Untitled

November 16, 2021

[510]:
```
import numpy as np
import torch
import torch.nn as nn
from torchvision import models
import os,glob
from torch.utils.data import Dataset, DataLoader,random_split
from torchvision import transforms
from PIL import Image,ImageOps
import matplotlib.pyplot as plt
import torch.nn.functional as F
from torch.optim import Adam
import time
from datetime import datetime
```

[515]:
```
def get_upsampling_weight(in_channels, out_channels, kernel_size):
    """Make a 2D bilinear kernel suitable for upsampling"""
    factor = (kernel_size + 1) // 2
    if kernel_size % 2 == 1:
        center = factor - 1
    else:
        center = factor - 0.5
    og = np.ogrid[:kernel_size, :kernel_size]
    filt = (1 - abs(og[0] - center) / factor) * \
           (1 - abs(og[1] - center) / factor)
    weight = np.zeros((in_channels, out_channels, kernel_size, kernel_size),
                      dtype=np.float64)
    weight[range(in_channels), range(out_channels), :, :] = filt
    return torch.from_numpy(weight).float()
```

[514]:
```
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()

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

        self._initialize_weights()

    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)

    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()

        return output
```

[516]:
```
class fcn16_resNet(nn.Module):
    def __init__(self):
        super().__init__()

        #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()

    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)

    def forward(self,x):
        h=x

        #passing the input till the pool4 layers of backbone network, ResNet18
        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
```

[621]:
```
color_dict={0:( 0,  0,  0),
            1:(  0,  0,  0),
            2:(  0,  0,  0),
            3:(  0,  0,  0),
            4:(  0,  0,  0),
            5:(111,74,0),
            6:(81,0,81),
            7:(128,64,128),
            8:(244,35,232),
            9:(250,170,160),
            10:(230,150,140),
            11:(70,70,70),
            12:(102,102,156),
            13:(190,153,153),
            14:(180,165,180),
            15:(150,100,100),
            16:(150,120,90),
```

```
            17:(153,153,153),
            18:(153,153,153),
            19:(250,170,30),
            20:(220,220,0),
            21:(107,142,35),
            22:(152,251,152),
            23:(70,130,180),
            24:(220,20,60),
            25:(255,0,0),
            26:(0,0,142),
            27:(0,0,70),
            28:(0,60,100),
            29:(0,0,90),
            30:(0,0,110),
            31:(0,80,100),
            32:(0,0,230),
            33:(119,11,32)}

color_dict[33]
```

[621]: (119, 11, 32)

[517]:
```
res=models.resnet18(pretrained=True)
```

[361]:
```
feat=nn.Sequential(*(list(res.children())[0:8]))
#feat
```

[475]:
```
model_fcn32=fcn32_resNet()
#model_fcn32
```

[456]:
```
model_fcn16=fcn16_resNet()
#model_fcn16
```

[518]:
```
a="C:\\Users\\hp631\\USC_Jupyter\\Fall'21\\Advanced computer␣
 ↪vision\\HW5\\data\\datafile\\training\\image_2"

#a=os.listdir(r"C:\Users\hp631\USC_Jupyter\Fall'21\Advanced computer␣
 ↪vision\HW5\data\datafile\training\image_2")
#b=os.listdir(r"C:\Users\hp631\USC_Jupyter\Fall'21\Advanced computer␣
 ↪vision\HW5\data\datafile\training\semantic")
b="C:\\Users\\hp631\\USC_Jupyter\\Fall'21\\Advanced computer␣
 ↪vision\\HW5\\data\\datafile\\training\\semantic"
print(b)
```

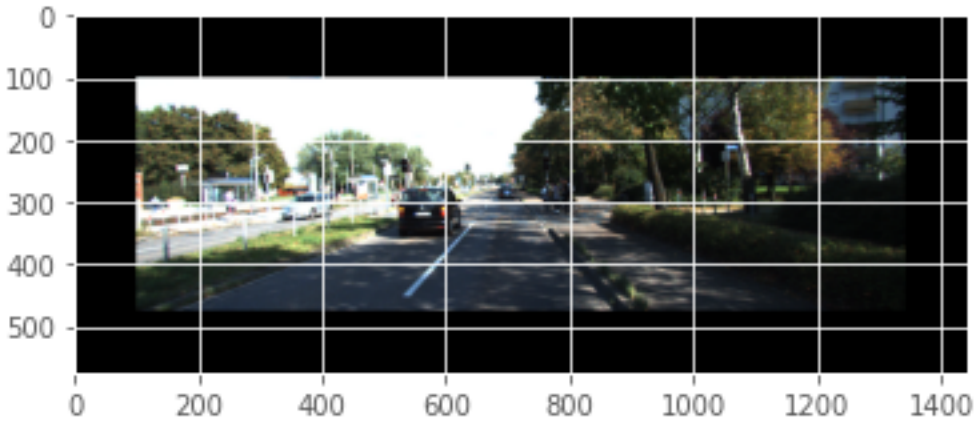
```
C:\Users\hp631\USC_Jupyter\Fall'21\Advanced computer
vision\HW5\data\datafile\training\semantic
```

