```
!pip3 install http://download.pytorch.org/whl/cu80/torch-0.3.0.post4-cp36-cp36m-li
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
ERROR: torch-0.3.0.post4-cp36-cp36m-linux_x86_64.whl is not a supported wheel
```

```
!pip3 install torchvision
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/
Requirement already satisfied: torch==1.12.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
```

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data as data
import torchvision
from torchvision import transforms
import torch.optim as optim
import matplotlib.pyplot as plt
import cv2
import os
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, cal
```

```
df = pd.read_csv("/content/gdrive/MyDrive/archive/CamVid/class_dict.csv")
label_dict = dict()
df
for x,rows in enumerate(df.iterrows()):
    rgb = [rows[1]['r'],rows[1]['g'],rows[1]['b']]
    label_dict[x] = rgb
```

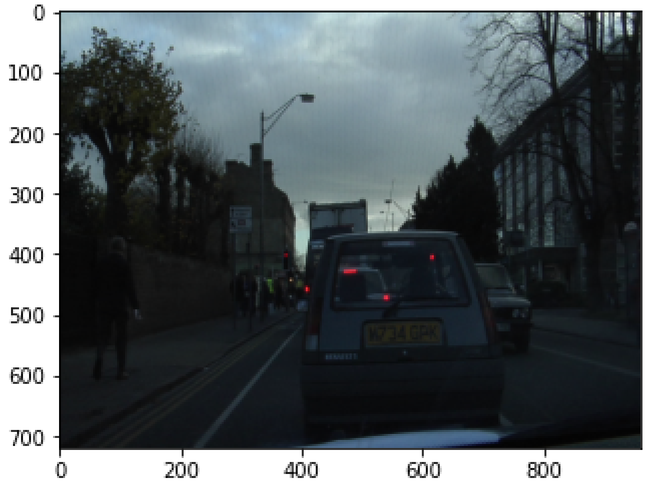
```
label_dict
```

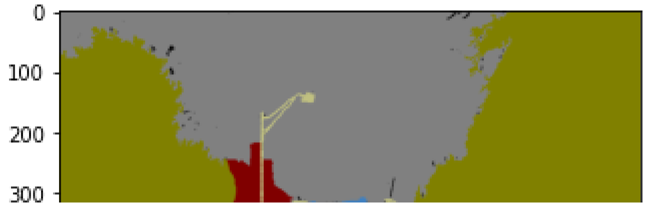
```
{0: [64, 128, 64],
 1: [192, 0, 128],
 2: [0, 128, 192],
 3: [0, 128, 64],
 4: [128, 0, 0],
 5: [64, 0, 128],
 6: [64, 0, 192],
 7: [192, 128, 64],
 8: [192, 192, 128],
 9: [64, 64, 128],
 10: [128, 0, 192],
 11: [192, 0, 64],
 12: [128, 128, 64],
 13: [192, 0, 192],
 14: [128, 64, 64],
 15: [64, 192, 128],
 16: [64, 64, 0],
 17: [128, 64, 128],
 18: [128, 128, 192],
 19: [0, 0, 192],
 20: [192, 128, 128],
 21: [128, 128, 128],
 22: [64, 128, 192],
 23: [0, 0, 64],
 24: [0, 64, 64],
 25: [192, 64, 128],
 26: [128, 128, 0],
 27: [192, 128, 192],
 28: [64, 0, 64],
 29: [192, 192, 0],
 30: [0, 0, 0],
 31: [64, 192, 0]}
```

```
img = cv2.imread("/content/gdrive/MyDrive/archive/CamVid/train/0001TP_009210.png")
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
mask = cv2.imread("/content/gdrive/MyDrive/archive/CamVid/train_labels/0001TP_0092
mask = cv2.cvtColor(mask,cv2.COLOR_BGR2RGB)
print(img .shape)
print(mask.shape)
```

