output = h[:, :, x_crop:x_crop+ x.size()[2]-200, y_crop:y_crop + x.size()[3]-200].contiguous()
```

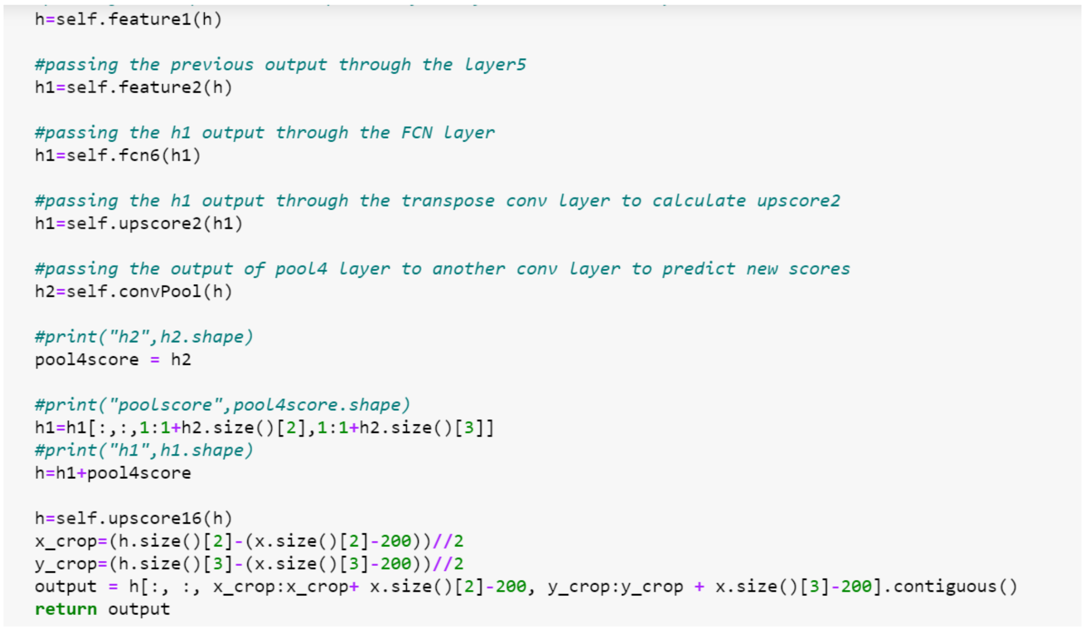
This is the forward function of the architecture. At the end, I crop the output to match the dimension of the Ground truth.

b) Architecture: FCN16



```
#initialize the pretrained model
resNet18=models.resnet18(pretrained=True)

#initialize the first 7 layers (0-6) of the pretrained model
self.feature1=nn.Sequential(*(list(resNet18.children())[0:7]))

#initialize the 8th layer (7) of the pretrained model
self.feature2=nn.Sequential(*(list(resNet18.children())[7:8]))

#initialize the averagePooling layer
self.pool5=nn.AvgPool2d(kernel_size=7, stride=1, padding=0)

#initialize the fcn layer
self.fcn6=nn.Conv2d(in_channels=512, out_channels=34, kernel_size=1)

#initialize the tranpose conv layer on top of fcn layer
self.upscore2 = nn.ConvTranspose2d(in_channels=34, out_channels=34, kernel_size=4, stride=2)

#initialize conv layer on top of pool4 layer
self.convPool=nn.Conv2d(in_channels=256, out_channels=34, kernel_size=1)

#initialize the transpose conv later on top of the new conv layer on top of pool4 layer
self.upscore16=nn.ConvTranspose2d(in_channels=34, out_channels=34, kernel_size=32, stride=16)


self._initialize_weights()
```

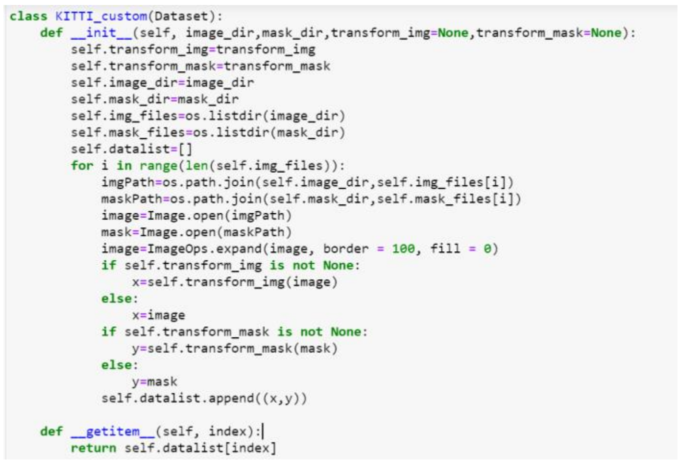
In addition to the FCN32 architecture, we add another convolutional layer on the output of conv4_x block. On top of this another transpose convolutional layer is added.