[520]:
```
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]

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

[521]:
```
mydataset=KITTI_custom(a,b,transforms.ToTensor(),transforms.Compose([transforms.
 ↪ToTensor(), lambda x: x*255]))
dataloader=DataLoader(mydataset,batch_size=1,shuffle=True)
x1,y1=next(iter(dataloader))
plt.imshow((x1.squeeze()).permute(1,2,0))
#plt.imshow(y1.squeeze())
print(x1.shape)
print(x1)
```

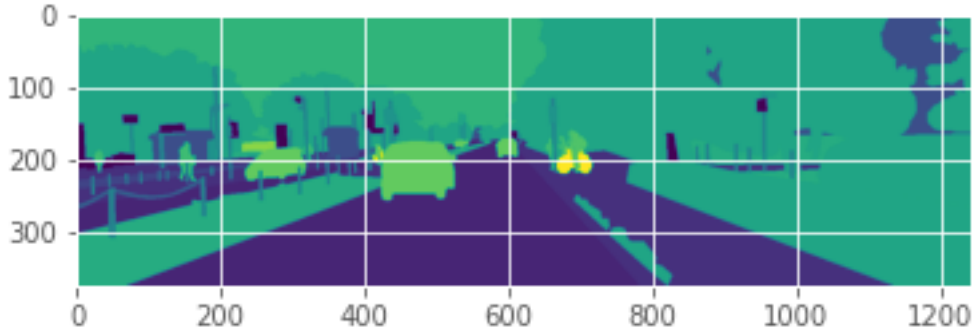
```
torch.Size([1, 3, 575, 1442])
tensor([[[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
```

```
 [0., 0., 0.,  …, 0., 0., 0.]],

[[0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.],
 …,
 [0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.]],

[[0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.],
 …,
 [0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.],
 [0., 0., 0.,  …, 0., 0., 0.]]])
```



[522]:
```
plt.imshow(y1.squeeze())
```

[522]:
```
<matplotlib.image.AxesImage at 0x2001680f670>
```



[523]:
```
classes=np.unique(y1.detach().numpy())
classes
```

[523]:
```
array([ 4.,  7.,  8., 10., 11., 13., 17., 19., 20., 21., 22., 23., 24.,
       25., 26., 28., 32., 33.], dtype=float32)
```

[525]:
```
#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
```

[533]:
```
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":[]}

    #initialize training the model
    print("//....Training initiated...//")
    start=time.time()

    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())

        print("//... epoch: {}/{}".format(epo + 1, epochs))
        print("Training loss: {:.4f}".format(trainLoss_avg))
        print("Validation loss: {:.4f} \n".format(valLoss_avg))

    end=time.time()
    print("//...training the network took {:.2f} sec...//".format(end-start))

    #plotting loss and acc
    plt.style.use("ggplot")
    plt.figure()
```

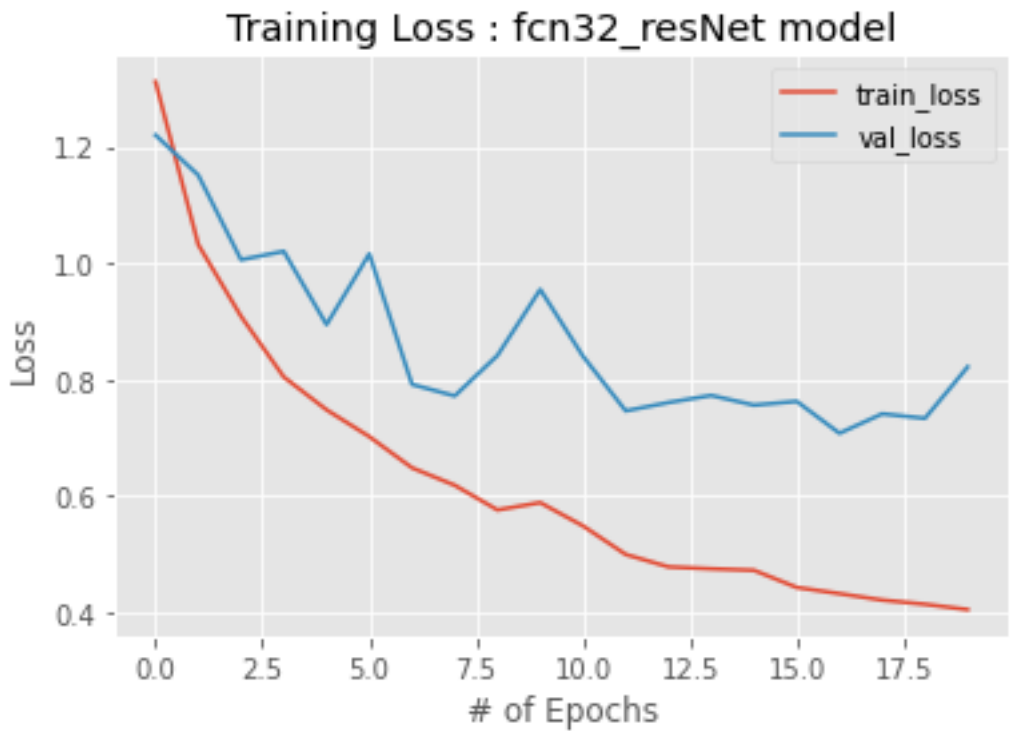
```
    plt.plot(myDict["train_loss"], label="train_loss")
    plt.plot(myDict["val_loss"], label="val_loss")
    plt.title("Training Loss : {} model".format(type(model).__name__))
    plt.xlabel("# of Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