```
    (720, 960, 3)
    (720, 960, 3)
```

```
plt.figure(1)
plt.subplot(111)
plt.imshow(img)
plt.show()
plt.subplot(111)
plt.imshow(mask)
plt.show()
```





```
df
```

	name	r	g	b
0	Animal	64	128	64
1	Archway	192	0	128
2	Bicyclist	0	128	192
3	Bridge	0	128	64
4	Building	128	0	0
5	Car	64	0	128
6	CartLuggagePram	64	0	192
7	Child	192	128	64
8	Column_Pole	192	192	128
9	Fence	64	64	128
10	LaneMkgsDriv	128	0	192
11	LaneMkgsNonDriv	192	0	64
12	Misc_Text	128	128	64

There are total 32 classes in thee total images

```
# def colortogray(cn):
#     cn = np.reshape(cn, (1, 1, 3));
#     cn = cv2.cvtColor(cn, cv2.COLOR_BGR2GRAY);
#     return cn
# #these are the colors that are used for making the boundaries(ie classfication c
# colors = [];
# colors.append(colortogray(np.array([64, 128, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 0, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 128, 0], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 128, 0], dtype = 'uint8')))
# colors.append(colortogray(np.array([0, 0, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 0, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 0, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 128, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 192, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 64, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 0, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 0, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 128, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 0, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 64, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 192, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([0, 64, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 64, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 128, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 0, 0], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 128, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 128, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 128, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 0, 0], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 64, 0], dtype = 'uint8')))
# colors.append(colortogray(np.array([128, 64, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([0, 128, 128], dtype = 'uint8')))
# colors.append(colortogray(np.array([192, 128, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([64, 0, 64], dtype = 'uint8')))
# colors.append(colortogray(np.array([0, 192, 192], dtype = 'uint8')))
# colors.append(colortogray(np.array([0, 0, 0], dtype = 'uint8')))
# colors.append(colortogray(np.array([0, 192, 64], dtype = 'uint8')))

# def class_pixel(label_img):
#      #      if label_img[0].any() == c[0] and label_img[1].any() == c[1] and labe
#       #for i in range(128):
#           #for j in range(128):
#             #for k in range(3):
#              # if label_img[k, i, j].any() == c.any():
#                   #class_pix = index
#     #return class_pix

#     class_pix = np.ones([128, 128, 1], dtype = int);
#     for index, c in enumerate(colors):
#         class_pix[label_img == c] = index
#     return class_pix

# # Convert all segmented images into labeled images with appropriate class number

# def label_img_list(img_list):
#     images = []
#     for image in img_list:
#         images.append(class_pixel(image))
#     return images


# transform_img = transforms.Compose([transforms.ToTensor(), transforms.Normalize(
# transform_img_label = transforms.Compose([transforms.ToTensor()])


def adjust_mask(mask,label_dict):
    segmentation_map_list = []
    for x,color in enumerate(label_dict.values()):
        segmentation_map = (mask==color).all(axis=-1)
        segmentation_map=(segmentation_map*1)
        segmentation_map*=x
        segmentation_map_list.append(segmentation_map)

    return np.amax(np.stack(segmentation_map_list,axis=-1),axis=-1)

def convert_n_channels_2_rgb(image,label_dict):
    image = np.amax(image,axis=-1)
    r = np.zeros_like(image).astype(np.uint8)
    g = np.zeros_like(image).astype(np.uint8)
    b = np.zeros_like(image).astype(np.uint8)

    for l in label_dict.keys():
        idx = image==l
        r[idx] = label_dict[l][0]
        r[idx] = label_dict[l][0]
        g[idx] = label_dict[l][1]
        b[idx] = label_dict[l][2]
    return np.stack([r,g,b],axis=-1)


import glob
class CamVidDataset(Dataset):
    def __init__(self,label_dict,IMAGE_PATH,MASK_PATH,transforms,mask_transforms):
        self.image_list = glob.glob(IMAGE_PATH)
        self.label_list = glob.glob(MASK_PATH)
        self.label_dict = label_dict
        self.transform = transforms
        self.mask_transforms = mask_transforms

        self.image_list.sort()
        self.label_list.sort()

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

    def __getitem__(self,idx):
        img = cv2.imread(self.image_list[idx])
        img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
        mask = cv2.imread(self.label_list[idx])
        mask = cv2.cvtColor(mask,cv2.COLOR_BGR2RGB)

        if self.transform:
            img = self.transform(img)

        if self.mask_transforms:
            mask = self.mask_transforms(mask)

        mask = np.array(mask)
        mask = adjust_mask(mask,self.label_dict)
        mask = torch.tensor(mask)
        mask = torch.squeeze(mask,dim=0)
        return img,mask


transform=transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize((128,128)),
        transforms.ToTensor(),
#         transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
    ])

mask_transforms = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize((128,128))
    ])


IMAGE_PATH = "/content/gdrive/MyDrive/archive/CamVid/train/*.png"
MASK_PATH = "/content/gdrive/MyDrive/archive/CamVid/train_labels/*.png"
```

```
VAL_PATH = "/content/gdrive/MyDrive/archive/CamVid/val/*.png"
VAL_MASK = "/content/gdrive/MyDrive/archive/CamVid/val_labels/*.png"

TEST_PATH = "/content/gdrive/MyDrive/archive/CamVid/test/*.png"
TEST_MASK = "/content/gdrive/MyDrive/archive/CamVid/test_labels/*png"



traindataset = CamVidDataset(label_dict,IMAGE_PATH,MASK_PATH,transform,mask_transf
# trainloader = DataLoader(train_dataset,batch_size = 32,shuffle = True)

valdataset = CamVidDataset(label_dict,VAL_PATH,VAL_MASK,transform,mask_transforms)
# val_loader = DataLoader(val_dataset,batch_size = 32,shuffle = True)
# dataset = ConcatDataset([train_dataset,val_dataset])
testdataset = CamVidDataset(label_dict,TEST_PATH,TEST_MASK,transform,mask_transfor


train_loader = data.DataLoader(traindataset, batch_size = 1, shuffle=True,  num_wo
val_loader = data.DataLoader(valdataset, batch_size = 1, shuffle=True,  num_worker
test_loader = data.DataLoader(testdataset, batch_size = 1, shuffle=True,  num_work
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:566: Us
  cpuset_checked))
```

Defining the Unet model, the first part of the U is made by conv2d class and the second part is made by convTranspose2d.

```
class u_net(nn.Module):
    def __init__(self):
      super().__init__()
      self.conv1 = nn.Conv2d(3, 64, 3)
      self.conv2 = nn.Conv2d(64, 128, 3)
      self.conv3 = nn.Conv2d(128, 256, 3)
      self.conv4 = nn.Conv2d(256, 512, 3)
      self.conv5 = nn.Conv2d(512, 1024, 3)
      self.conv6 = nn.Conv2d(1024, 512, 3)
      self.conv7 = nn.Conv2d(512, 512, 3)
      self.conv8 = nn.Conv2d(512, 256, 3)
      self.conv9 = nn.Conv2d(256, 256, 3)
      self.conv10 = nn.Conv2d(256, 128, 3)
      self.conv11 = nn.Conv2d(128, 128, 3)
      self.conv12 = nn.Conv2d(64, 64, 3)
      self.b1 = nn.BatchNorm2d(64)
      self.b2 = nn.BatchNorm2d(128)
      self.b3 = nn.BatchNorm2d(256)
      self.b4 = nn.BatchNorm2d(512)
      self.b5 = nn.BatchNorm2d(1024)
      self.convT1 = nn.ConvTranspose2d(1024, 512, 2, 2)
      self.convT2 = nn.ConvTranspose2d(512, 256, 2, 2)
      self.convT3 = nn.ConvTranspose2d(256, 256, 2, 2)
      self.convT4 = nn.ConvTranspose2d(128, 64, 2, 2)
      self.convT5 = nn.ConvTranspose2d(64, 32, 2, 2)
      self.pool1 = nn.MaxPool2d(2, 2)

    def forward(self, x):
      x = F.relu(self.b1(self.conv1(x)))
      x = F.relu(self.b1(self.conv12(x)))
      x = F.relu(self.b2(self.conv2(x)))
      x = self.pool1(x)
      x = F.relu(self.b2(self.conv11(x)))
      x = F.relu(self.b3(self.conv3(x)))
      x1 = x
      x1 = x1[:, :, int((58 - 24)/2) : int((58 + 24)/2), int((58 - 24)/2) : int((5
      x = self.pool1(x)
      x = F.relu(self.b3(self.conv9(x)))
      x = F.relu(self.b4(self.conv4(x)))
      x2 = x
      x2 = x2[:, :, int((25 - 16)/2) : int((25 + 16)/2), int((25 - 16)/2) : int((2
      x = self.pool1(x)
      x = F.relu(self.b4(self.conv7(x)))
      x = F.relu(self.b5(self.conv5(x)))
      x = self.b4(self.convT1(x))
      x = torch.cat((x2, x), dim = 1)
      x = F.relu(self.b4(self.conv6(x)))
      x = F.relu(self.b4(self.conv7(x)))
      x = self.b3(self.convT2(x))
      x = torch.cat((x1, x), dim = 1)
      x = F.relu(self.b3(self.conv8(x)))
      x = F.relu(self.b3(self.conv9(x)))
      x = self.b3(self.convT3(x))
      x = F.relu(self.b2(self.conv10(x)))
      x = F.relu(self.b2(self.conv11(x)))
      x = F.relu(self.b2(self.conv11(x)))
      x = F.relu(self.b2(self.conv11(x)))
      x = self.b1(self.convT4(x))
      x = self.convT5(x)
      del x1
      del x2
      return x
net = u_net()
print(net)
```

```
u_net(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1))
  (conv6): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1))
  (conv7): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1))
  (conv8): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1))
  (conv9): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
  (conv10): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv11): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv12): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (b1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
  (b2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
  (b3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
  (b4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
  (b5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running
  (convT1): ConvTranspose2d(1024, 512, kernel_size=(2, 2), stride=(2, 2))
  (convT2): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
  (convT3): ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2, 2))
  (convT4): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (convT5): ConvTranspose2d(64, 32, kernel_size=(2, 2), stride=(2, 2))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
)
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
UNET = u_net()
UNET.to(device)
```

```
    cpu
    u_net(
      (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1))
      (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
      (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
      (conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1))
      (conv5): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1))
      (conv6): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1))
      (conv7): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1))
      (conv8): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1))
      (conv9): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
      (conv10): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1))
      (conv11): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
      (conv12): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
      (b1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
      (b2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
      (b3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
      (b4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
      (b5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
      (convT1): ConvTranspose2d(1024, 512, kernel_size=(2, 2), stride=(2, 2))
      (convT2): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
      (convT3): ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2, 2))
      (convT4): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
      (convT5): ConvTranspose2d(64, 32, kernel_size=(2, 2), stride=(2, 2))
      (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
    )
```

```
loss1 = nn.CrossEntropyLoss()
optimizer = optim.Adam(UNET.parameters(), lr = 0.0001, betas = (0.9, 0.999), eps =
```

```
for epoch in range(100):
  running_loss_train = 0
  running_loss_val = 0
  sum = 0
  for i, data in enumerate(train_loader):
      inputs, labels = data;
      if labels.size() == torch.Size([1, 1, 128, 128]):
        labels = labels.reshape(1, 128, 128)
      inputs, labels = inputs.to(device), labels.to(device)
      optimizer.zero_grad()
      outputs = UNET(inputs)
      loss_train= loss1(outputs, labels)
      running_loss_train=(loss_train.item()*inputs.size(0))
      sum = sum + running_loss_train
      loss_train.backward()
      optimizer.step()
      print("Epoch:{epoch} Train loss:{train_loss}".format(epoch = epoch+1, train_los
    print("avg.train_loss:{avg_loss}".format(avg_loss = sum/len(traindataset)))
```

```
     .
    Epoch:8 Train loss:1.223685383796692
    Epoch:8 Train loss:1.2542178630828857
    Epoch:8 Train loss:0.5777255296707153
    Epoch:8 Train loss:1.4297538995742798
    Epoch:8 Train loss:1.1071542501449585
    Epoch:8 Train loss:1.4176884889602661
    Epoch:8 Train loss:1.538763165473938
    Epoch:8 Train loss:0.6367810368537903
    Epoch:8 Train loss:0.9905219078063965
    Epoch:8 Train loss:1.2213284969329834
    Epoch:8 Train loss:1.786043643951416
    Epoch:8 Train loss:1.8317824602127075
    Epoch:8 Train loss:0.9234600067138672
    Epoch:8 Train loss:1.1210198402404785
    Epoch:8 Train loss:0.9503546357154846
    Epoch:8 Train loss:1.638059377670288
    Epoch:8 Train loss:0.9450034499168396
    Epoch:8 Train loss:0.8021291494369507
    Epoch:8 Train loss:1.5742720365524292
    Epoch:8 Train loss:0.9983381032943726
    Epoch:8 Train loss:1.5073297023773193
    Epoch:8 Train loss:1.7611501216888428
    Epoch:8 Train loss:1.1384538412094116
    Epoch:8 Train loss:1.0294276475906372
    Epoch:8 Train loss:1.3683900833129883
    Epoch:8 Train loss:0.8787370920181274
    Epoch:8 Train loss:1.1217515468597412
    Epoch:8 Train loss:0.9133260846138
    Epoch:8 Train loss:1.2565563917160034
    Epoch:8 Train loss:1.2385839223861694
    Epoch:8 Train loss:1.036454439163208
    Epoch:8 Train loss:1.2577763795852661
    Epoch:8 Train loss:0.9936730265617371
    Epoch:8 Train loss:1.1554046869277954
    Epoch:8 Train loss:1.139275312423706
    Epoch:8 Train loss:0.6548929810523987
    Epoch:8 Train loss:0.8487564325332642
    Epoch:8 Train loss:1.199241042137146
    Epoch:8 Train loss:0.9186874032020569
    Epoch:8 Train loss:0.9957265257835388
    Epoch:8 Train loss:0.902000367641449
    Epoch:8 Train loss:1.1270873546600342
    Epoch:8 Train loss:1.5105595588684082
    Epoch:8 Train loss:1.2820963859558105
    Epoch:8 Train loss:1.0667760372161865
```

```
Epoch:8 Train loss:0.9328413009643555
Epoch:8 Train loss:1.729661464691162
Epoch:8 Train loss:0.7813495397567749
Epoch:8 Train loss:1.718488335609436
Epoch:8 Train loss:0.8977041244506836
Epoch:8 Train loss:0.8666271567344666
Epoch:8 Train loss:1.5102282762527466
Epoch:8 Train loss:1.1825392246246338
Epoch:8 Train loss:0.9216552972793579
Epoch:8 Train loss:1.0225152969360352
Epoch:8 Train loss:1.253833293914795
Epoch:8 Train loss:1.5889148712158203
Epoch:8 Train loss:1.0288312435150146
```

```
#Saving the model to the gdrive.
PATH = '/content/gdrive/MyDrive/archive/CamVid/saved1.pth'
torch.save(UNET.state_dict(),PATH); # A state_dict is simply a Python dictionary
# object that maps each layer to its parameter tensor.
```

! 24m 33s completed at 20:36