```
    h=self.feature1(h)

    #passing the previous output through the layer5
    h1=self.feature2(h)

    #passing the h1 output through the FCN layer
    h1=self.fcn6(h1)

    #passing the h1 output through the transpose conv layer to calculate upscore2
    h1=self.upscore2(h1)

    #passing the output of pool4 layer to another conv layer to predict new scores
    h2=self.convPool(h)

    #print("h2",h2.shape)
    pool4score = h2

    #print("poolscore",pool4score.shape)
    h1=h1[:,:,1:1+h2.size()[2],1:1+h2.size()[3]]
    #print("h1",h1.shape)
    h=h1+pool4score

    h=self.upscore16(h)
    x_crop=(h.size()[2]-(x.size()[2]-200))//2
    y_crop=(h.size()[3]-(x.size()[3]-200))//2
    output = h[:, :, x_crop:x_crop+ x.size()[2]-200, y_crop:y_crop + x.size()[3]-200].contiguous()
    return output
```

This is the forward function of FCN16 network. I have cropped at 2 places to match the dimension of tensor. The 2 upscores are added to finally have the output which is trimmed to match the dimension of gt_mask.

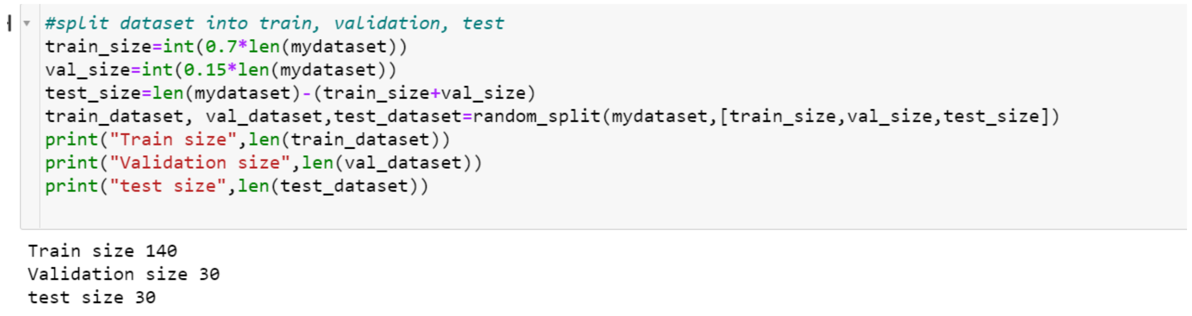
c) Custom Dataset Class



```
class KITTI_custom(Dataset):
    def __init__(self, image_dir,mask_dir,transform_img=None,transform_mask=None):
        self.transform_img=transform_img
        self.transform_mask=transform_mask
        self.image_dir=image_dir
        self.mask_dir=mask_dir
        self.img_files=os.listdir(image_dir)
        self.mask_files=os.listdir(mask_dir)
        self.datalist=[]
        for i in range(len(self.img_files)):
            imgPath=os.path.join(self.image_dir,self.img_files[i])
            maskPath=os.path.join(self.mask_dir,self.mask_files[i])
            image=Image.open(imgPath)
            mask=Image.open(maskPath)
            image=ImageOps.expand(image, border = 100, fill = 0)
            if self.transform_img is not None:
                x=self.transform_img(image)
            else:
                x=image
            if self.transform_mask is not None:
                y=self.transform_mask(mask)
            else:
                y=mask
            self.datalist.append((x,y))

    def __getitem__(self, index):
        return self.datalist[index]
```

This is the dataset class I have created for Kitti dataset. It reads the image files, apply border of 100 pixels on each side of input image and then apply transforms if any. In getitem class, we return the input image and gt_mask at particular index.

d) Dataset splitter

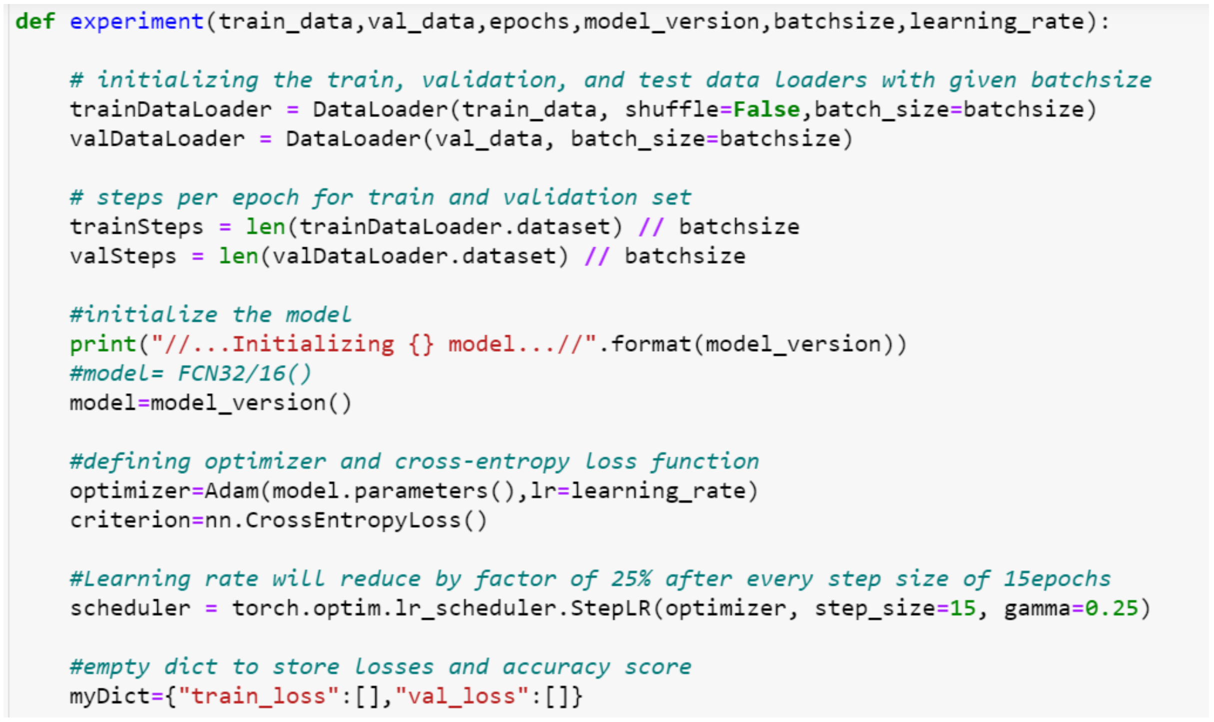