    return model
```

[534]:
```
model_32=experiment(train_dataset,val_dataset,20,fcn32_resNet,1,0.001)
PATH="E:\\usc\\Fall'21\\CSCI677 Adv Computer Vision\\HW5\\models\\model32\\{}.
 ↪pth".format(datetime.now().strftime("%d_%m_%H_%M"))
torch.save(model_32.state_dict(), PATH)
```

```
//…Initializing <class '__main__.fcn32_resNet'> model…//
//…Training initiated…//
//… epoch: 1/20
Training loss: 1.3115
Validation loss: 1.2198

//… epoch: 2/20
Training loss: 1.0315
Validation loss: 1.1509

//… epoch: 3/20
Training loss: 0.9084
Validation loss: 1.0056

//… epoch: 4/20
Training loss: 0.8042
Validation loss: 1.0201

//… epoch: 5/20
Training loss: 0.7479
Validation loss: 0.8942

//… epoch: 6/20
Training loss: 0.7018
Validation loss: 1.0156

//… epoch: 7/20
Training loss: 0.6481
Validation loss: 0.7914

//… epoch: 8/20
Training loss: 0.6181
Validation loss: 0.7717
```

```
//… epoch: 9/20
Training loss: 0.5757
Validation loss: 0.8414

//… epoch: 10/20
Training loss: 0.5883
Validation loss: 0.9546

//… epoch: 11/20
Training loss: 0.5483
Validation loss: 0.8399

//… epoch: 12/20
Training loss: 0.4992
Validation loss: 0.7460

//… epoch: 13/20
Training loss: 0.4779
Validation loss: 0.7603

//… epoch: 14/20
Training loss: 0.4746
Validation loss: 0.7725

//… epoch: 15/20
Training loss: 0.4720
Validation loss: 0.7558

//… epoch: 16/20
Training loss: 0.4426
Validation loss: 0.7623

//… epoch: 17/20
Training loss: 0.4319
Validation loss: 0.7075

//… epoch: 18/20
Training loss: 0.4209
Validation loss: 0.7409

//… epoch: 19/20
Training loss: 0.4137
Validation loss: 0.7331

//… epoch: 20/20
Training loss: 0.4045
Validation loss: 0.8215
```

```
//…training the network took 8822.77 sec…//
```

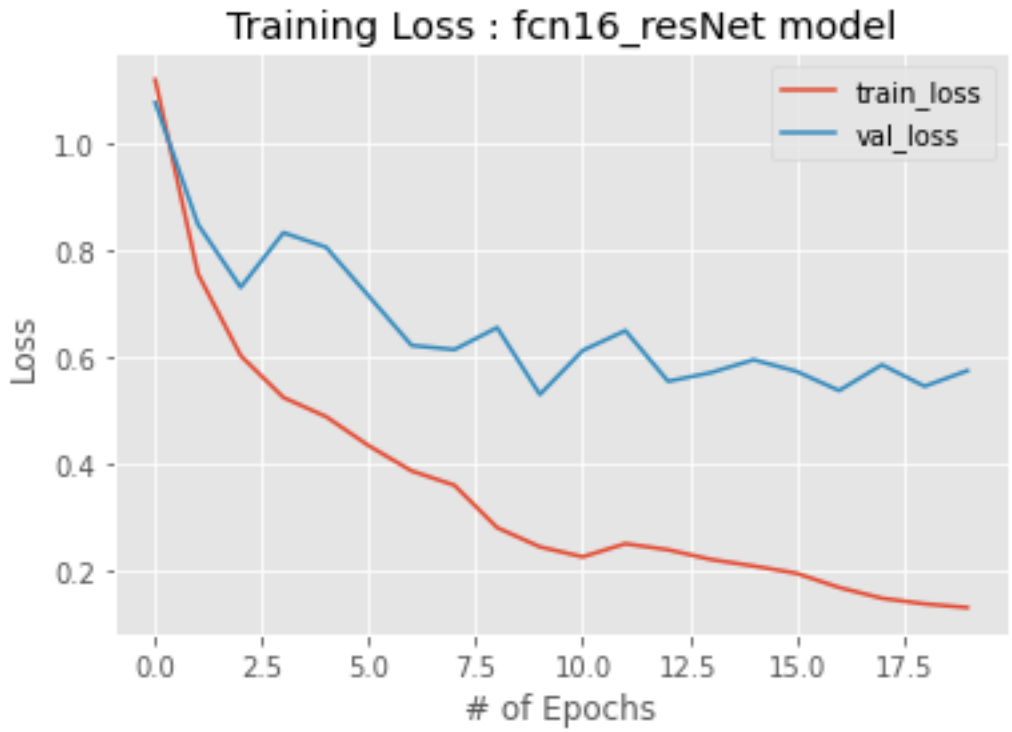


[535]:
```
model_16=experiment(train_dataset,val_dataset,20,fcn16_resNet,1,0.001)
PATH="E:\\usc\\Fall'21\\CSCI677 Adv Computer Vision\\HW5\\models\\model16\\{}.
 ↪pth".format(datetime.now().strftime("%d_%m_%H_%M"))
torch.save(model_16.state_dict(), PATH)
```