```
#split dataset into train, validation, test
train_size=int(0.7*len(mydataset))
val_size=int(0.15*len(mydataset))
test_size=len(mydataset)-(train_size+val_size)
train_dataset, val_dataset,test_dataset=random_split(mydataset,[train_size,val_size,test_size])
print("Train size",len(train_dataset))
print("Validation size",len(val_dataset))
print("test size",len(test_dataset))
```

```
Train size 140
Validation size 30
test size 30
```

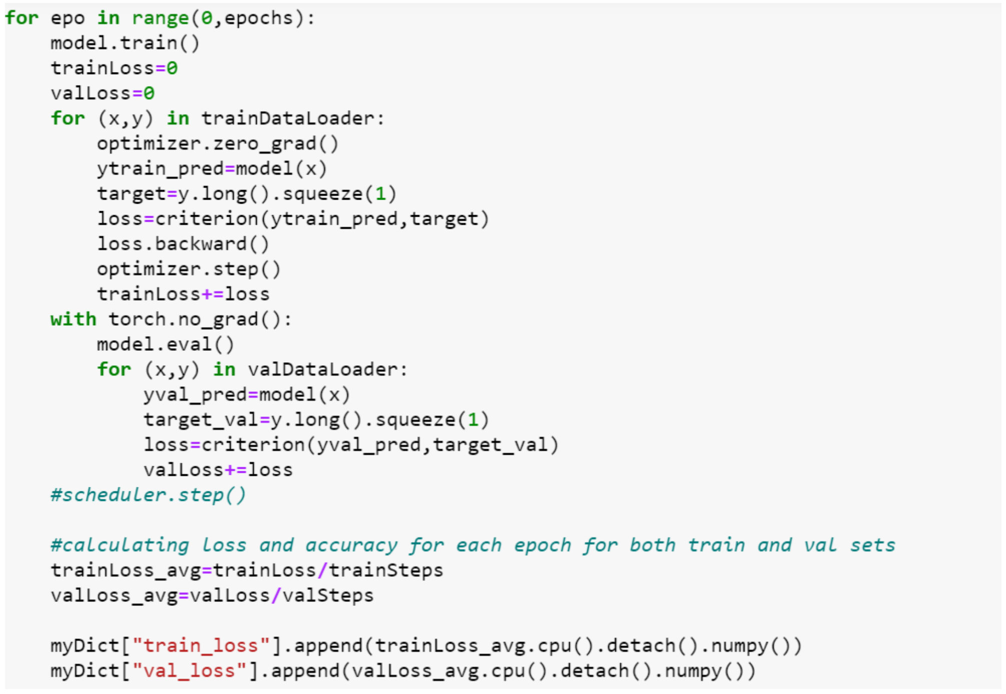
We split the dataset into train/val/test with 70%/15%/15% .We have total 200 images.

e) Experiment function



```
def experiment(train_data,val_data,epochs,model_version,batchsize,learning_rate):

    # initializing the train, validation, and test data loaders with given batchsize
    trainDataLoader = DataLoader(train_data, shuffle=False,batch_size=batchsize)
    valDataLoader = DataLoader(val_data, batch_size=batchsize)

    # steps per epoch for train and validation set
    trainSteps = len(trainDataLoader.dataset) // batchsize
    valSteps = len(valDataLoader.dataset) // batchsize

    #initialize the model
    print("//...Initializing {} model...//".format(model_version))
    #model= FCN32/16()
    model=model_version()

    #defining optimizer and cross-entropy loss function
    optimizer=Adam(model.parameters(),lr=learning_rate)
    criterion=nn.CrossEntropyLoss()

    #Learning rate will reduce by factor of 25% after every step size of 15epochs
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=15, gamma=0.25)

    #empty dict to store losses and accuracy score
    myDict={"train_loss":[],"val_loss":[]}
```

Here, we run the dataloader for train and validation dataset. We initialize the model. We define the Adam optimizer and criterion function as Cross Entropy loss. We also initialize the dictionary to include the train and val loss.



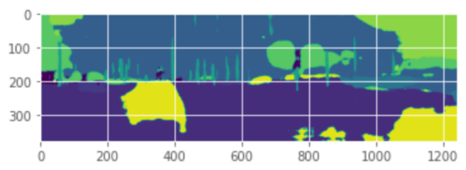
```
for epo in range(0,epochs):
    model.train()
    trainLoss=0
    valLoss=0
    for (x,y) in trainDataLoader:
        optimizer.zero_grad()
        ytrain_pred=model(x)
        target=y.long().squeeze(1)
        loss=criterion(ytrain_pred,target)
        loss.backward()
        optimizer.step()
        trainLoss+=loss
    with torch.no_grad():
        model.eval()
        for (x,y) in valDataLoader:
            yval_pred=model(x)
            target_val=y.long().squeeze(1)
            loss=criterion(yval_pred,target_val)
            valLoss+=loss
    #scheduler.step()

    #calculating loss and accuracy for each epoch for both train and val sets
    trainLoss_avg=trainLoss/trainSteps
    valLoss_avg=valLoss/valSteps

    myDict["train_loss"].append(trainLoss_avg.cpu().detach().numpy())
    myDict["val_loss"].append(valLoss_avg.cpu().detach().numpy())
```

We Start the training of our network. We load the train and validation loss after each epoch. We do Backward propagation for loss. We make sure that for validation, model is kept in evaluation mode and set torch,no_grad.

f) Prediction

```
y_pred_16=model_16.forward(x1)
#print(y_pred_16.shape)
#print(y1.shape)
y_pred_16_lab=np.argmax(((y_pred_16.permute(0,2,3,1)).detach().numpy()),axis=3)[0,:,:]
#y_pred_32_soft_v2=np.argmax((y_pred_32.detach().numpy()),axis=1)[0,:,:]

#print(y_pred_16_lab.shape)
#print(y_pred_16_lab)
#print(y_pred_16)
#print(y1)
#plt.imshow(y1.squeeze())
plt.imshow(y_pred_16_lab)
```

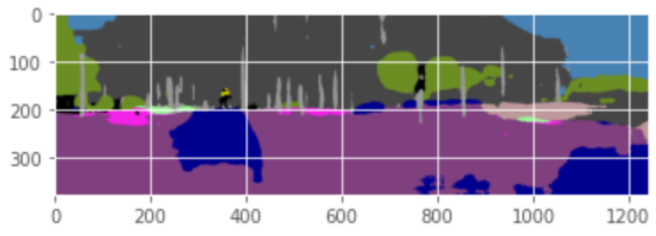
```
<matplotlib.image.AxesImage at 0x2002e0c0130>
```



We forward pass an input through our trained network and display the output. We take argmax on axis contain all classes which gives best label for the respective pixel. We convert this to numpy and display the output through matplotlib.plt.imshow function.

g) Color Mapping

```
pred = (torch.argmax(y_pred_16, dim=1)).squeeze().numpy()
color_img=np.zeros((pred.shape[0],pred.shape[1],3))

for i in range(pred.shape[0]):
    for j in range(pred.shape[1]):
        rgb_val=list(color_dict[pred[i][j]])
        for k in range(3):
            color_img[i][j][k]=rgb_val[k]
color_img=color_img.astype(np.uint8)
plt.imshow(color_img)
plt.show()
```



We color map each pixel with the help of defined dictionary earlier. Each pixel represents one class and accordingly color mapping is done. For this I created (w,h,3) array containing zeros. And then iterate over each pixel to color map. At end we change the datatype of this array to uint8 in order to display correctly through plt.imshow

h) One-Hot encoding of ytrue and ypred

```
def oneHot(Ypred,Ytrue,label_pos):
    ypred_hot=np.zeros((Ypred.shape[0],Ypred.shape[1]))
    ytrue_hot=np.zeros((Ytrue.shape[0],Ytrue.shape[1]))
    for i in range(Ypred.shape[0]):
        for j in range(Ypred.shape[1]):
            if Ypred[i][j]==label_pos:
                ypred_hot[i][j]=1
            else:
                ypred_hot[i][j]=0
            if Ytrue[i][j]==label_pos:
                ytrue_hot[i][j]=1
            else:
                ytrue_hot[i][j]=0
    return ypred_hot,ytrue_hot
```

I create 2 arrays of shape matching to ytrue and ypred. Then for the provided class, I compare each pixel to the given class and if True I'll change the pixel value of the ypred_hot and ytrue_hot to 1 respectively.

i) Evaluation Metric (Intersection over Union)

```
testDataLoader = DataLoader(test_dataset, shuffle=False,batch_size=1)
smooth=0.001
iou=[]
for cls in range(34):
    intersection_class=0
    union_class=0
    with torch.no_grad():
        model_16.eval()
        for (x,y) in testDataLoader:
            ypred=(torch.argmax(model_16.forward(x), dim=1)).squeeze().numpy()
            ytrue=y.squeeze().detach().numpy()
            ypred_hot,ytrue_hot=oneHot(ypred,ytrue,cls)
            intersection=np.sum(np.abs(ypred_hot*ytrue_hot),axis=(0,1))
            mask_sum = np.sum(np.abs(ytrue_hot), axis=(0,1)) + np.sum(np.abs(ypred_hot), axis=(0,1))
            union = mask_sum  - intersection
            intersection_class=intersection_class+intersection
            union_class=union_class+union
    score=(intersection_class+smooth)/(union_class+smooth)
    iou.append(score)
```

Here, I iterate over each class and keep model in evaluation mode. Then I'll convert the ytrue and ypred into on-hot encoded labels. Then Intersection is equal to the common 1's in both the arrays. That is computed by intersection=np.sum(np.abs(ypred_hot*ytrue_hot),axis=(0,1))

Mask_sum is the total '1' in both array (including common 1's). Hence union is computed by union = mask_sum - intersection (to remove the duplicate of common 1's).

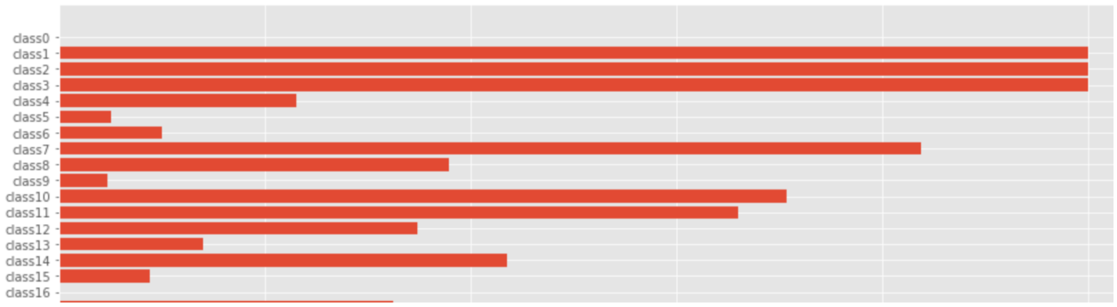
This is calculated for each test image and finally we have one value of IoU for each class given by score=(intersection_class+smooth)/(union_class+smooth).

Here, I provide smoothing=0.005 in order to avoid nan values when a particular class in not present in any test image.

Code reference: https://ilmonteux.github.io/2019/05/10/segmentation-metrics.html

j) Plotting bar chart for IoU values