```
//…Initializing <class '__main__.fcn16_resNet'> model…//
//…Training initiated…//
//… epoch: 1/20
Training loss: 1.1172
Validation loss: 1.0747

//… epoch: 2/20
Training loss: 0.7545
Validation loss: 0.8471

//… epoch: 3/20
Training loss: 0.6016
Validation loss: 0.7293

//… epoch: 4/20
Training loss: 0.5235
```

```
Validation loss: 0.8314

//… epoch: 5/20
Training loss: 0.4876
Validation loss: 0.8042

//… epoch: 6/20
Training loss: 0.4331
Validation loss: 0.7126

//… epoch: 7/20
Training loss: 0.3862
Validation loss: 0.6207

//… epoch: 8/20
Training loss: 0.3594
Validation loss: 0.6130

//… epoch: 9/20
Training loss: 0.2802
Validation loss: 0.6541

//… epoch: 10/20
Training loss: 0.2443
Validation loss: 0.5291

//… epoch: 11/20
Training loss: 0.2257
Validation loss: 0.6113

//… epoch: 12/20
Training loss: 0.2502
Validation loss: 0.6484

//… epoch: 13/20
Training loss: 0.2388
Validation loss: 0.5536

//… epoch: 14/20
Training loss: 0.2207
Validation loss: 0.5699

//… epoch: 15/20
Training loss: 0.2085
Validation loss: 0.5940

//… epoch: 16/20
Training loss: 0.1950
```

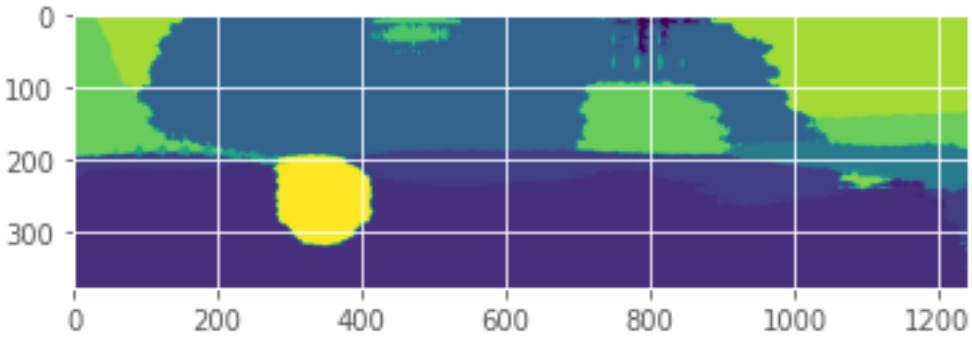
```
Validation loss: 0.5723

//… epoch: 17/20
Training loss: 0.1684
Validation loss: 0.5364

//… epoch: 18/20
Training loss: 0.1482
Validation loss: 0.5846

//… epoch: 19/20
Training loss: 0.1377
Validation loss: 0.5444

//… epoch: 20/20
Training loss: 0.1308
Validation loss: 0.5736

//…training the network took 9862.73 sec…//
```



[641]:
```
test_dataloader=DataLoader(test_dataset,batch_size=1,shuffle=True)
x1,y1=next(iter(test_dataloader))
#plt.imshow((x1.squeeze()).permute(1,2,0))
```

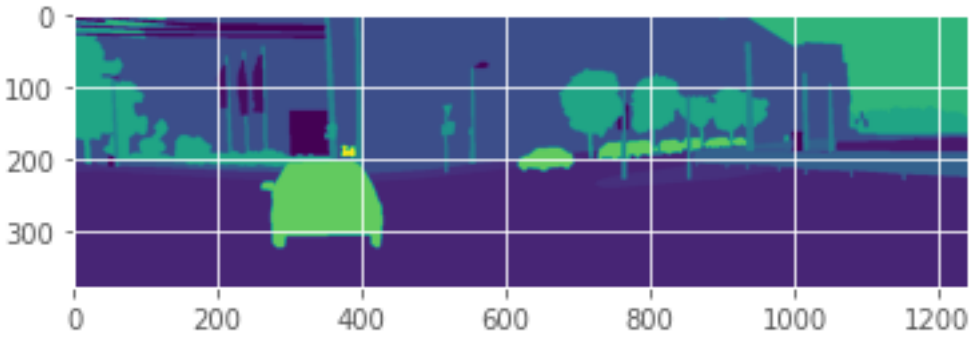
```
#plt.imshow(y1.squeeze())
#print(x1.shape)
#print(x1)

y_pred_32=model_32.forward(x1)
#print(y_pred_32.shape)
#print(y1.shape)
y_pred_32_lab=np.argmax(((y_pred_32.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_32_lab.shape)
#print(y_pred_32_lab)
#print(y_pred_32)
#print(y1)
#plt.imshow(y1.squeeze())
plt.imshow(y_pred_32_lab)
```

[641]: <matplotlib.image.AxesImage at 0x2001b3e28e0>



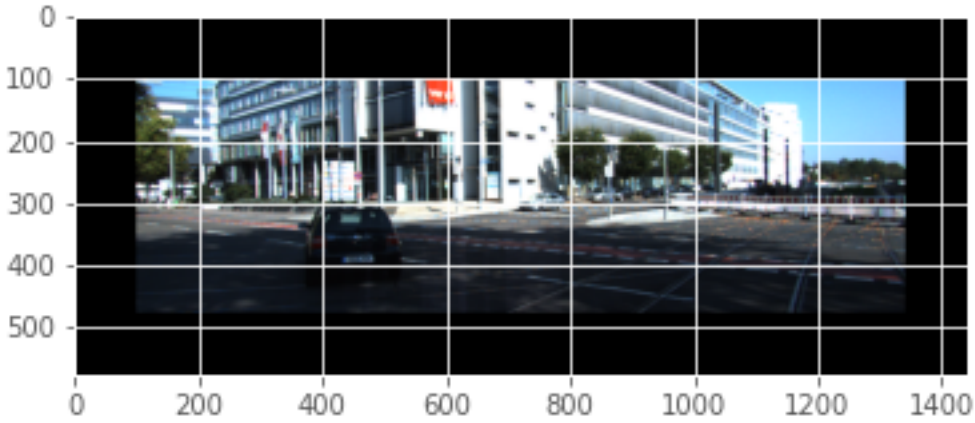
[642]:
```
plt.imshow(y1.squeeze())
```

[642]: <matplotlib.image.AxesImage at 0x2001b449640>



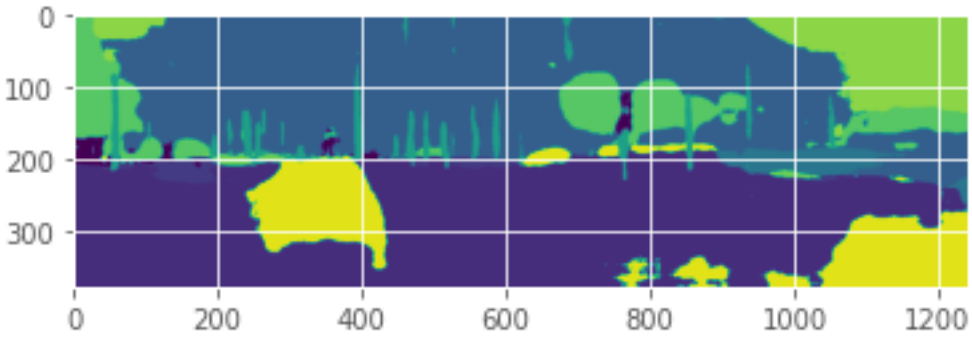
[643]:
```
plt.imshow((x1.squeeze()).permute(1,2,0))
```

[643]: <matplotlib.image.AxesImage at 0x2002de4a880>

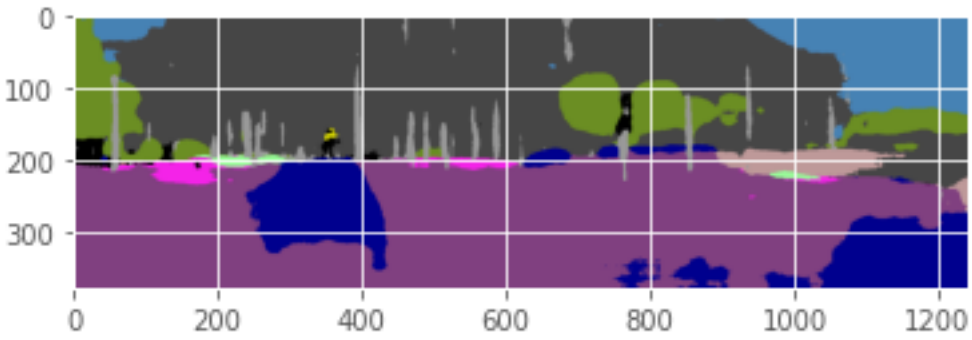


[644]:
```
y_pred_16=model_16.forward(x1)
#print(y_pred_16.shape)
#print(y1.shape)
y_pred_16_lab=np.argmax(((y_pred_16.permute(0,2,3,1)).detach().
 ↪numpy()),axis=3)[0,:,:]
#y_pred_32_soft_v2=np.argmax((y_pred_32.detach().numpy()),axis=1)[0,:,:]