```
x_scale=[]
for j in range(34):
    x_scale.append("class"+str(j))

fig, ax = plt.subplots(figsize =(16, 9))
ax.barh(x_scale, iou, align='center')
ax.invert_yaxis()
plt.show()
```



Here, I plot a bar chart for IoU values for each class. Note: here please consider IoU=1 as 'N/A' these are the classes that were not present in any test images. They are '1' due to smoothness parameter. Also, few of the classes have IoU very low and hence cannot be seen in the bar chart. I have presented all those values in the later part of this report.

2) Evolution of loss function with multiple steps.

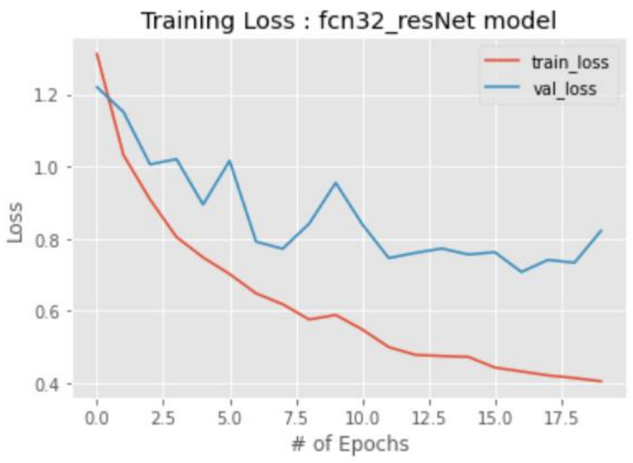


Figure 1: Train/Validation Loss_FCN32

Epoch	Training Loss	Validation Loss
1	1.3115	1.2198
2	1.0315	1.509
3	0.9084	1.0056
4	0.8042	1.0210
5	0.7479	0.8942
6	0.7018	1.0156
7	0.6481	0.7914
8	0.6181	0.7717
9	0.5757	0.8414
10	0.5883	0.9546
11	0.5483	0.8399
12	0.4992	0.7460
13	0.4779	0.7603
14	0.4746	0.7725
15	0.4720	0.7558
16	0.4426	0.7623
17	0.4319	0.7075
18	0.4209	0.7409
19	0.4137	0.7331
20	0.4045	0.8215

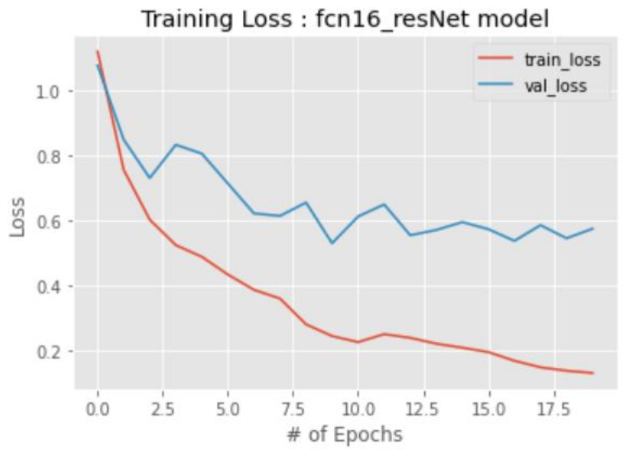
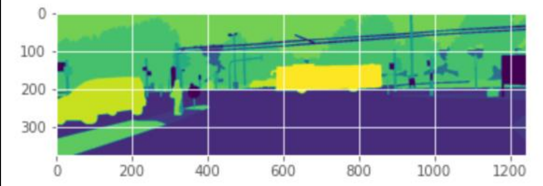
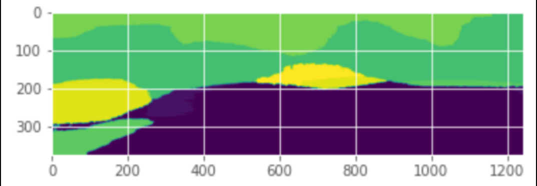
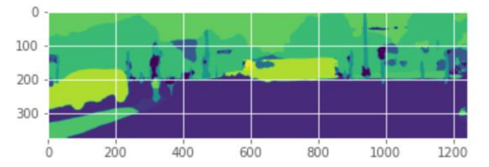
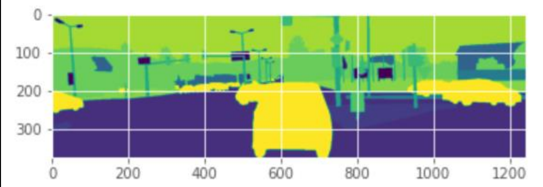
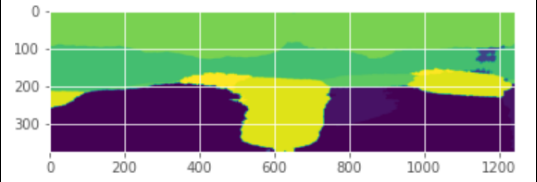