#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)
```

[644]: <matplotlib.image.AxesImage at 0x2002e0c0130>

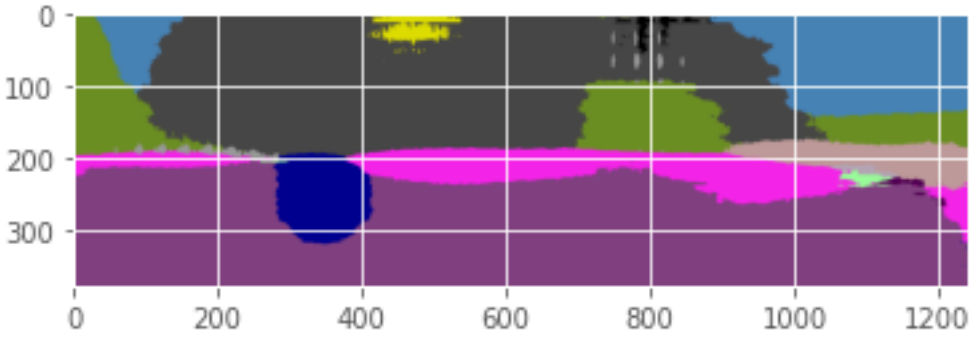


[645]:
```
pred = (torch.argmax(y_pred_16, dim=1)).squeeze().numpy()
color_img=np.zeros((pred.shape[0],pred.shape[1],3))

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()
```



[646]:
```
pred_32 = (torch.argmax(y_pred_32, dim=1)).squeeze().numpy()
color_img_32=np.zeros((pred_32.shape[0],pred_32.shape[1],3))

for i in range(pred_32.shape[0]):
    for j in range(pred_32.shape[1]):
        rgb_val=list(color_dict[pred_32[i][j]])
        for k in range(3):
            color_img_32[i][j][k]=rgb_val[k]
```

```
color_img_32=color_img_32.astype(np.uint8)
plt.imshow(color_img_32)
plt.show()
```



[659]:
```
#dummy code
ypred=(torch.argmax(model_16.forward(x1), dim=1)).squeeze().numpy()
ytrue=y1.squeeze().detach().numpy()
print(ypred.shape[0])
print(ytrue.shape)
```

```
376
(376, 1241)
```

[700]:
```
#dummy code
a1=np.array([[0,1,0],[1,0,1]])
a2=np.array([[0,1,0],[0,1,1]])
intersection=np.sum(np.abs(a1*a2),axis=(0,1))
mask_sum = np.sum(np.abs(a1), axis=(0,1)) + np.sum(np.abs(a2), axis=(0,1))
union = mask_sum  - intersection
iou_temp=intersection/union
iou_temp
```

[700]: 0.5

[663]:
```
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
```

[668]:
```
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)
```

[719]:
```
iou_scores=iou
for i,sco in enumerate(iou_scores):
    if sco==1:
        iou_scores[i]='N/A'
iou_scores
```

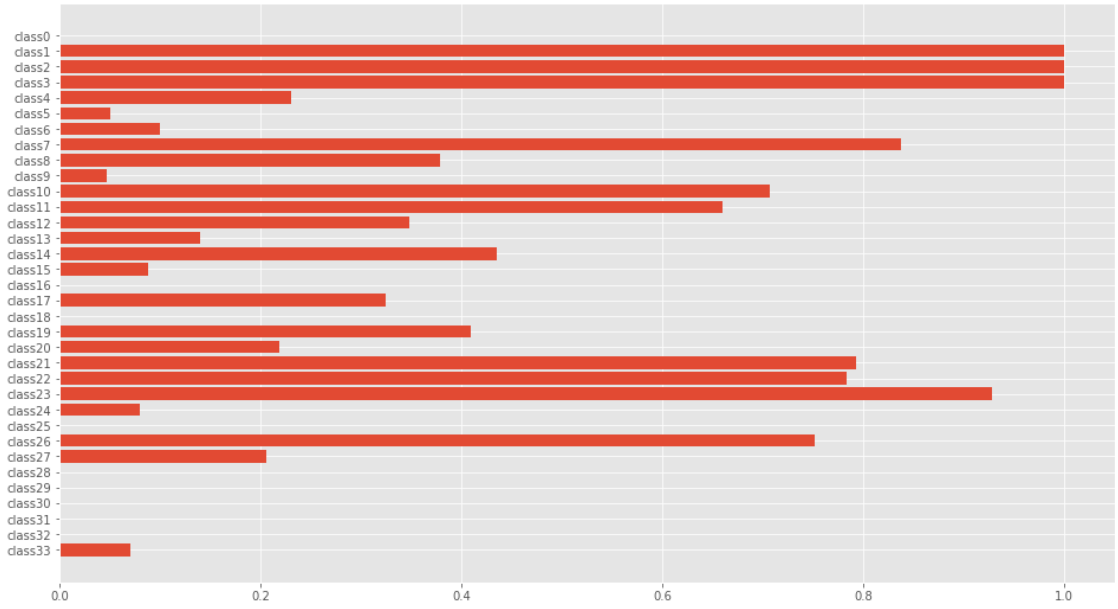
[719]:
```
[2.2222172839615914e-06,
 'N/A',
 'N/A',
 'N/A',
 0.231036164739202,
 0.050636755358297175,
 0.09985620065583961,
 0.8368964067061173,
 0.37932320877778863,
 0.047371263515181505,
 0.706572502004664,
 0.6594890555327194,
 0.34769565951593834,
 0.13958343161002015,
```