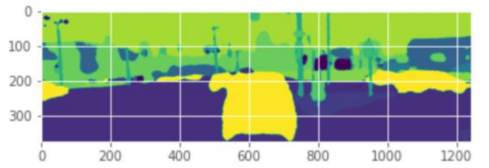
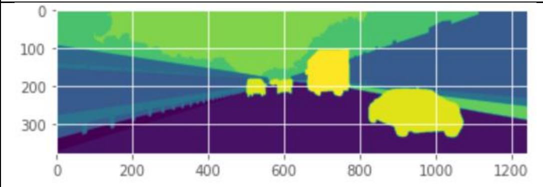
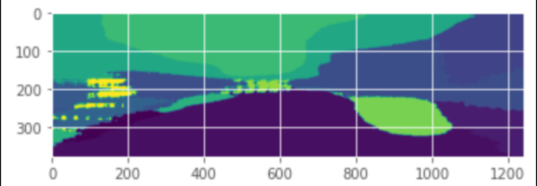
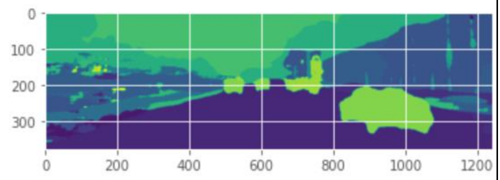
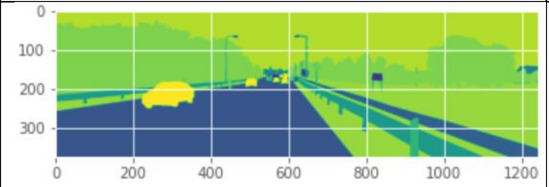
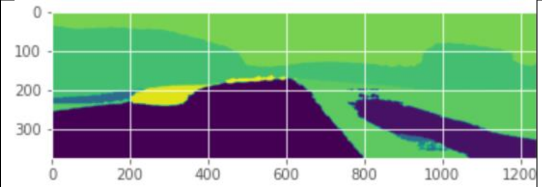
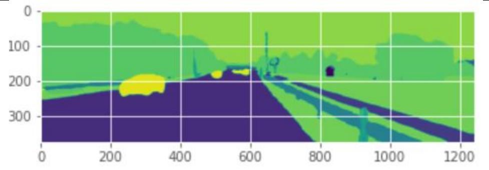
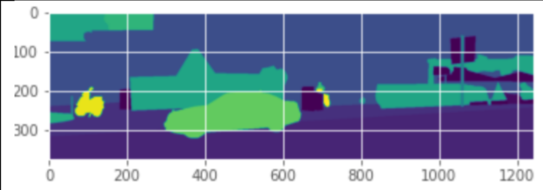
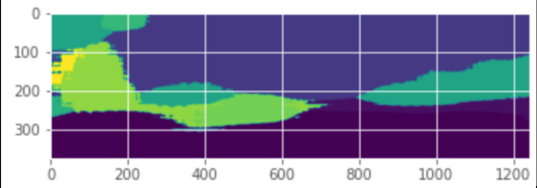
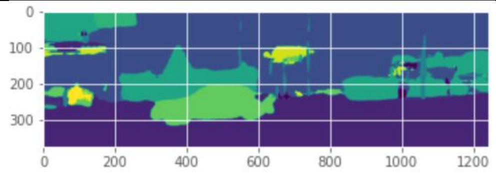


Figure 2: Train\Validation Loss: FCN16

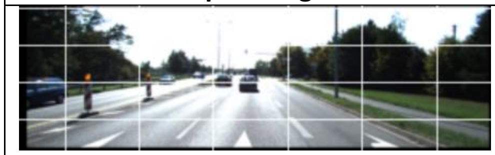
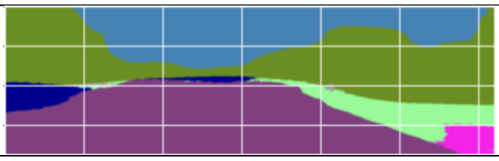
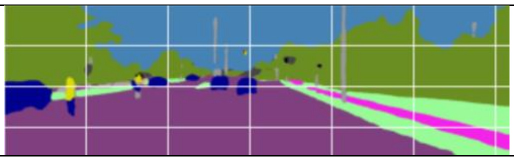
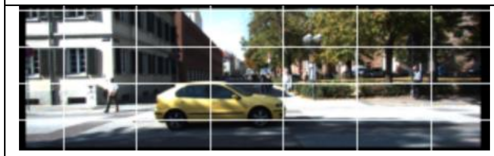
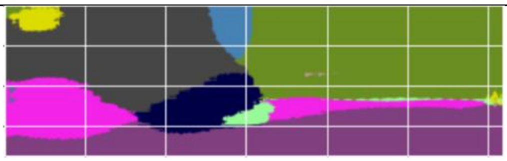
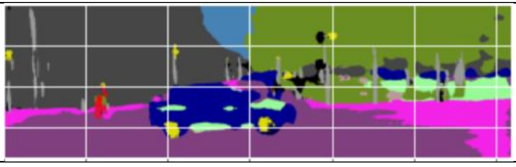
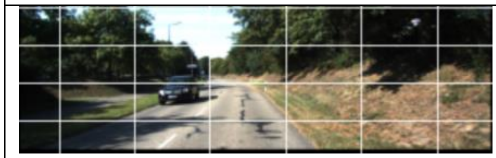
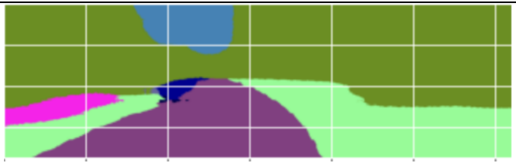
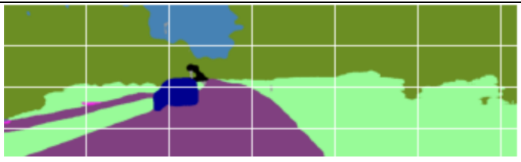
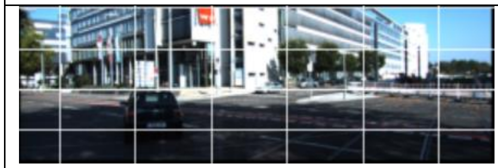
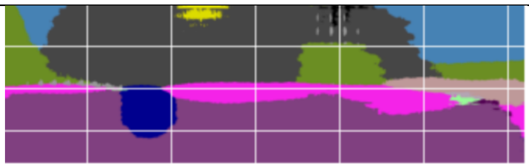
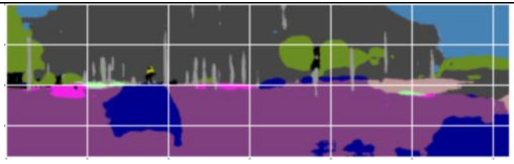
Epoch	Training Loss	Validation Loss
1	1.1172	1.0747
2	0.7545	0.8471
3	0.6016	0.7293
4	0.5235	0.8314
5	0.4876	0.8042
6	0.4331	0.7126
7	0.3862	0.6207
8	0.3594	0.6130
9	0.2802	0.6541
10	0.2443	0.5291
11	0.2257	0.6113
12	0.2502	0.6484
13	0.2388	0.5536
14	0.2207	0.5699
15	0.2085	0.5940
16	0.1950	0.5723
17	0.1684	0.5364
18	0.1482	0.5846
19	0.1377	0.5444
20	0.1308	0.5736

3) A summary and discussion of the results, including the effects of parameter choices. Compare the 2 versions of modified FCN (32s and 16s). Include the visualization of results; show some examples of successful and some failure examples.

Ouput prediction of both Networks compared with GT_mask

GT mask	**FCN32 Output**	**FCN16 Output**
		
		
		
		
		

Color mapping of the output prediction.

Input image	FCN32	FCN16
		
		
		
		

FCN16: IoU SCORES

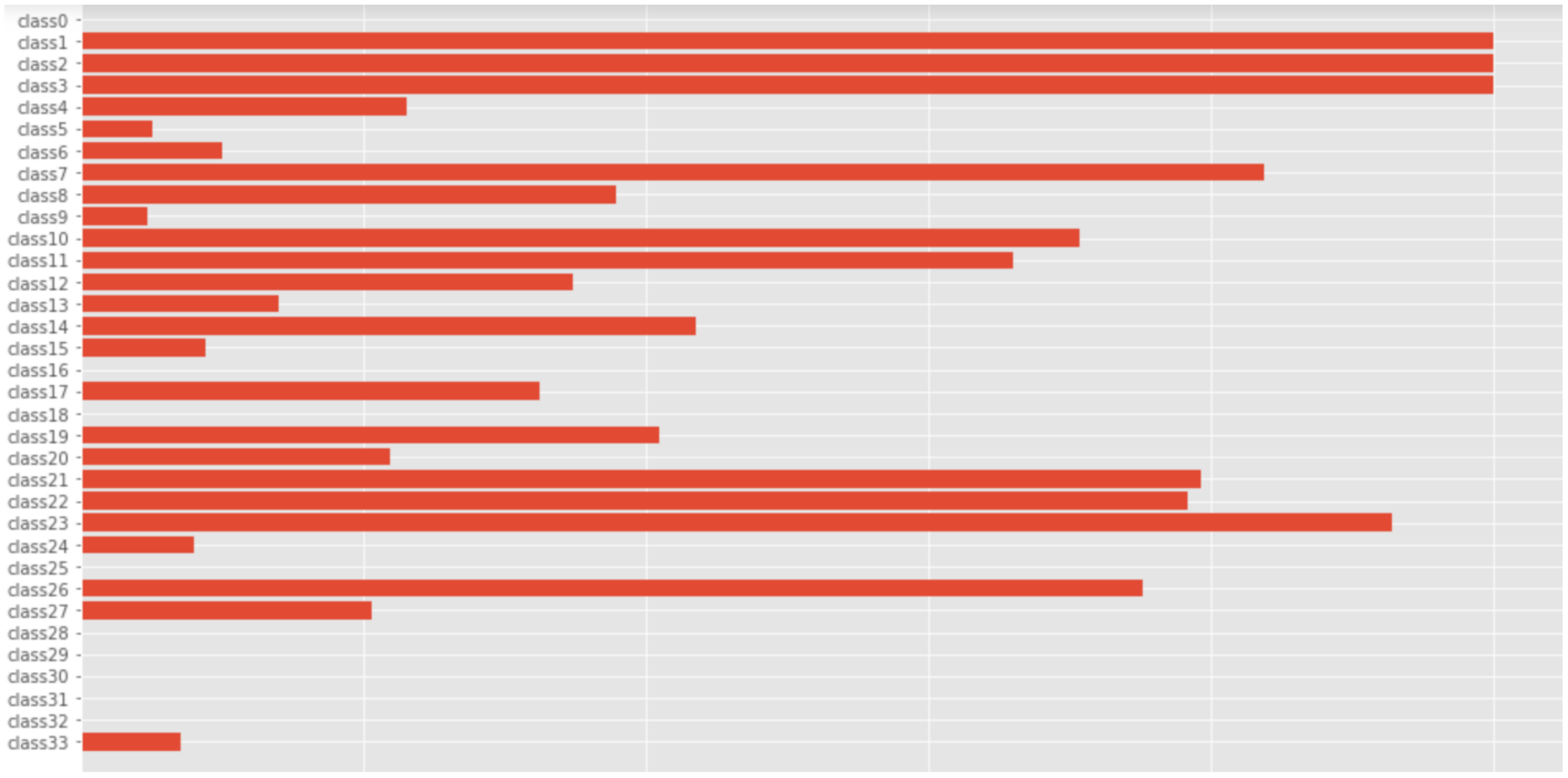


Figure 3: IoU_FCN16

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
IoU score	2.2e-06	N/A	N/A	N/A	0.23	0.05	0.099	0.83	0.37	0.04	0.71	0.66	0.35	0.14	0.44	0.09	3.3e-09	0.33	9.6e-06	0.41	0.22

Class	**21**	**22**	**23**	**25**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
IoU score	0.79	0.78	0.93	0.08	8.7e-07	0.75	0.21	2.5e-08	2.6e-07	6.5e-07	1.1e-07	2.7e-07	0.07

Mean IoU score = 0.2769

FCN32: IoU SCORES

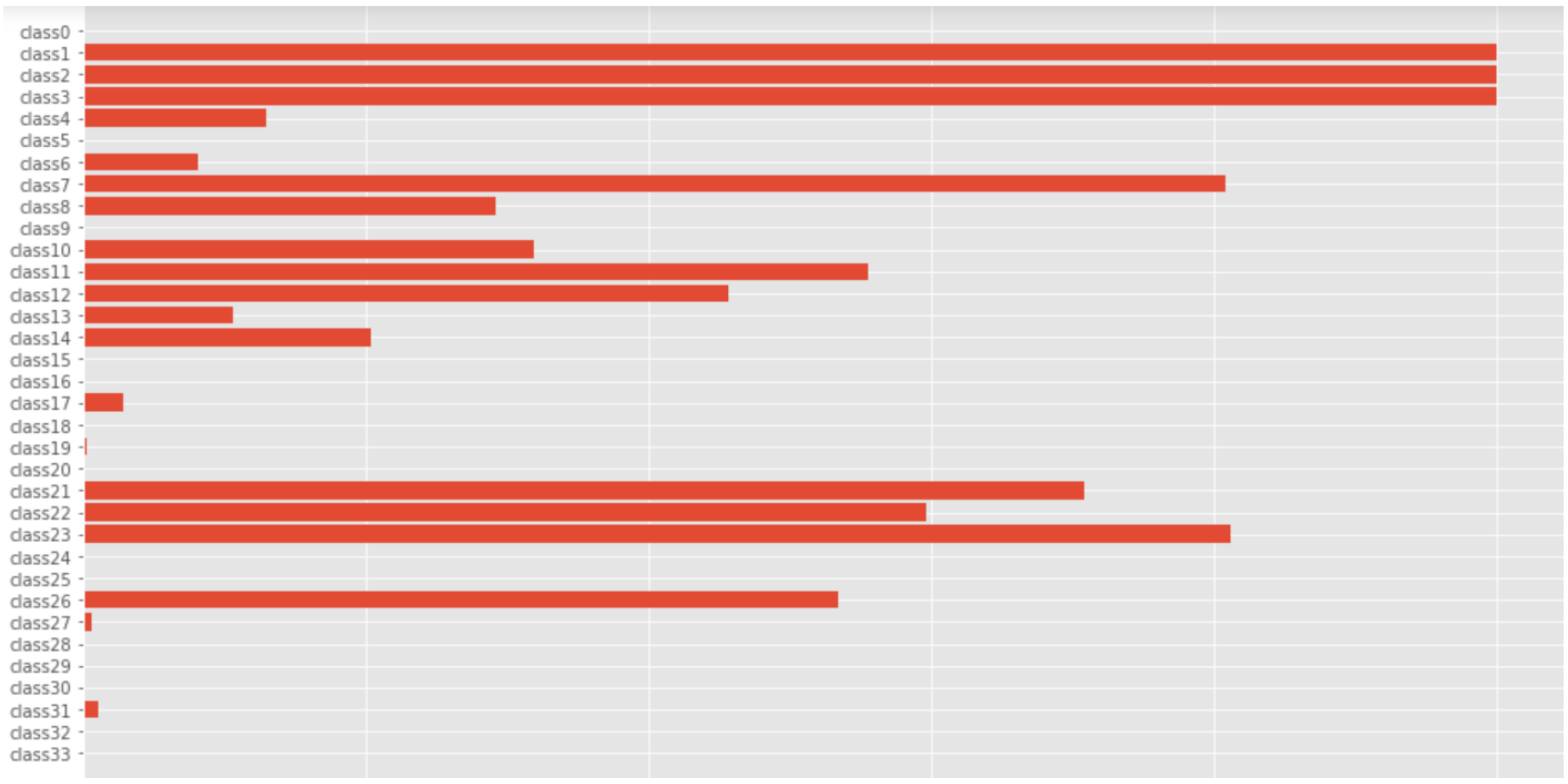


Figure 4: IoU_FCN32

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
IoU score	2.4e-06	N/A	N/A	N/A	0.13	1.5e-07	0.08	0.81	0.29	2.3e-08	0.31	0.55	0.46	0.11	0.20	4.6e-07	3.3e-09	0.02	9.6e-06	0.002	2.4e-08

Class	**21**	**22**	**23**	**25**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
IoU score	0.71	0.59	0.81	7.9e-08	1.6e-06	0.53	0.005	2.5e-08	4.2e-07	7.0e-07	0.01	2.7e-07	4.0e-07

Mean IoU score= 0.1821

Observations:

1) **FCN16 performs far better than FCN32 when the visualizations are compared.**
2) **The networks were trained on CPU as I kept batchsize=1. Due to which using GPU was ineffective due to large number of data transfer from CPU to GPU.**
3) **FCN32 could be trained with more number of epochs and might get better results.**
4) **FCN16 segments more coarse details of the image due to upscoring from previous convolutional block.**
5) **Using trained ResNet network, helped in training as we got good weights values for initializing.**
6) **IoU score is best for class 23: sky for both networks. 'Road' is second best classified and segmented for both networks.**
7) **Few classes were finely classified with FCN16 which was not classified more often in case of FCN32. Eg. Class 27 (truck)**
8) **Mean Intersection over Union value is more for FCN16 compared to FCN32.**
9) **Cross Entropy loss function calculates loss over multi dimension, in our case (34, number of classes) and hence was good consideration of loss function.**
10) **Since, I used only 20 epochs, I did not lower the learning rate after few epochs. I continued with initial learning rate of 0.001.**