```
 0.4351507003795953,
 0.0878950441202621,
 3.3394668095753643e-09,
 0.32514361258903973,
 9.615292160652301e-06,
 0.4087523506846543,
 0.2187025981908527,
 0.7931750532250866,
 0.7833640451679014,
 0.9276868336407721,
 0.08001980541315529,
 8.77962354730154e-07,
 0.7512057936058436,
 0.20530787972981865,
 2.5562371534704206e-08,
 2.6130121209793257e-07,
 6.353236116114285e-07,
 1.1650936543057608e-07,
 2.7570987711335064e-07,
 0.07022727707584585]
```

[712]:
```
scores_16=[]
for sco in iou_scores:
    if sco!='N/A':
        scores_16.append(sco)
miou_16=np.mean(scores_16)
print("mIoU for FCN16 netowrk = {}".format(miou_16))
```

```
mIoU for FCN16 netowrk = 0.2769388927566549
```

[718]:
```
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()
```



[684]:
```
testDataLoader = DataLoader(test_dataset, shuffle=False,batch_size=1)
smooth=0.001
iou_32=[]
for cls in range(34):
    intersection_class=0
    union_class=0
    with torch.no_grad():
        model_32.eval()
        for (x,y) in testDataLoader:
            ypred=(torch.argmax(model_32.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_32.append(score)
```

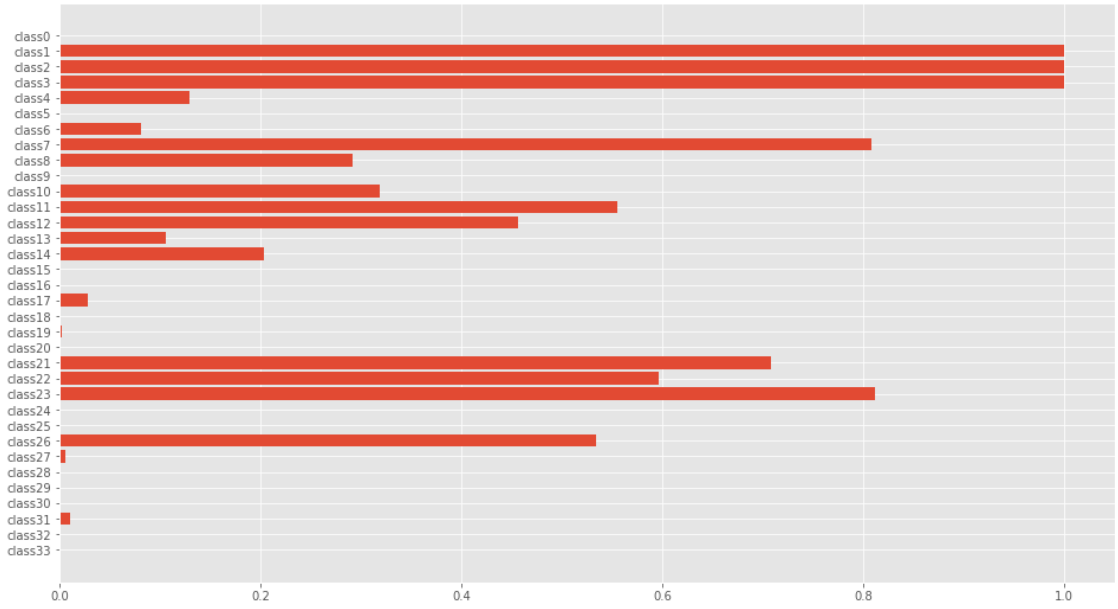
[706]:
```
iou_scores_32=iou_32
for i,sco in enumerate(iou_scores_32):
    if sco==1:
        iou_scores_32[i]='N/A'
iou_scores_32
```

[706]:
```
[2.4096327478728968e-06,
 'N/A',
 'N/A',
 'N/A',
 0.12957794552844612,
 1.5119441318499953e-07,
 0.08132142504381275,
 0.8081879788867206,
 0.2919028296615624,
 2.31481476123114e-08,
 0.31895706817178393,
 0.5546574405534996,
 0.45665644148383766,
 0.1055692507591559,
 0.20401999038000185,
 4.659830074636498e-07,
 3.340281106901418e-09,
 0.027847473597112257,
 9.615292160652301e-06,
 0.00279012790127491,
 2.4074340987184243e-08,
 0.7076955413350989,
 0.5958109202161124,
 0.8116267771882951,
 7.942180292099096e-08,
 1.6286618425702891e-06,
 0.5343571795753566,
 0.005967691589713594,
 2.5562371534704206e-08,
 4.208752437393755e-07,
 7.037292725339391e-07,
 0.010255546312181027,
 2.7570987711335064e-07,
 4.0584399113474384e-07]
```

[710]:
```
scores=[]
for sco in iou_scores_32:
    if sco!='N/A':
        scores.append(sco)
miou_32=np.mean(scores)
print("mIoU for FCN32 netowrk = {}".format(miou_32))
```

```
mIoU for FCN32 netowrk = 0.1821683180855957
```

[686]:
```
fig, ax = plt.subplots(figsize =(16, 9))
ax.barh(x_scale, iou_scores_32, align='center')
ax.invert_yaxis()
plt.show()
```



[458]:
```
#dummy code
loss = nn.CrossEntropyLoss()
inp=y_pred_32
target=y1.long().squeeze(1)
target_sub=target-1
print("input",inp.shape)
print("target",target.shape)
print("target_sub",target_sub.shape)
print("input",inp)
print("target",target)
print("target_sub",target_sub)

output = loss(inp,target_sub)
print("loss",output)
```

```
input torch.Size([1, 34, 374, 1238])
target torch.Size([1, 374, 1238])
target_sub torch.Size([1, 374, 1238])
input tensor([[[[ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          …,
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014]],

         [[ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
```

```
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          …,
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014],
          [ 0.0014,  0.0014,  0.0014,  …,  0.0014,  0.0014,  0.0014]],

         [[-0.0012, -0.0012, -0.0012,  …, -0.0012, -0.0012, -0.0012],
          [-0.0012, -0.0012, -0.0012,  …, -0.0012, -0.0012, -0.0012],
          [-0.0012, -0.0012, -0.0012,  …, -0.0012, -0.0012, -0.0012],
          …,
          [-0.0012, -0.0012, -0.0012,  …, -0.0012, -0.0012, -0.0012],
          [-0.0012, -0.0012, -0.0012,  …, -0.0012, -0.0012, -0.0012],
          [-0.0012, -0.0012, -0.0012,  …, -0.0012, -0.0012, -0.0012]],

         …,

         [[ 0.0015,  0.0015,  0.0015,  …,  0.0015,  0.0015,  0.0015],
          [ 0.0015,  0.0015,  0.0015,  …,  0.0015,  0.0015,  0.0015],
          [ 0.0015,  0.0015,  0.0015,  …,  0.0015,  0.0015,  0.0015],
          …,
          [ 0.0015,  0.0015,  0.0015,  …,  0.0015,  0.0015,  0.0015],
          [ 0.0015,  0.0015,  0.0015,  …,  0.0015,  0.0015,  0.0015],
          [ 0.0015,  0.0015,  0.0015,  …,  0.0015,  0.0015,  0.0015]],

         [[-0.0013, -0.0013, -0.0013,  …, -0.0013, -0.0013, -0.0013],
          [-0.0013, -0.0013, -0.0013,  …, -0.0013, -0.0013, -0.0013],
          [-0.0013, -0.0013, -0.0013,  …, -0.0013, -0.0013, -0.0013],
          …,
          [-0.0013, -0.0013, -0.0013,  …, -0.0013, -0.0013, -0.0013],
          [-0.0013, -0.0013, -0.0013,  …, -0.0013, -0.0013, -0.0013],
          [-0.0013, -0.0013, -0.0013,  …, -0.0013, -0.0013, -0.0013]],

         [[ 0.0008,  0.0008,  0.0008,  …,  0.0008,  0.0008,  0.0008],
          [ 0.0008,  0.0008,  0.0008,  …,  0.0008,  0.0008,  0.0008],
          [ 0.0008,  0.0008,  0.0008,  …,  0.0008,  0.0008,  0.0008],
          …,
          [ 0.0008,  0.0008,  0.0008,  …,  0.0008,  0.0008,  0.0008],
          [ 0.0008,  0.0008,  0.0008,  …,  0.0008,  0.0008,  0.0008],
          [ 0.0008,  0.0008,  0.0008,  …,  0.0008,  0.0008,  0.0008]]]],
        grad_fn=<CopyBackwards>)
target tensor([[[11, 11, 11,  …, 21, 21, 21],
         [11, 11, 11,  …, 21, 21, 21],
         [11, 11, 11,  …, 21, 21, 21],
         …,
         [ 7,  7,  7,  …,  7,  7,  7],
         [ 7,  7,  7,  …,  7,  7,  7],
         [ 7,  7,  7,  …,  7,  7,  7]]])
```

```
target_sub tensor([[[10, 10, 10,  …, 20, 20, 20],
         [10, 10, 10,  …, 20, 20, 20],
         [10, 10, 10,  …, 20, 20, 20],
         …,
         [ 6,  6,  6,  …,  6,  6,  6],
         [ 6,  6,  6,  …,  6,  6,  6],
         [ 6,  6,  6,  …,  6,  6,  6]]])
loss tensor(3.5257, grad_fn=<NllLoss2DBackward>)
```

[ ]:

[ ]: