SegNet implementation in PyTorch

Move the frame to colab [**Done**]

Mount data [**done**]

Fix the training function [**Done**]

Report training accuracy [**Done**]

Learning Rate decay and cooldown [**Done**]

Save checkpints [**Done**]

Visualize an output [**Done**]

Plot training curve [**Done**]

Calculate class weights median frequency balancing [**Done**]

Online Data Augmentation (intensity, noise, stretch, center shift) [**Done**]

Quantitive evaluation (accuracy, etc) [**Done**]

Video visualization [**Done**]

Demo on CamVid Seq01, Kitti(preprocessing?) [**Done**]

In [1]:

```
# import necessary libraries
%matplotlib inline
from matplotlib import pyplot as plt
import torch
from torchvision import models
import torchsummary
import torch.nn as nn
import torch.nn.functional as F
import cv2
import numpy as np
```

```
import numpy as np
from torch.utils.data import DataLoader
import time
import os


from tqdm import tqdm_notebook as tqdm
```

In [2]:

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

# !ls gdrive/MyDrive
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/g
drive", force_remount=True).
```

In [ ]:

```
# get the patht to the camvid data
camvid_path = r'/content/gdrive/MyDrive/CamVid'
!ls '/content/gdrive/MyDrive/CamVid'

# get the path to the scripts
!ls gdrive/MyDrive/Segmentation/code

# update imgaug
!pip uninstall imgaug
!pip install imgaug==0.4.0

# access necessary code
from gdrive.MyDrive.Segmentation.code.model import SegNet
import gdrive.MyDrive.Segmentation.code.data_aug as data
```

Global vars

In [4]:

```
# for reproduction
```

```
torch.manual_seed(123)

# Pre config
IN_CHANNELS = 3
NUM_CLASS = 2

# training parameters
NUM_EPOCHS = 200
LEARNING_RATE = 0.1   # will decrease with epoch growing larger
BATCH_SIZE = 16
DEVICE = "cuda" if torch.cuda.is_available else "cpu"
LABEL_WEIGHTS = torch.FloatTensor([0.5, 2])
WEIGHT_DECAY = 0.0005
NUM_WORKERS = 0
PIN_MEMORY = False
```

Reporting/Evaluation Functions

In [5]:

```
def avg_pixelwise_accuracy(model, dataset):

  if not dataset:
    return -1

  # change to eval mode
  model.eval()

  # get the dataloader using large batch since only forward pass required
  loader = DataLoader(dataset=dataset,
                      batch_size=8,
                      )

  correct, total = 0, 0
  for i, batch in enumerate(loader):

    # mount to GPU if available [Need to fix y]
    imgs, labels = batch['X'].to(DEVICE), batch['y'].to(DEVICE)

    # argmax X along dim 1
    out = model(imgs)
    predicted = torch.argmax(out, dim=1)
    total += labels.nelement()
    correct += predicted.eq(labels).sum().item()

  return correct / total
```

Save/Load checkpoints

In [6]:

```
def save_checkpoints(model, filename, epoch, optimizer, loss, train_acc, valid_acc):

  dirname = "/content/gdrive/MyDrive/Segmentation/check_points/large_aug"
  os.makedirs(dirname, exist_ok=True)

  save_path = os.path.join(dirname, filename)
  print(save_path)
  torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss,
            'train_acc': train_acc,
            'valid_acc': valid_acc,
            'transfer_learning': TRANSFER_LEARNING
            }, save_path)


def load_checkpoints(filename, transfer_learning=True):

  dirname = "/content/gdrive/MyDrive/Segmentation/check_points/large_aug"
  file_path = os.path.join(dirname, filename)
  checkpoint = torch.load(file_path)
  if 'transfer_learning' in checkpoint:
    transfer_learning = checkpoint['transfer_learning']
  model = SegNet(IN_CHANNELS, NUM_CLASS, transfer_learning=transfer_learning)
  model.load_state_dict(checkpoint['model_state_dict'])

  print("Model Loaded: Epoch#{}\tLoss:{:.4f}\tTrainAcc:{:.4f}\tValidAcc:{:.4f}".format(
      checkpoint['epoch'],
      checkpoint['loss'],
      checkpoint['train_acc'],
      checkpoint['valid_acc'],
  ))
  return model.to(DEVICE)


# model2 = load_checkpoints("Epoch1_loss59.1495_trainacc31.217_valacc28.866.pth")
# model2
```

Visualization

In [7]:

```
def show_img(img_tensor):

  if len(img_tensor.shape) == 4:
    img = img_tensor.squeeze()
  else:
    img = img_tensor
  img = torch.movedim(img, 0, -1).type(torch.uint8).numpy()[:,:,::-1]
  plt.imshow(img)
  plt.show()

def show_mask(mask_tensor):
  # shape height x width, single channel
  plt.imshow(mask_tensor)
  plt.show()

def show_pred_mask(model, img_tensor):
  if len(img_tensor.shape) == 4:
    source_img = img_tensor
  else:
    source_img = img_tensor.unsqueeze(0).to(DEVICE)

  model.eval()
  out = model(source_img)
  predicted = torch.argmax(out, dim=1)
  predicted = predicted.cpu().data.numpy()[0, :]
  plt.imshow(predicted)
  plt.show()

def show_all(model, img_tensor, mask_tensor, cmap='jet'):

  # regulate the image tensor
  if len(img_tensor.shape) == 4:
    img = img_tensor.squeeze()
    source_img = img_tensor
  else:
    img = img_tensor
    source_img = img_tensor.unsqueeze(0).to(DEVICE)

  # Raw image
  img = torch.movedim(img, 0, -1).type(torch.uint8).numpy()[:,:,::-1]

  # Ground Truth mask
  mask = mask_tensor

  # predicted mask
  model.eval()
  out = model(source_img)
  predicted = torch.argmax(out, dim=1)
  predicted = predicted.cpu().data.numpy()[0, :]
```

```
fig, axs = plt.subplots(1, 5, figsize=(20, 20), constrained_layout=True)

axs[0].imshow(img)
axs[0].set_title("Image")

axs[1].imshow(mask)
axs[1].set_title("Ground Truth")

axs[2].imshow(predicted)
axs[2].set_title("SegNet Outcome")

img[:, :, 1] += (predicted * 255 * 0.15).astype(np.uint8)
img[:, :, 0] += (predicted * 255 * 0.3).astype(np.uint8)
axs[3].imshow(img, vmin=0, vmax=255)
# axs[3].imshow(predicted, cmap=cmap, alpha=0.3)
axs[3].set_title("SegNet Overlay")

axs[4].imshow((predicted - mask.numpy()) ** 2)
axs[4].set_title("Error")

plt.show()
```

Dataset Preparation

In [8]:

```
# load the dataset
X_train, y_train = data.load_data(camvid_path+"/train", im_height=360, im_width=480, aug_copies=5)
print("Loaded {} training samples".format(len(X_train)))


X_valid, y_valid = data.load_data(camvid_path+"/val", im_height=360, im_width=480)
print("Loaded {} validation samples".format(len(X_valid)))


X_test, y_test = data.load_data(camvid_path+"/test", im_height=360, im_width=480)
print("Loaded {} testing samples".format(len(X_test)))
```

```
Bkg : Lane = 43570264.0 : 19847338.0
Loaded 2202 training samples

Bkg : Lane = 12400839.0 : 5051963.0
Loaded 101 validation samples

Bkg : Lane = 29962033.0 : 10300369.0
Loaded 233 testing samples
```

Train

In [9]:

```
# Observe an image from KiTTi
kitti_observe = cv2.imread("/content/gdrive/MyDrive/KiTTi_dataset/testing_crop/um_000052.png")
kitti_observe_tensor = torch.from_numpy(np.moveaxis(kitti_observe, -1, 0))
```

In [10]:

```
# train function
from torch.utils.data import Dataset
class CamVidDataset(Dataset):
    def __init__(self, imgs, masks):
        self.X =imgs
        self.y = masks
    def __len__(self):
        return len(self.X[:, 0, 0, 0])

    def __getitem__(self, index):
        data = {'X': torch.FloatTensor(self.X[index, :, :, :]), 'y': torch.LongTensor(self.y[index, :,
        return data

def train(model, optimizer, loss_fn, X_train, y_train, X_valid=None, y_valid=None):

  # get the dataloader
  train_dataset = CamVidDataset(X_train, y_train)
  train_loader = DataLoader(dataset=train_dataset,
                            batch_size=BATCH_SIZE,
                            num_workers=NUM_WORKERS,
                            pin_memory=False,
                            shuffle=True,
                            drop_last=True,
                            worker_init_fn=None
                            )

  valid_dataset = CamVidDataset(X_valid, y_valid) if X_valid is not None else None


  # for plotting and logging
  epoch_lst, iters, losses, train_acc_lst, val_acc_lst, iter_counter = [], [], [], [], [], 0

  # for checkpoints
  checkpoint_path_template = "Epoch{}_loss{:.4f}_trainacc{:.3f}_valacc{:.3f}.pth"
  acc_recoder, save_gap = 96, 0.2

  # start training
  for epoch in range(NUM_EPOCHS):

    # visualize an segmentation every 10 epoch
    if epoch % 4 == 0:
      show_all(model, X_valid[0, :], y_valid[0, :])
      show_all(model, kitti_observe_tensor.float(), kitti_observe_tensor.float()[0, :])

    # Learning Rate Decay [Optional]
```

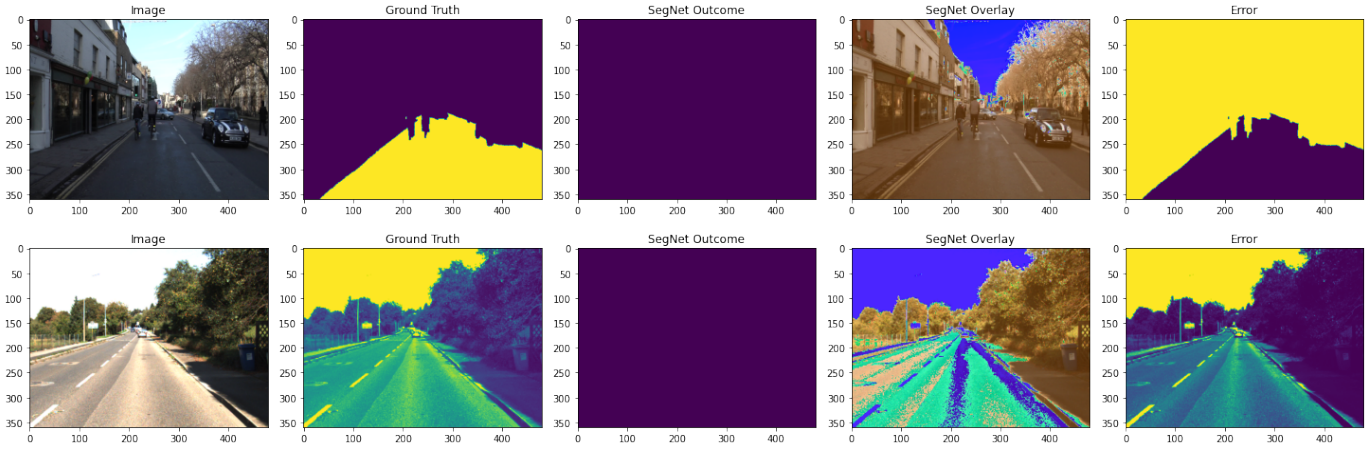
```
    # Learning Rate Decay [Optional]
    if (epoch + 1) % 15 == 0:
      optimizer.param_groups[0]['lr'] /= 3
      print("<================ Learning Rate {} -> {}===== ==========>".format(optimizer.param_groups[(

    epoch_loss = 0
    t_start = time.time()
    for i, batch in enumerate(train_loader):

      # mount to GPU if available [Need to fix y]
      imgs, labels = batch['X'].to(DEVICE), batch['y'][:, :, :].to(DEVICE)

      # change the mode to training mode and step training
      model.train()
      out = model(imgs)
      loss = loss_fn(out, labels)   # note: soft-max should not be used here since it's included in n
      loss.backward()
      optimizer.step()
      optimizer.zero_grad()

      # save the current training information, TODO
      losses.append(loss)
      epoch_loss += loss
      iters.append(iter_counter)
      iter_counter += 1


    delta = time.time() - t_start

    # train_acc = avg_pixelwise_accuracy(model, train_dataset)
    train_acc = avg_pixelwise_accuracy(model, train_dataset) * 100
    valid_acc = avg_pixelwise_accuracy(model, valid_dataset) * 100
    print("Epoch #{}\tLoss: {:.6f}\tTrain Acc: {:.3f}%\tValid Acc: {:.3f}%\tTime: {:.2f}s".format(epoc

    train_acc_lst.append(train_acc)
    val_acc_lst.append(valid_acc)
    epoch_lst.append(epoch)

    # Save checkpoints
    if valid_acc > acc_recoder + save_gap:
      acc_recoder = valid_acc
      checkpoint_name = checkpoint_path_template.format(epoch+1, epoch_loss, train_acc, valid_acc)
      save_checkpoints(model, checkpoint_name, epoch+1, optimizer, epoch_loss, train_acc, valid_acc)

# Plot the curve
plt.title("Learning Curve")
plt.plot(iters, losses, label="Train")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()

plt.title("Learning Curve")
plt.plot(epoch_lst, train_acc_lst, label="Train")
plt.plot(epoch_lst, val_acc_lst, label="Validation")
```

```
    plt.xlabel("Epoch")
    plt.ylabel("Pixelwise Accuracy")
    plt.legend(loc='best')
    plt.show()
```

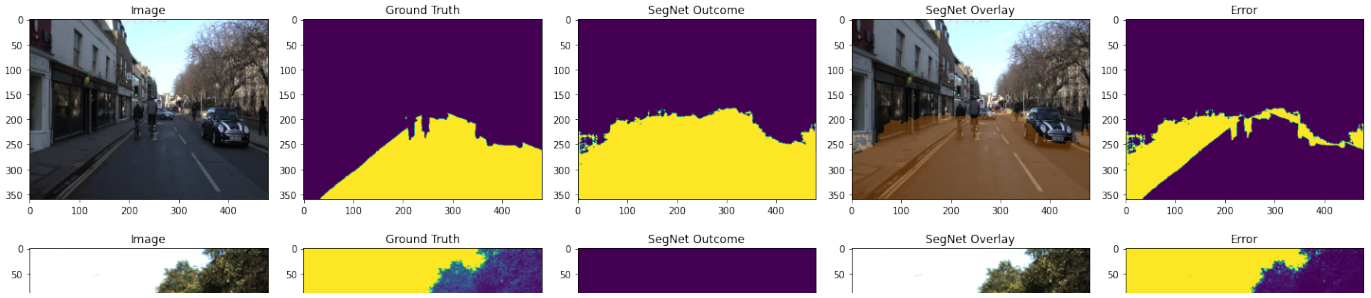
Train from scratch

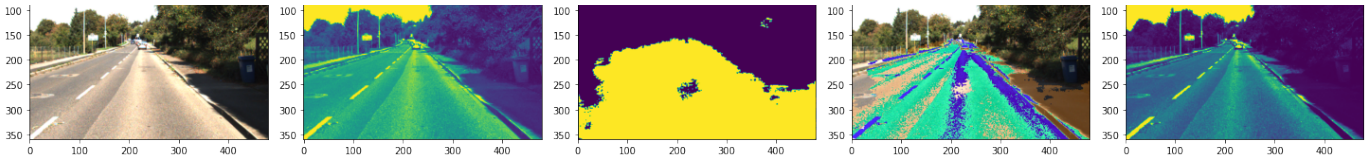
In [11]:

```
torch.cuda.empty_cache()

TRANSFER_LEARNING = False
model = SegNet(3,2, transfer_learning=TRANSFER_LEARNING).to(DEVICE)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
loss = nn.CrossEntropyLoss(weight=LABEL_WEIGHTS).to(DEVICE)


# Launch training
train(model, optimizer, loss, X_train, y_train, X_valid, y_valid)
```

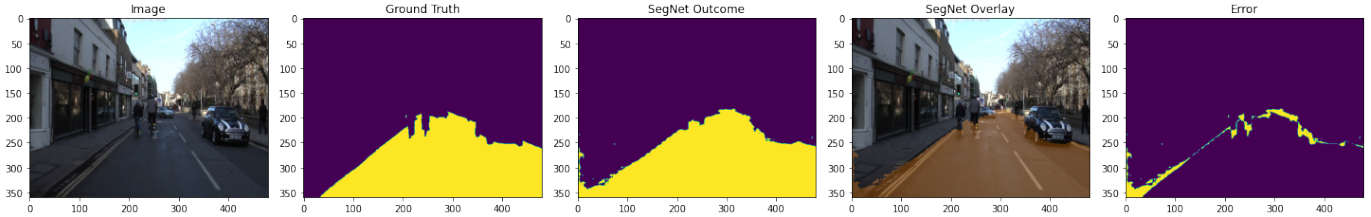


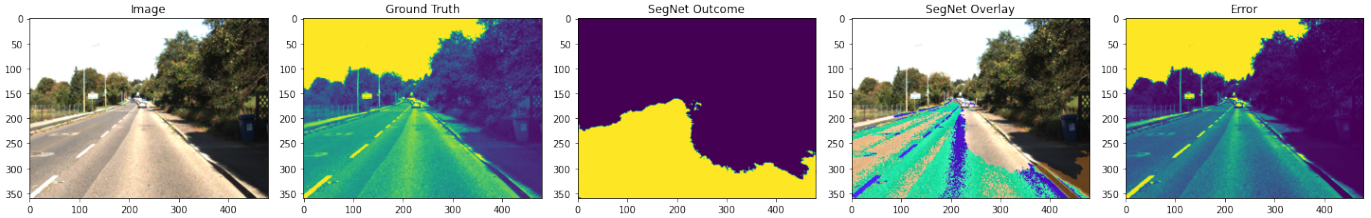

```
Epoch #1        Loss: 113.761894        Train Acc: 86.099%      Valid Acc: 88.964%      Time: 85.88s
Epoch #2        Loss: 41.867458 Train Acc: 90.610%      Valid Acc: 89.851%      Time: 85.89s
Epoch #3        Loss: 23.835253 Train Acc: 93.763%      Valid Acc: 93.365%      Time: 85.93s
Epoch #4        Loss: 19.350954 Train Acc: 90.274%      Valid Acc: 86.047%      Time: 85.89s
```



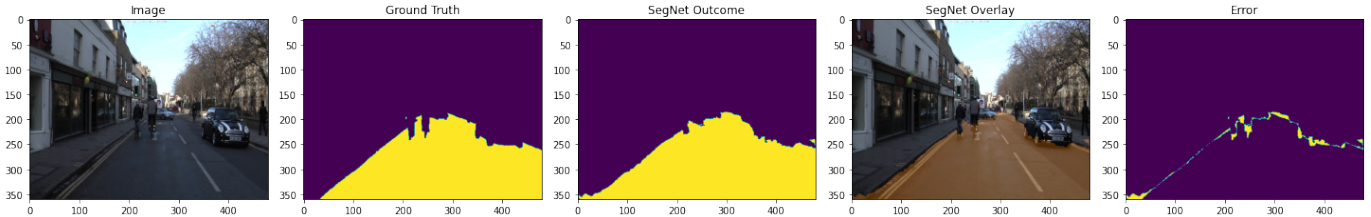


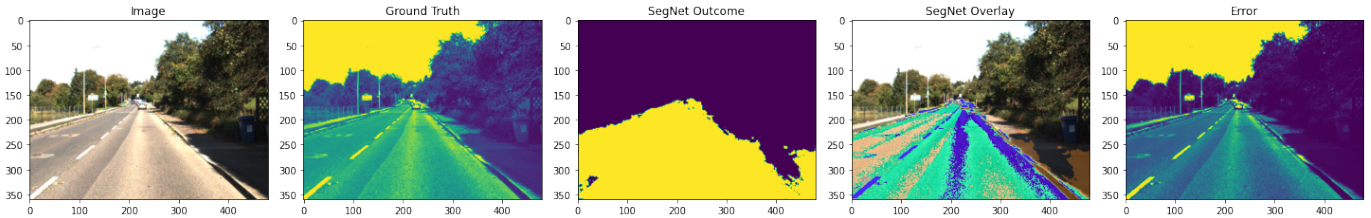
```
Epoch #5        Loss: 17.510023 Train Acc: 89.190%      Valid Acc: 86.207%      Time: 85.87s
Epoch #6        Loss: 17.662569 Train Acc: 92.100%      Valid Acc: 87.802%      Time: 85.87s
Epoch #7        Loss: 16.499863 Train Acc: 94.074%      Valid Acc: 94.786%      Time: 85.89s
Epoch #8        Loss: 15.529844 Train Acc: 93.574%      Valid Acc: 94.386%      Time: 85.88s
```





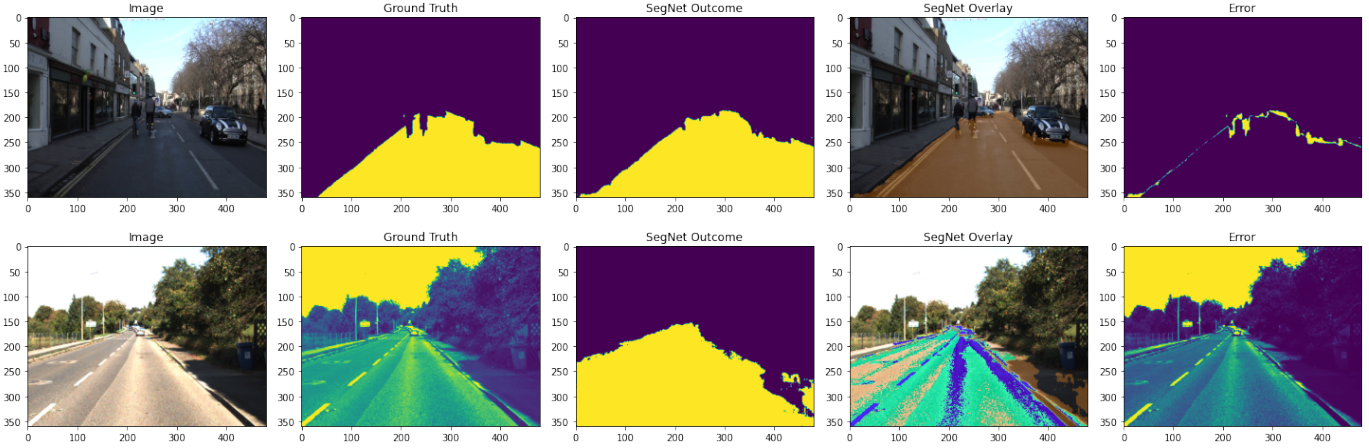

```
Epoch #9        Loss: 14.994182 Train Acc: 96.063%      Valid Acc: 97.197%      Time: 85.89s
/content/gdrive/MyDrive/Segmentation/check_points/large_aug/Epoch9_loss14.9942_trainacc96.063_valacc97
.197.pth
Epoch #10       Loss: 14.117742 Train Acc: 94.361%      Valid Acc: 95.133%      Time: 85.86s
Epoch #11       Loss: 13.702608 Train Acc: 95.265%      Valid Acc: 96.897%      Time: 85.90s
Epoch #12       Loss: 13.000116 Train Acc: 95.437%      Valid Acc: 97.075%      Time: 85.89s
```



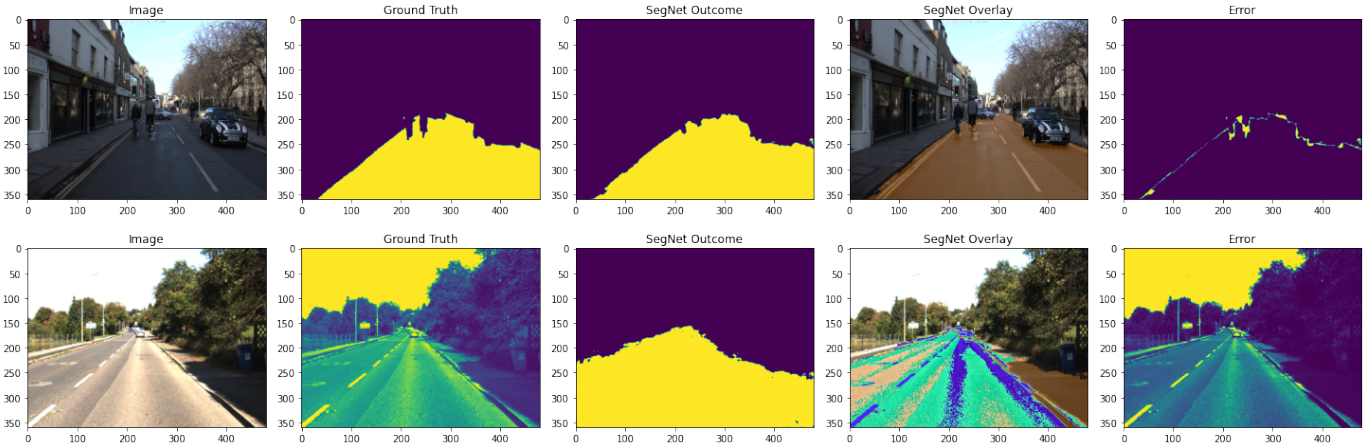



```
Epoch #13       Loss: 12.387345 Train Acc: 93.363%      Valid Acc: 96.052%      Time: 85.89s
Epoch #14       Loss: 12.141721 Train Acc: 95.798%      Valid Acc: 97.617%      Time: 85.90s
/content/gdrive/MyDrive/Segmentation/check_points/large_aug/Epoch14_loss12.1417_trainacc95.798_valacc9
7.617.pth
<=============== Learning Rate 0.1 -> 0.03333333333333333===== ==========>
```

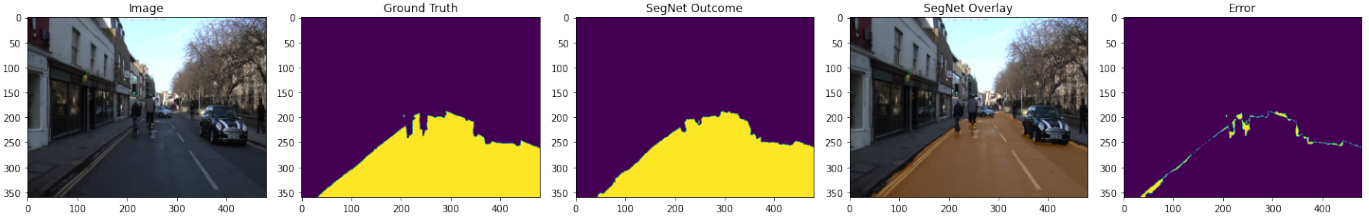
```
Epoch #15       Loss: 10.643561 Train Acc: 96.307%      Valid Acc: 98.070%      Time: 85.90s
/content/gdrive/MyDrive/Segmentation/check_points/large_aug/Epoch15_loss10.6436_trainacc96.307_valacc9
8.070.pth
Epoch #16       Loss: 10.597965 Train Acc: 95.758%      Valid Acc: 97.692%      Time: 85.92s
```

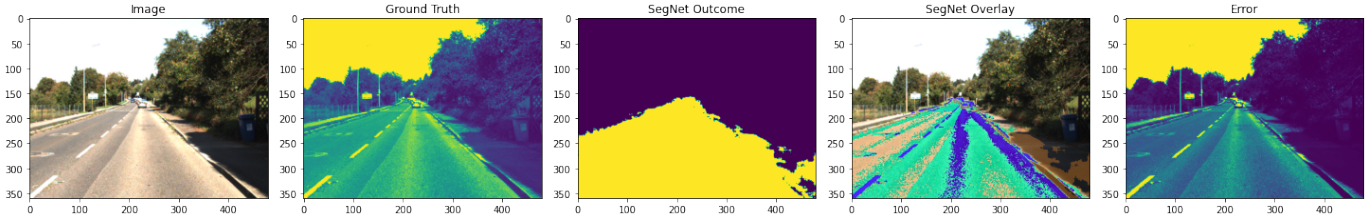



```
Epoch #17       Loss: 10.195443 Train Acc: 96.019%      Valid Acc: 97.674%      Time: 85.90s
Epoch #18       Loss: 9.994283  Train Acc: 96.044%      Valid Acc: 97.943%      Time: 85.90s
Epoch #19       Loss: 9.911497  Train Acc: 96.394%      Valid Acc: 97.950%      Time: 85.89s
Epoch #20       Loss: 9.835297  Train Acc: 95.538%      Valid Acc: 97.824%      Time: 85.93s
```

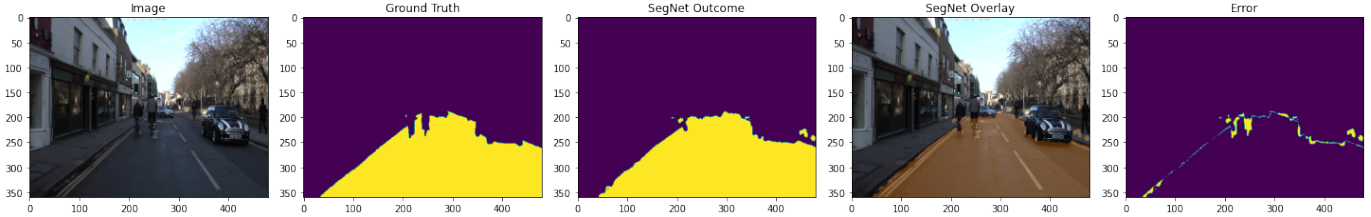


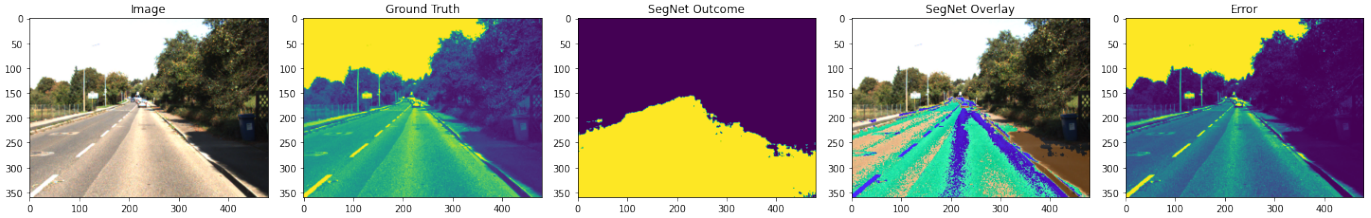

```
Epoch #21       Loss: 9.657845  Train Acc: 95.388%      Valid Acc: 97.402%      Time: 85.93s
Epoch #22       Loss: 9.526237  Train Acc: 95.580%      Valid Acc: 97.826%      Time: 85.95s
Epoch #23       Loss: 9.347882  Train Acc: 96.856%      Valid Acc: 98.201%      Time: 85.94s
Epoch #24       Loss: 9.458021  Train Acc: 96.388%      Valid Acc: 98.263%      Time: 85.89s
```



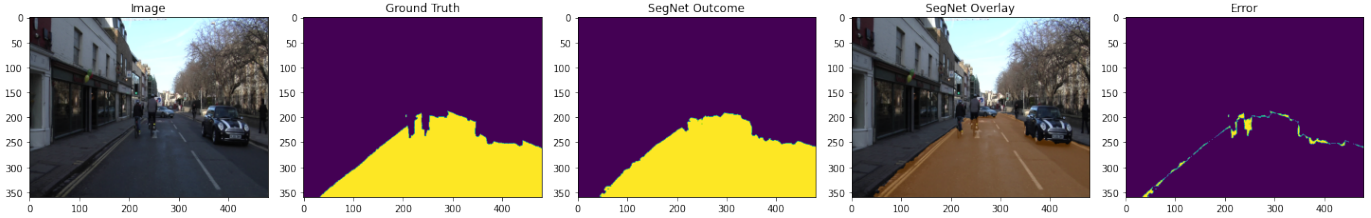


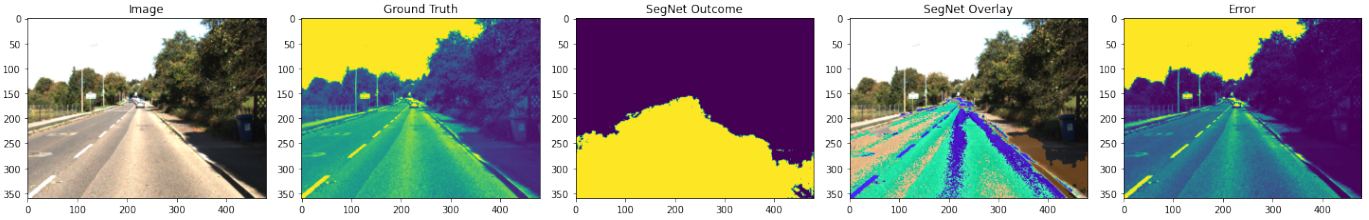

```
Epoch #25       Loss: 8.865716  Train Acc: 96.090%      Valid Acc: 98.095%      Time: 85.92s
Epoch #26       Loss: 8.797677  Train Acc: 96.723%      Valid Acc: 98.120%      Time: 85.95s
Epoch #27       Loss: 8.607855  Train Acc: 96.638%      Valid Acc: 97.870%      Time: 85.92s
Epoch #28       Loss: 8.652934  Train Acc: 96.506%      Valid Acc: 97.880%      Time: 85.93s
```



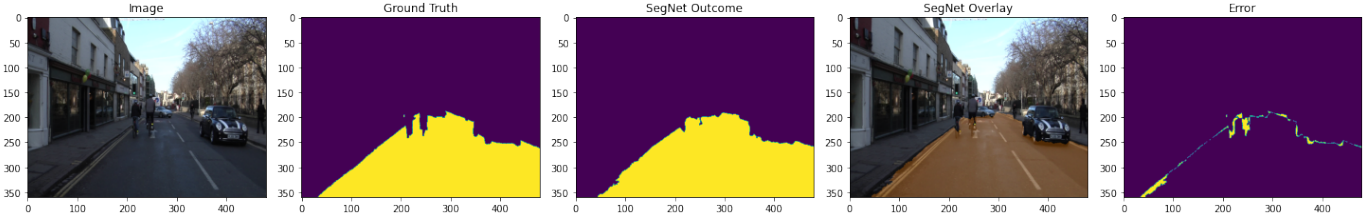


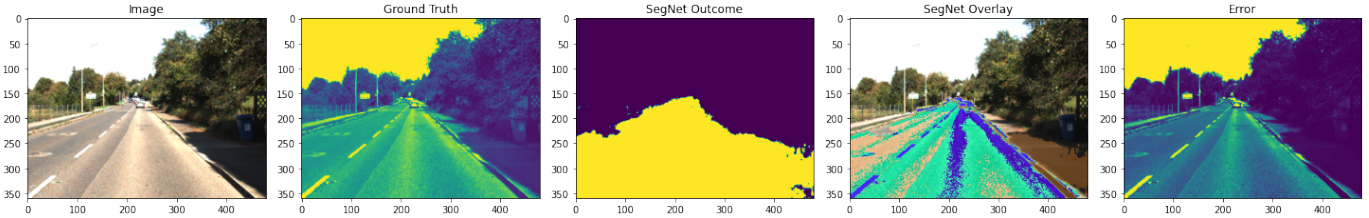

```
Epoch #29       Loss: 8.706271  Train Acc: 97.282%      Valid Acc: 98.383%      Time: 85.91s
/content/gdrive/MyDrive/Segmentation/check_points/large_aug/Epoch29_loss8.7063_trainacc97.282_valacc98
.383.pth
<================ Learning Rate 0.03333333333333333 -> 0.011111111111111112===== ==========>
Epoch #30       Loss: 7.423449  Train Acc: 97.307%      Valid Acc: 98.320%      Time: 85.89s
Epoch #31       Loss: 7.090017  Train Acc: 97.660%      Valid Acc: 98.425%      Time: 85.91s
Epoch #32       Loss: 6.966231  Train Acc: 97.112%      Valid Acc: 98.174%      Time: 85.89s
```



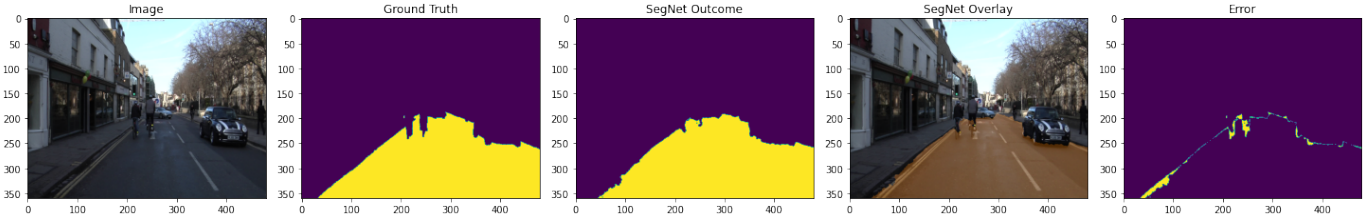


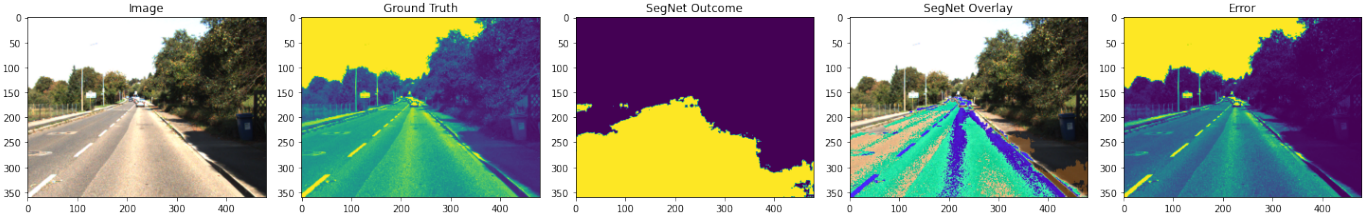

```
Epoch #33       Loss: 6.798064  Train Acc: 97.556%       Valid Acc: 98.417%       Time: 85.90s
Epoch #34       Loss: 6.549056  Train Acc: 97.632%       Valid Acc: 98.395%       Time: 85.91s
Epoch #35       Loss: 6.710833  Train Acc: 97.555%       Valid Acc: 98.320%       Time: 85.90s
Epoch #36       Loss: 6.522848  Train Acc: 97.153%       Valid Acc: 98.134%       Time: 85.90s
```





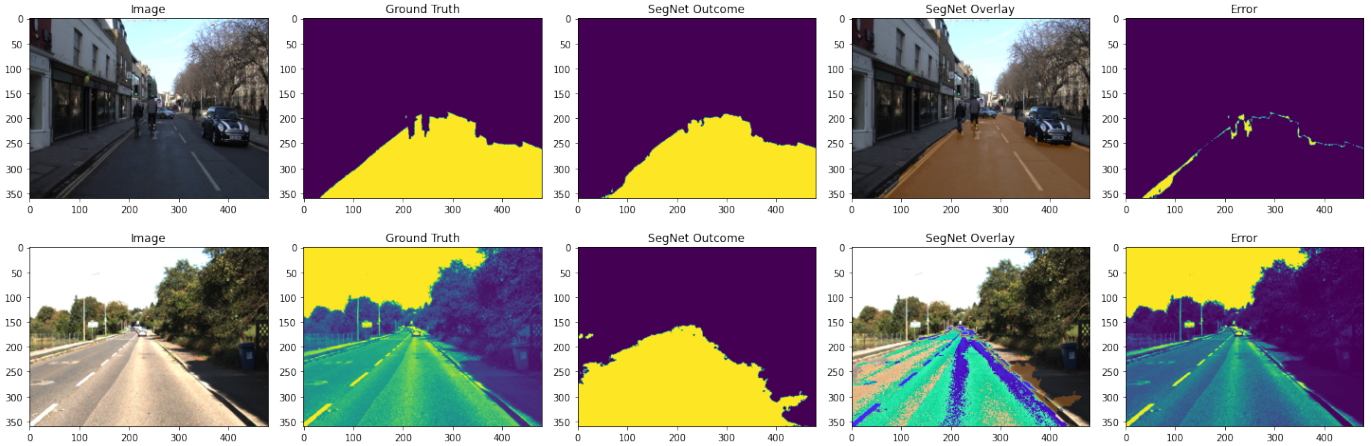

```
Epoch #37       Loss: 6.420828  Train Acc: 97.607%       Valid Acc: 98.191%       Time: 85.90s
Epoch #38       Loss: 6.245625  Train Acc: 97.786%       Valid Acc: 98.446%       Time: 85.92s
Epoch #39       Loss: 6.244062  Train Acc: 97.326%       Valid Acc: 98.070%       Time: 85.93s
Epoch #40       Loss: 6.117337  Train Acc: 97.685%       Valid Acc: 98.306%       Time: 85.91s
```





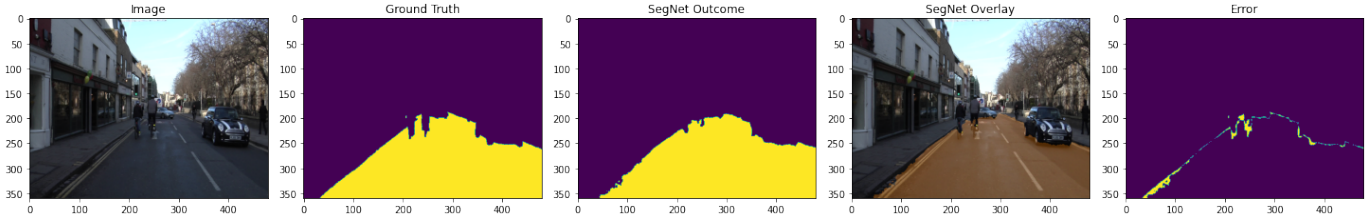

```
Epoch #41       Loss: 6.034916  Train Acc: 97.942%       Valid Acc: 98.434%       Time: 85.92s
Epoch #42       Loss: 5.891027  Train Acc: 97.912%       Valid Acc: 98.394%       Time: 85.94s
Epoch #43       Loss: 5.976671  Train Acc: 97.453%       Valid Acc: 98.246%       Time: 85.91s
```

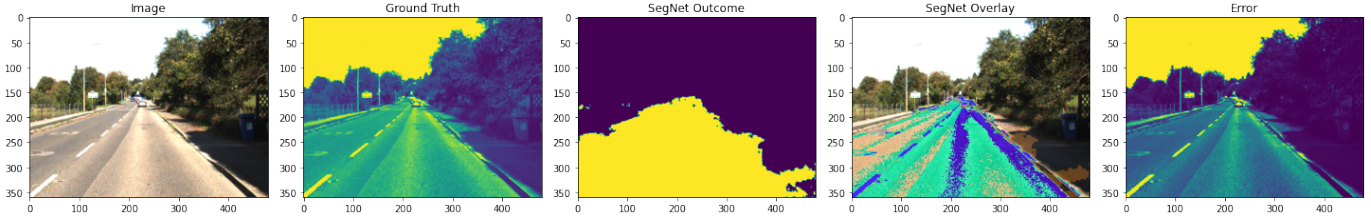
Epoch #44 Loss: 5.854548 Train Acc: 98.041% Valid Acc: 98.296% Time: 85.96s



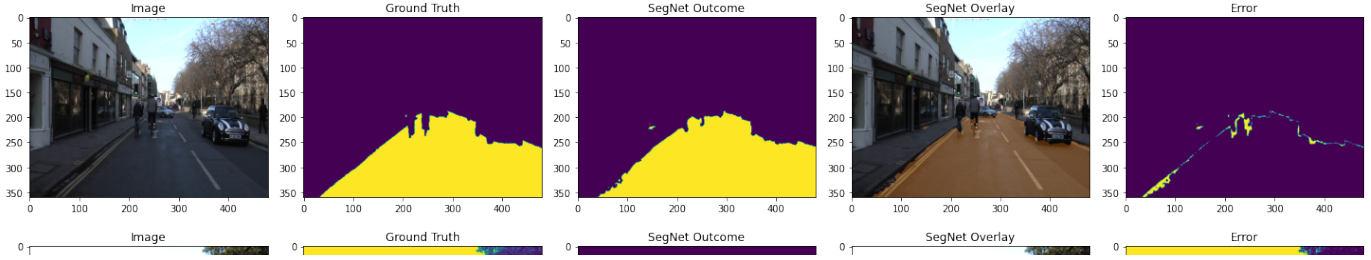
<=========== Learning Rate 0.011111111111111112 -> 0.003703703703703704===== ==========>

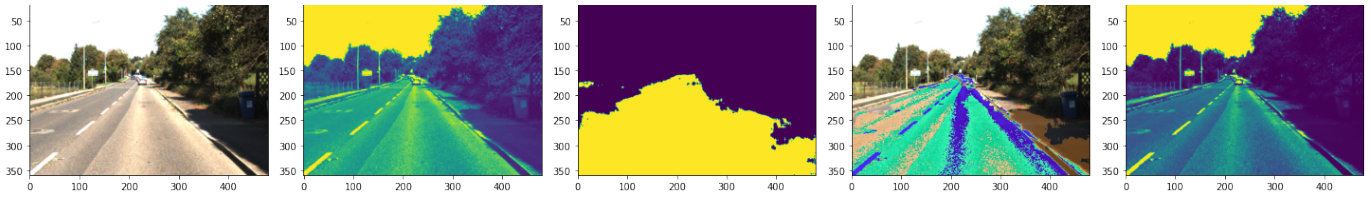
Epoch #45 Loss: 5.181281 Train Acc: 98.107% Valid Acc: 98.373% Time: 85.90s
Epoch #46 Loss: 5.001117 Train Acc: 98.119% Valid Acc: 98.328% Time: 85.89s
Epoch #47 Loss: 4.907640 Train Acc: 98.031% Valid Acc: 98.322% Time: 85.91s
Epoch #48 Loss: 4.909502 Train Acc: 98.278% Valid Acc: 98.338% Time: 85.92s



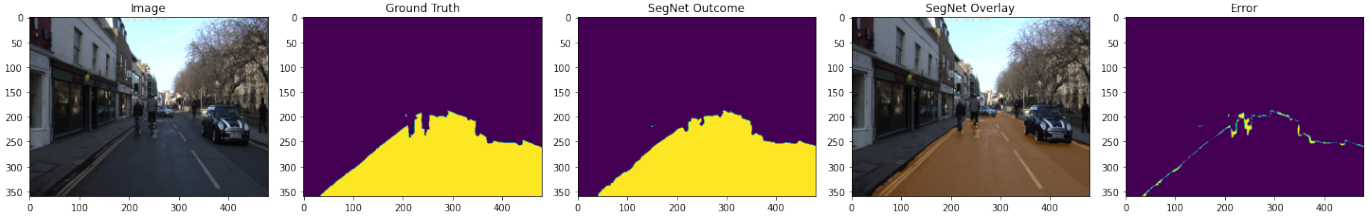


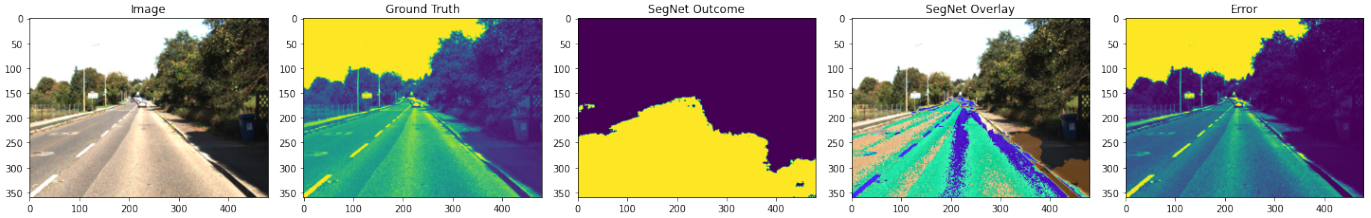
Epoch #49 Loss: 4.819682 Train Acc: 98.211% Valid Acc: 98.444% Time: 85.91s
Epoch #50 Loss: 4.781480 Train Acc: 98.278% Valid Acc: 98.485% Time: 85.94s
Epoch #51 Loss: 4.678722 Train Acc: 98.131% Valid Acc: 98.347% Time: 85.91s
Epoch #52 Loss: 4.673874 Train Acc: 98.192% Valid Acc: 98.366% Time: 85.93s



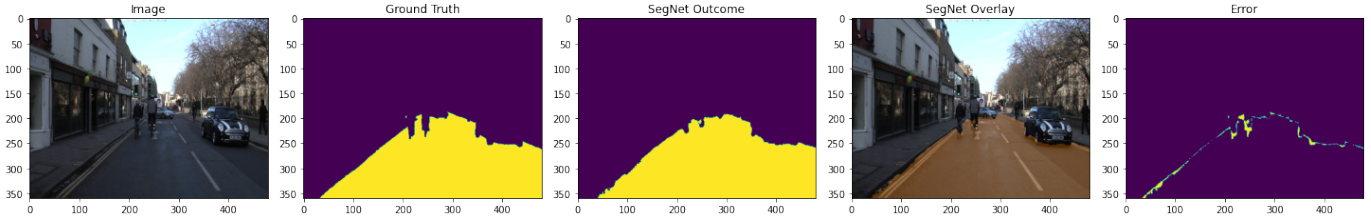


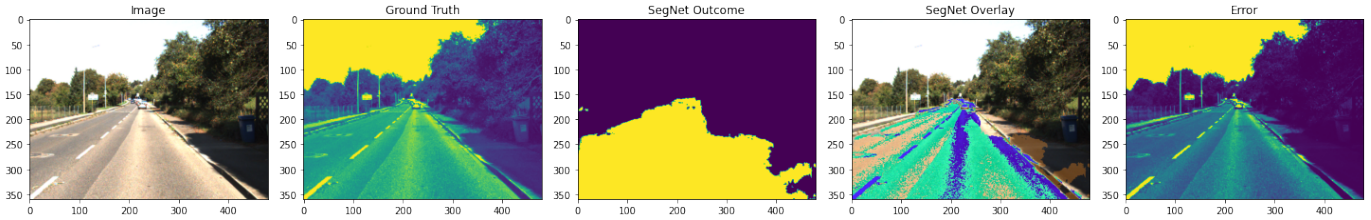
```
Epoch #53       Loss: 4.709645  Train Acc: 98.162%       Valid Acc: 98.424%       Time: 85.92s
Epoch #54       Loss: 4.658604  Train Acc: 98.306%       Valid Acc: 98.524%       Time: 85.92s
Epoch #55       Loss: 4.562550  Train Acc: 98.275%       Valid Acc: 98.400%       Time: 85.94s
Epoch #56       Loss: 4.578988  Train Acc: 98.151%       Valid Acc: 98.392%       Time: 85.90s
```





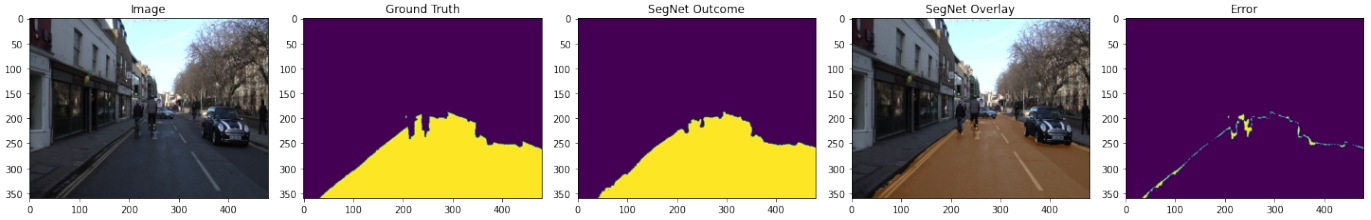

```
Epoch #57       Loss: 4.528505  Train Acc: 98.461%       Valid Acc: 98.482%       Time: 85.92s
Epoch #58       Loss: 4.430164  Train Acc: 98.352%       Valid Acc: 98.381%       Time: 85.92s
Epoch #59       Loss: 4.453888  Train Acc: 98.374%       Valid Acc: 98.417%       Time: 85.91s
<================ Learning Rate 0.003703703703703704 -> 0.0012345679012345679===== ==========>
Epoch #60       Loss: 4.170705  Train Acc: 98.458%       Valid Acc: 98.455%       Time: 85.90s
```

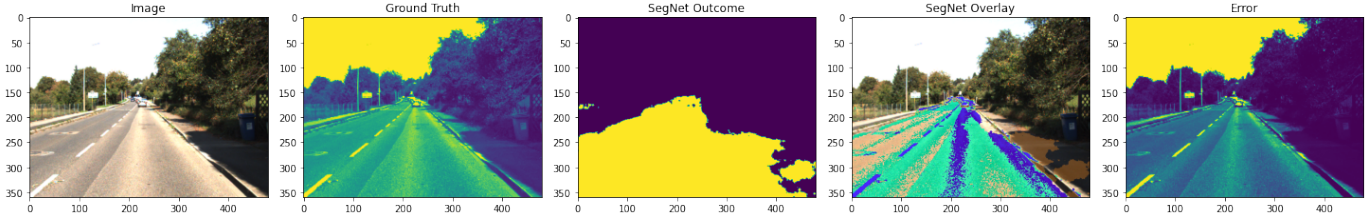




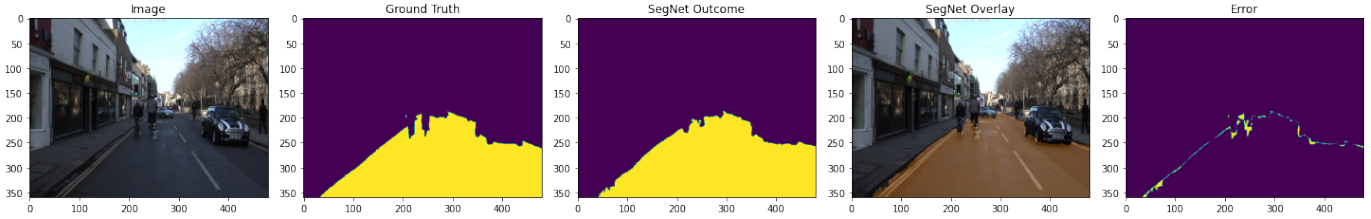

```
Epoch #61       Loss: 4.075523  Train Acc: 98.354%       Valid Acc: 98.390%       Time: 85.89s
Epoch #62       Loss: 4.035420  Train Acc: 98.425%       Valid Acc: 98.395%       Time: 85.90s
Epoch #63       Loss: 4.009435  Train Acc: 98.540%       Valid Acc: 98.466%       Time: 85.89s
```

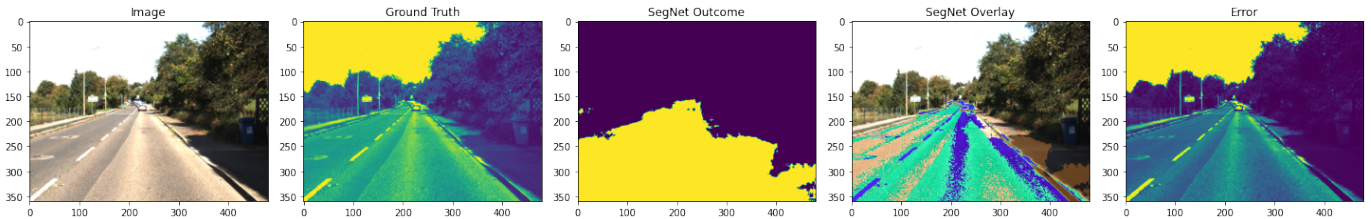
Epoch #64 Loss: 3.994226 Train Acc: 98.425% Valid Acc: 98.444% Time: 85.92s



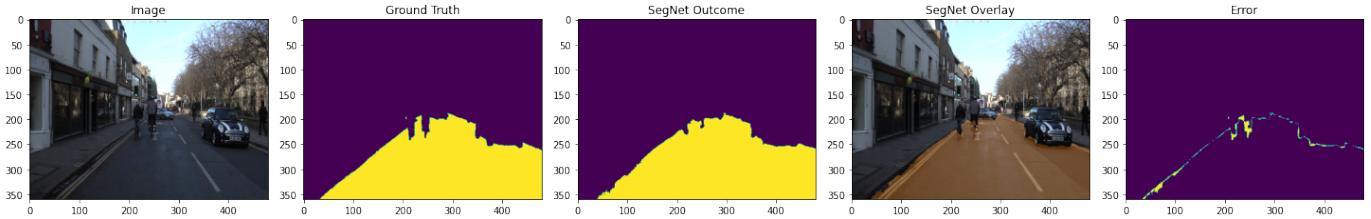


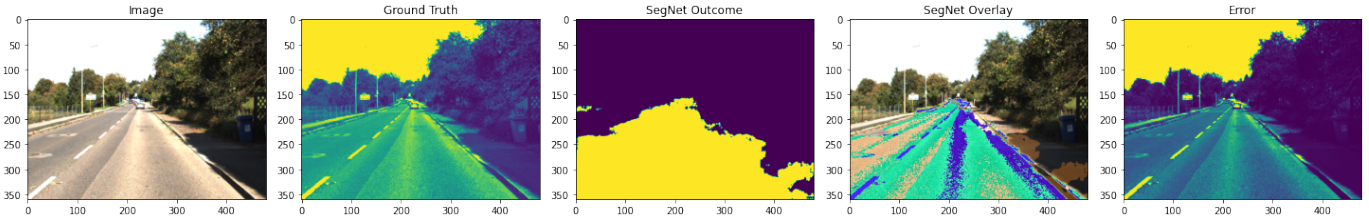
Epoch #65 Loss: 3.996300 Train Acc: 98.452% Valid Acc: 98.417% Time: 85.93s
Epoch #66 Loss: 3.921269 Train Acc: 98.517% Valid Acc: 98.417% Time: 85.88s
Epoch #67 Loss: 3.906095 Train Acc: 98.568% Valid Acc: 98.451% Time: 85.88s
Epoch #68 Loss: 3.910293 Train Acc: 98.504% Valid Acc: 98.450% Time: 85.92s



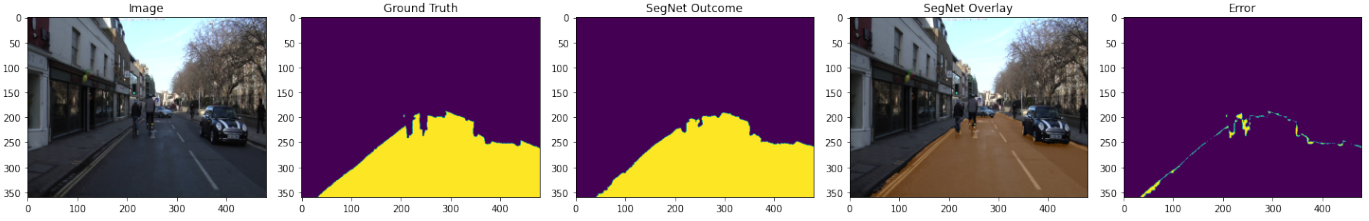


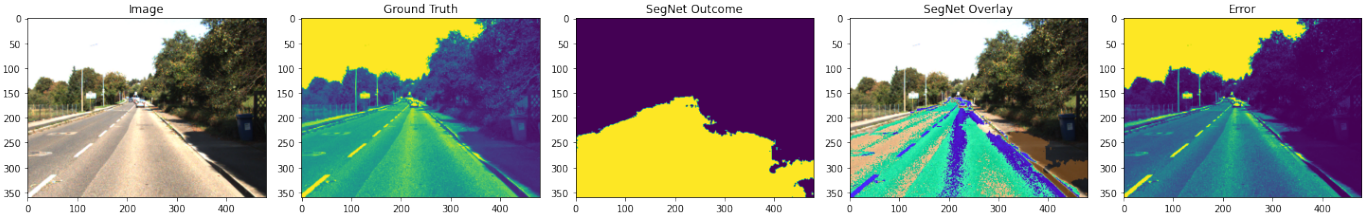
Epoch #69 Loss: 3.928397 Train Acc: 98.541% Valid Acc: 98.486% Time: 85.86s
Epoch #70 Loss: 3.887527 Train Acc: 98.602% Valid Acc: 98.455% Time: 85.90s
Epoch #71 Loss: 3.856665 Train Acc: 98.422% Valid Acc: 98.402% Time: 85.94s
Epoch #72 Loss: 3.861521 Train Acc: 98.540% Valid Acc: 98.466% Time: 85.97s



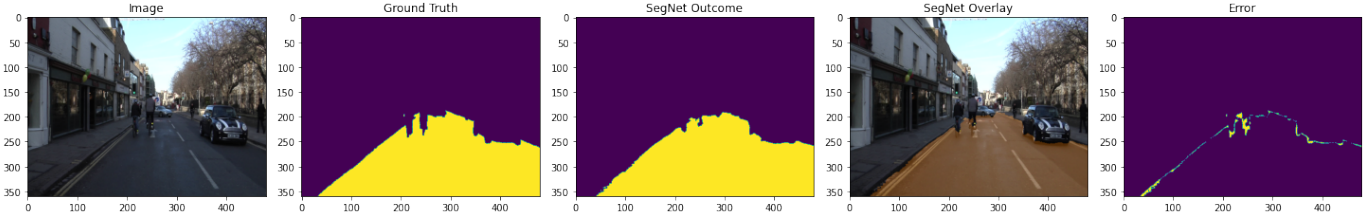


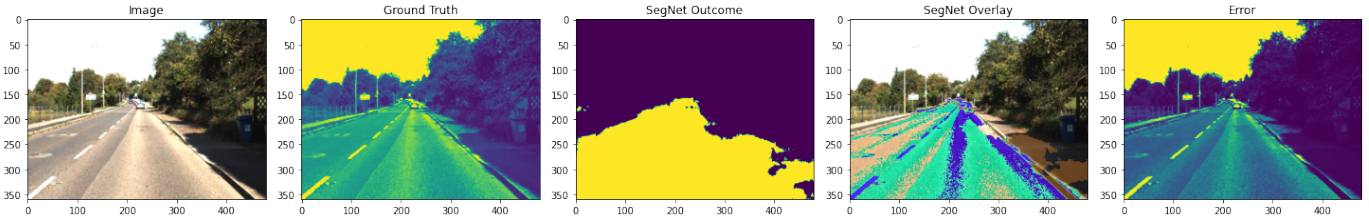

```
Epoch #73       Loss: 3.864235  Train Acc: 98.503%       Valid Acc: 98.438%       Time: 85.90s
Epoch #74       Loss: 3.829767  Train Acc: 98.615%       Valid Acc: 98.429%       Time: 85.94s
<================ Learning Rate 0.0012345679012345679 -> 0.0004115226337448596===== ==========>
Epoch #75       Loss: 3.708163  Train Acc: 98.633%       Valid Acc: 98.481%       Time: 85.92s
Epoch #76       Loss: 3.661913  Train Acc: 98.601%       Valid Acc: 98.477%       Time: 85.92s
```





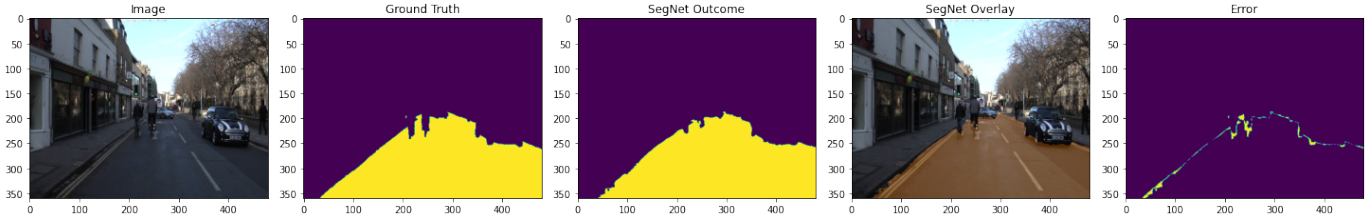

```
Epoch #77       Loss: 3.662009  Train Acc: 98.628%       Valid Acc: 98.486%       Time: 85.87s
Epoch #78       Loss: 3.702063  Train Acc: 98.611%       Valid Acc: 98.430%       Time: 85.92s
Epoch #79       Loss: 3.687805  Train Acc: 98.617%       Valid Acc: 98.470%       Time: 85.95s
Epoch #80       Loss: 3.657182  Train Acc: 98.630%       Valid Acc: 98.487%       Time: 85.91s
```

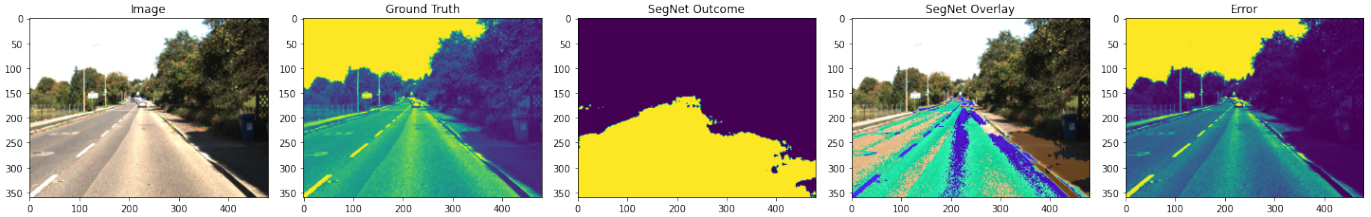




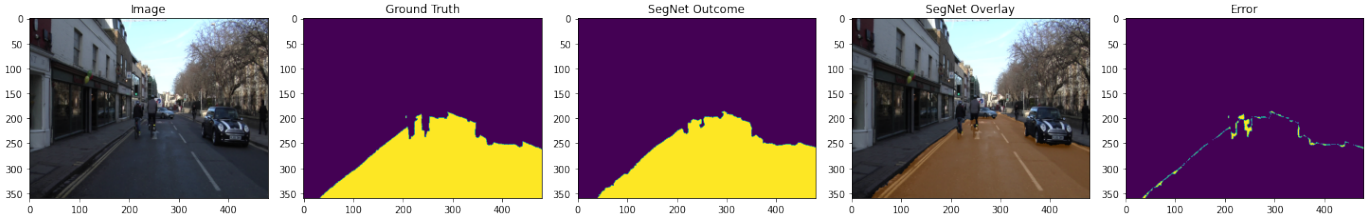

```
Epoch #81       Loss: 3.622637  Train Acc: 98.598%       Valid Acc: 98.471%       Time: 85.94s
Epoch #82       Loss: 3.670632  Train Acc: 98.624%       Valid Acc: 98.467%       Time: 85.90s
Epoch #83       Loss: 3.616052  Train Acc: 98.700%       Valid Acc: 98.489%       Time: 85.95s
```

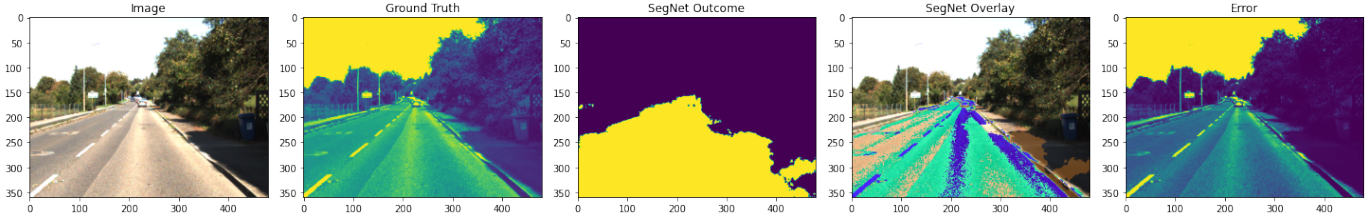
Epoch #84 Loss: 3.607867 Train Acc: 98.606% Valid Acc: 98.435% Time: 85.95s



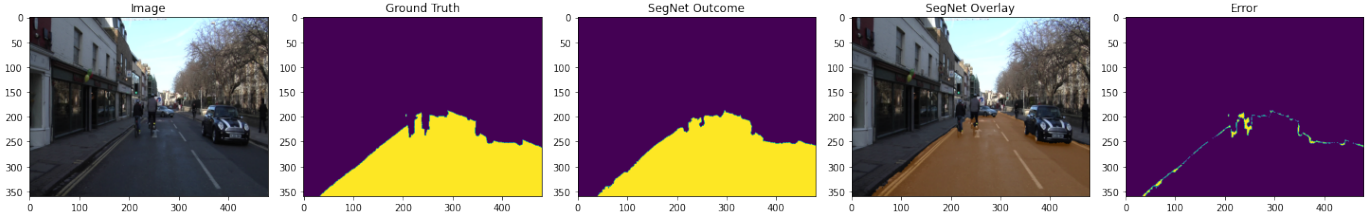


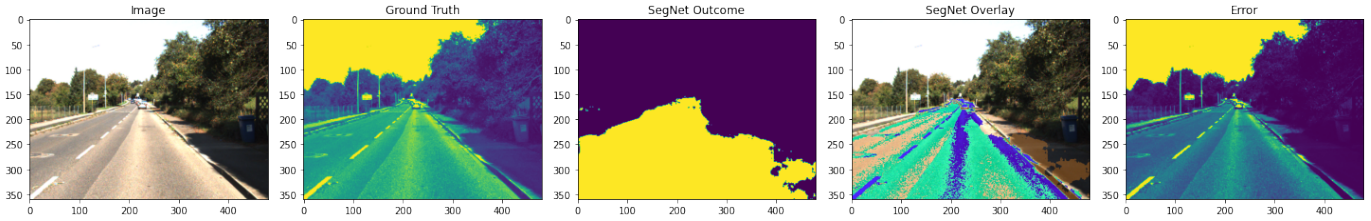
Epoch #85 Loss: 3.596084 Train Acc: 98.668% Valid Acc: 98.474% Time: 85.89s
Epoch #86 Loss: 3.595608 Train Acc: 98.643% Valid Acc: 98.499% Time: 85.92s
Epoch #87 Loss: 3.587838 Train Acc: 98.640% Valid Acc: 98.464% Time: 85.93s
Epoch #88 Loss: 3.590524 Train Acc: 98.602% Valid Acc: 98.429% Time: 85.91s



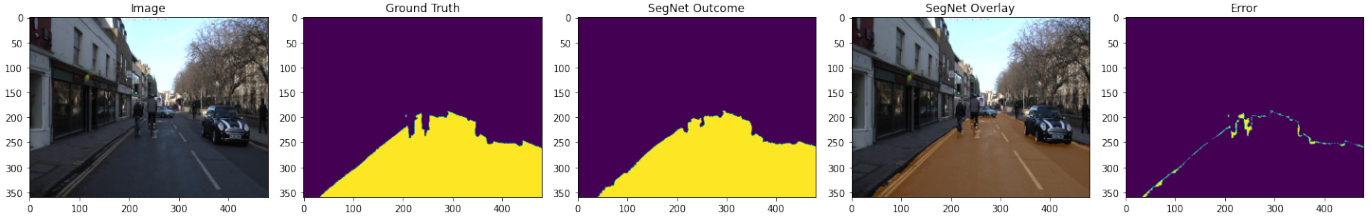


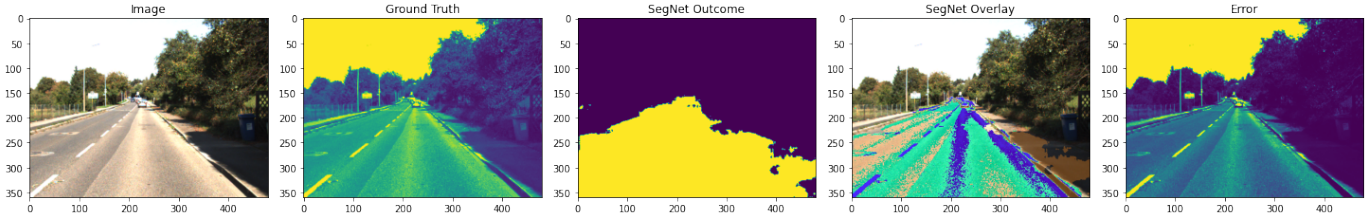
Epoch #89 Loss: 3.577438 Train Acc: 98.649% Valid Acc: 98.477% Time: 85.90s
<=============== Learning Rate 0.00041152263374485596 -> 0.00013717421124828533===== ==========>
Epoch #90 Loss: 3.551960 Train Acc: 98.664% Valid Acc: 98.481% Time: 85.96s
Epoch #91 Loss: 3.556124 Train Acc: 98.668% Valid Acc: 98.473% Time: 85.94s
Epoch #92 Loss: 3.526460 Train Acc: 98.683% Valid Acc: 98.502% Time: 85.97s



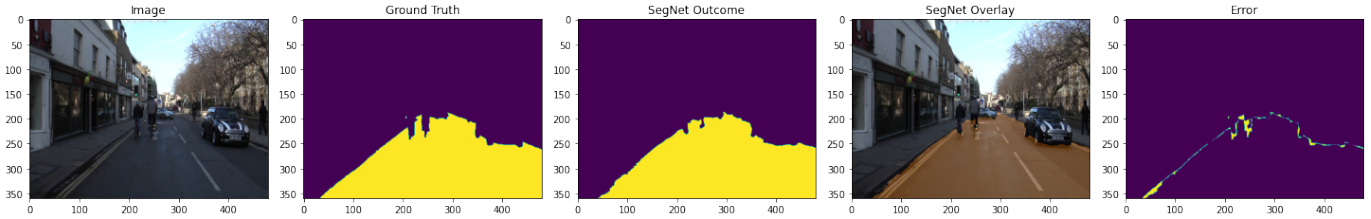


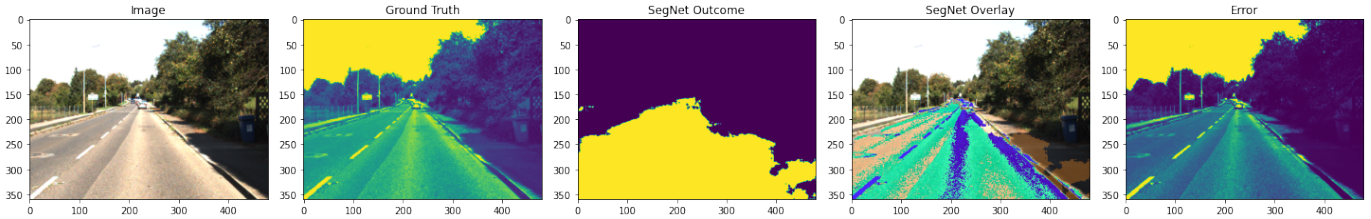

```
Epoch #93       Loss: 3.511168  Train Acc: 98.649%       Valid Acc: 98.459%       Time: 85.95s
Epoch #94       Loss: 3.514572  Train Acc: 98.677%       Valid Acc: 98.482%       Time: 85.94s
Epoch #95       Loss: 3.554174  Train Acc: 98.650%       Valid Acc: 98.459%       Time: 85.93s
Epoch #96       Loss: 3.496509  Train Acc: 98.650%       Valid Acc: 98.469%       Time: 85.92s
```






```
Epoch #97       Loss: 3.563272  Train Acc: 98.641%       Valid Acc: 98.486%       Time: 85.90s
Epoch #98       Loss: 3.541467  Train Acc: 98.649%       Valid Acc: 98.482%       Time: 85.95s
Epoch #99       Loss: 3.499984  Train Acc: 98.671%       Valid Acc: 98.464%       Time: 85.94s
Epoch #100      Loss: 3.521144  Train Acc: 98.666%       Valid Acc: 98.473%       Time: 85.96s
```






```
Epoch #101      Loss: 3.525872  Train Acc: 98.650%       Valid Acc: 98.460%       Time: 85.93s
```

```
KeyboardInterrupt                         Traceback (most recent call last)
```

```
<ipython-input-11-dcbe98f56ddd> in <module>()
      8
      9 # Launch training
---> 10 train(model, optimizer, loss, X_train, y_train, X_valid, y_valid)

<ipython-input-10-1292fd992415> in train(model, optimizer, loss_fn, X_train, y_train, X_valid, y_valid
)
     73
     74     # train_acc = avg_pixelwise_accuracy(model, train_dataset)
---> 75     train_acc = avg_pixelwise_accuracy(model, train_dataset) * 100
     76     valid_acc = avg_pixelwise_accuracy(model, valid_dataset) * 100
     77     print("Epoch #{}\tLoss: {:.6f}\tTrain Acc: {:.3f}%\tValid Acc: {:.3f}%\tTime: {:.2f}s".for
mat(epoch+1, epoch_loss, train_acc, valid_acc, delta))

<ipython-input-5-967846405a9e> in avg_pixelwise_accuracy(model, dataset)
     13
     14   correct, total = 0, 0
---> 15   for i, batch in enumerate(loader):
     16
     17     # mount to GPU if available [Need to fix y]

/usr/local/lib/python3.6/dist-packages/torch/utils/data/dataloader.py in __next__(self)
    433         if self._sampler_iter is None:
    434             self._reset()
--> 435         data = self._next_data()
    436         self._num_yielded += 1
    437         if self._dataset_kind == _DatasetKind.Iterable and \

/usr/local/lib/python3.6/dist-packages/torch/utils/data/dataloader.py in _next_data(self)
    473     def _next_data(self):
    474         index = self._next_index()  # may raise StopIteration
--> 475         data = self._dataset_fetcher.fetch(index)  # may raise StopIteration
    476         if self._pin_memory:
    477             data = _utils.pin_memory.pin_memory(data)

/usr/local/lib/python3.6/dist-packages/torch/utils/data/_utils/fetch.py in fetch(self, possibly_batche
d_index)
     45         else:
     46             data = self.dataset[possibly_batched_index]
---> 47         return self.collate_fn(data)

/usr/local/lib/python3.6/dist-packages/torch/utils/data/_utils/collate.py in default_collate(batch)
     71         return batch
     72     elif isinstance(elem, container_abcs.Mapping):
---> 73         return {key: default_collate([d[key] for d in batch]) for key in elem}
     74     elif isinstance(elem, tuple) and hasattr(elem, '_fields'):  # namedtuple
     75         return elem_type(*(default_collate(samples) for samples in zip(*batch)))

/usr/local/lib/python3.6/dist-packages/torch/utils/data/_utils/collate.py in <dictcomp>(.0)
     71         return batch
     72     elif isinstance(elem, container_abcs.Mapping):
---> 73         return {key: default_collate([d[key] for d in batch]) for key in elem}
     74     elif isinstance(elem, tuple) and hasattr(elem, '_fields'):  # namedtuple
     75         return elem_type(*(default_collate(samples) for samples in zip(*batch)))

/usr/local/lib/python3.6/dist-packages/torch/utils/data/_utils/collate.py in default_collate(batch)
     53             storage = elem.storage()._new_shared(numel)
     54             out = elem.new(storage)
```

```
---> 55         return torch.stack(batch, 0, out=out)
     56     elif elem_type.__module__ == 'numpy' and elem_type.__name__ != 'str_' \
     57             and elem_type.__name__ != 'string_':

KeyboardInterrupt:
```

Train from VGG weights

In [ ]:

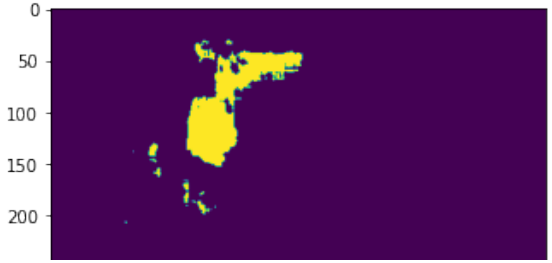
```
TRANSFER_LEARNING = True
model_vgginit = SegNet(3, 2, transfer_learning=TRANSFER_LEARNING).to(DEVICE)
optimizer = torch.optim.Adam(model_vgginit.parameters(), lr=LEARNING_RATE)
loss = nn.CrossEntropyLoss(weight=LABEL_WEIGHTS).to(DEVICE)

# Launch training
torch.cuda.empty_cache()
train(model_vgginit, optimizer, loss, X_train, y_train, X_valid, y_valid)
```

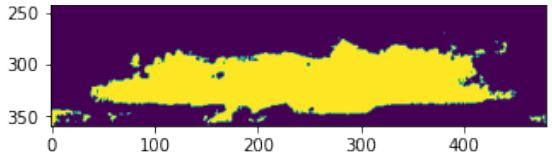
Load model

In [9]:

```
# model_name = r'Epoch101_loss3.5258_trainacc98.650_valacc98.460.pth' # augmented
# model_name = r'../Epoch49_loss2.1120_trainacc97.727_valacc97.859.pth' # scratch
model_name = r'../Epoch49_loss2.1120_trainacc97.727_valacc97.859.pth' # scratch
model_load = load_checkpoints(model_name)

show_pred_mask(model_load, X_train[0, :])
model = model_load
```

```
Model Loaded: Epoch#49  Loss:2.1120      TrainAcc:97.7268         ValidAcc:97.8592
```





Report Evaluation results on test dataset

In [10]:

```
torch.cuda.empty_cache()

model_selected = model
test_dataset = data.CamVidDataset(X_test, y_test)
test_accuracy = avg_pixelwise_accuracy(model_selected, test_dataset)
print("Selected model's pixelwise accuracy on test dataset : {:.5f}%".format(test_accuracy * 100))
```

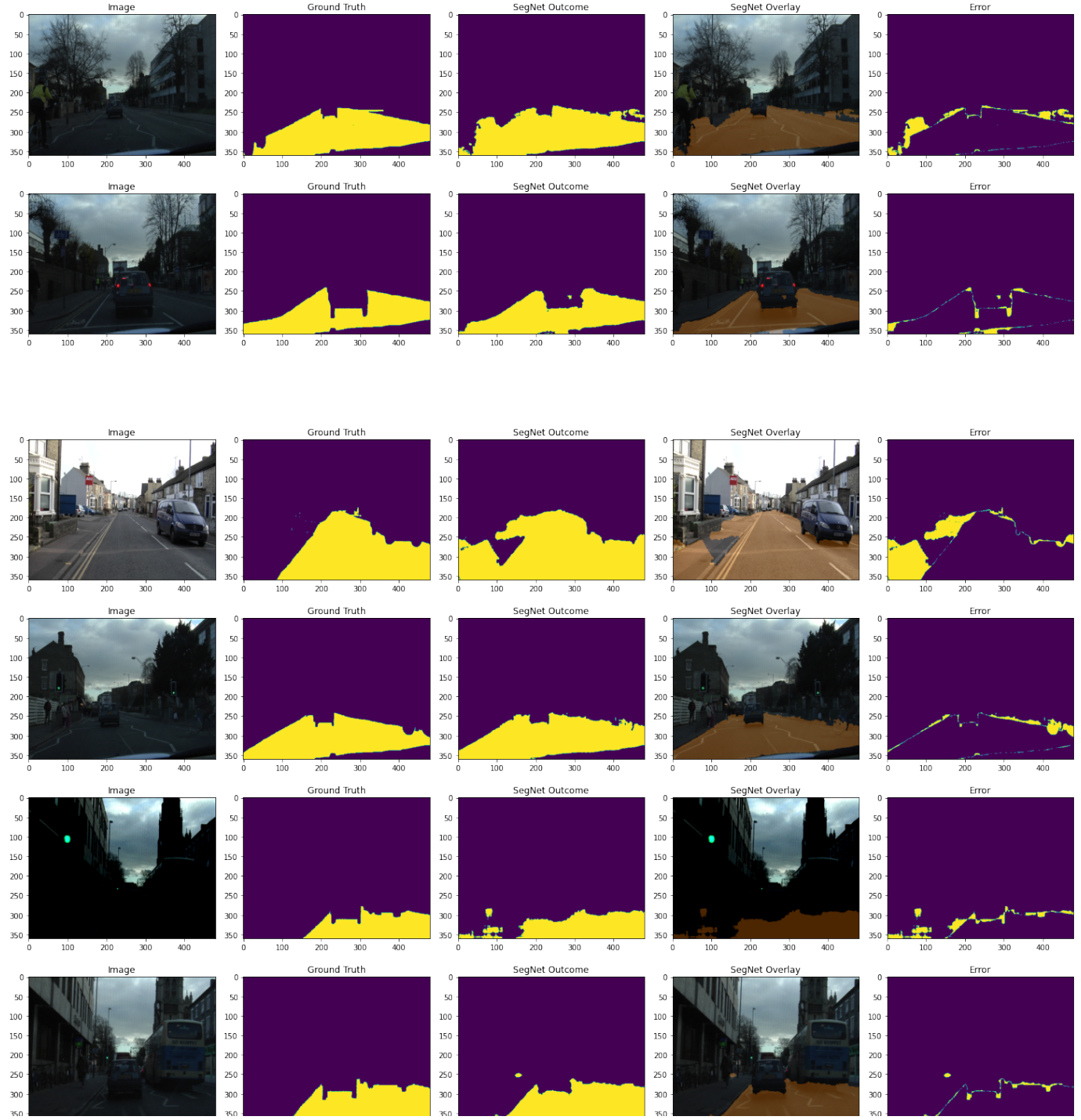
```
Selected model's pixelwise accuracy on test dataset : 90.24887%
```

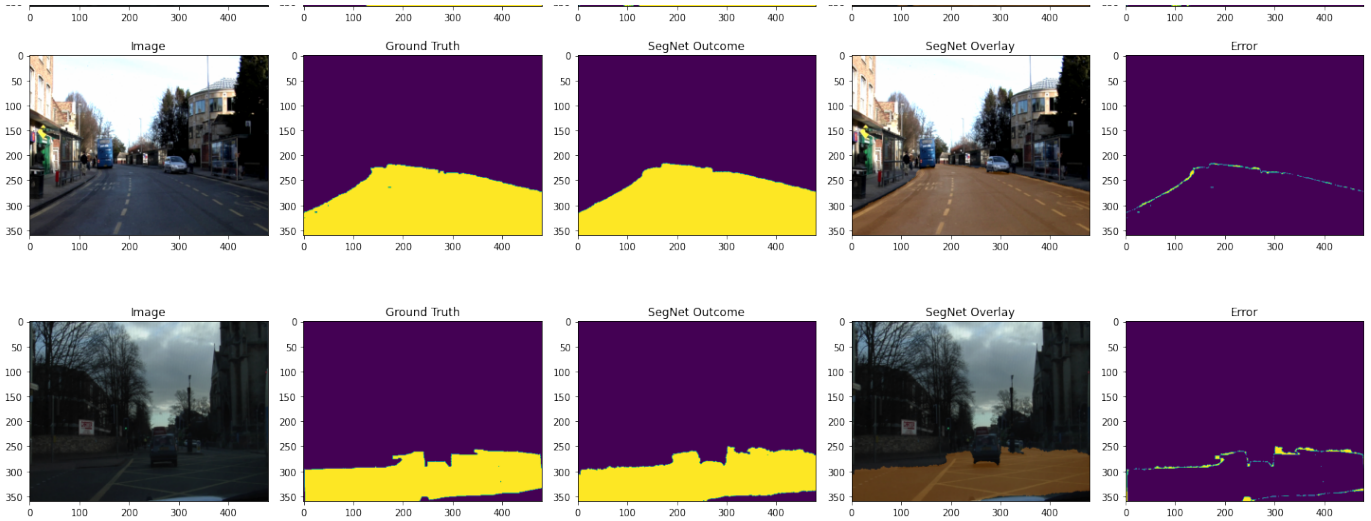
Visualization Output

In [15]:

```
# visualize a single input image
# show_img(X_test[0, :])
# show_mask(y_test[0, 1, :])
# show_pred_mask(model_load, X_test[0, :])


# cmap = 'prism'
# cmap = 'brg'
# cmap = 'Paired'
# cmap = 'jet'

model_selected = model
show_all(model_selected, X_test[2, :], y_test[2, :], cmap='jet')
show_all(model_selected, X_test[11, :], y_test[11, :], cmap='jet')
show_all(model_selected, X_test[-2, :], y_test[-2, :], cmap='jet')
show_all(model_selected, X_test[43, :], y_test[43, :], cmap='jet')

show_all(model_selected, X_train[2, :], y_train[2, :], cmap='jet')
show_all(model_selected, X_train[11, :], y_train[11, :], cmap='jet')
show_all(model_selected, X_train[-2, :], y_train[-2, :], cmap='jet')
```

```
show_all(model_selected, x_train[43, :], y_train[43, :], cmap='jet')
```



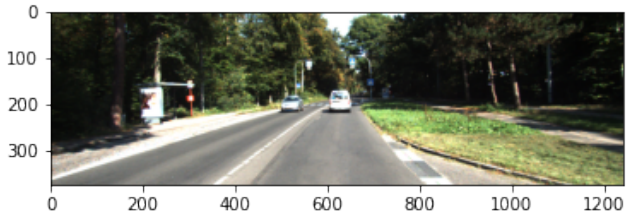


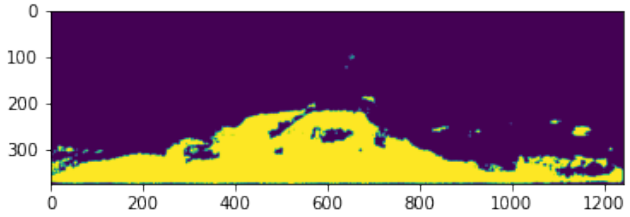
Demostration

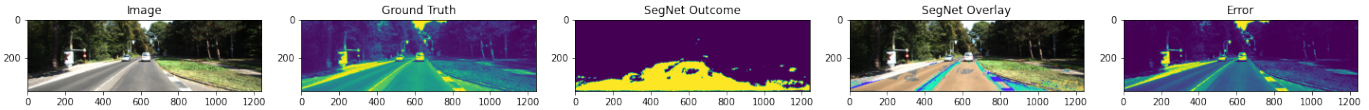
On Demo.py

In [16]:

```
model.eval()

kitti_img = cv2.imread("/content/gdrive/MyDrive/Segmentation/KiTTi/um_000007.png")

kitti_tensor = torch.from_numpy(np.moveaxis(kitti_img, -1, 0))

show_img(kitti_tensor)

show_pred_mask(model, kitti_tensor.float())

show_all(model, kitti_tensor.float(), kitti_tensor.float()[0, :])
```







Overlay Segmentation results

In [18]:

```
def write_to_dir(model, load_dir, save_dir):

  # change to eval mode
  model.eval()

  # create output dir
  os.makedirs(save_dir, exist_ok=True)

  # loop over the given directory
  for item in sorted(os.listdir(load_dir)):
```

```
  for item in sorted(os.listdir(load_dir)):
    if '.png' in item:
      img = cv2.imread(os.path.join(load_dir, item))
      img = cv2.resize(img, (480, 320))
      img = np.moveaxis(img, -1, 0)
      img = torch.from_numpy(img).type(torch.float32)
      out = model(img.unsqueeze(0).to(DEVICE))
      predicted = torch.argmax(out, dim=1)
      predicted = predicted.cpu().data.numpy()[0, :]

      img = torch.movedim(img, 0, -1).type(torch.uint8).numpy().astype(np.float)
      img[:, :, 1] += (predicted * 255 * 0.5).astype(np.uint8)
      img = np.clip(img, 0, 255).astype(np.uint8)
      # img[:, :, 0] += (predicted * 255 * 0.7).astype(np.uint8)

      cv2.imwrite(os.path.join(save_dir, item), img)
```

In [19]:

```
# model_name = r'Epoch20_loss1.5263_trainacc97.227_valacc97.571.pth'
model_load = model


load_dir = r'/content/gdrive/MyDrive/KiTTi_dataset/testing_crop'
save_dir = r'/content/gdrive/MyDrive/KiTTi_dataset/testing_crop_segnet_new'
write_to_dir(model_load, load_dir, save_dir)
```

In [21]:

```
load_dir = r'/content/gdrive/MyDrive/KiTTi_dataset/training_crop'
save_dir = r'/content/gdrive/MyDrive/KiTTi_dataset/training_crop_segnet_new'
write_to_dir(model_load, load_dir, save_dir)
```

In [22]:

```
load_dir = camvid_path+"/train"
save_dir = camvid_path+"/train_segnet_new"
write_to_dir(model_load, load_dir, save_dir)

load_dir = camvid_path+"/test"
save_dir = camvid_path+"/test_segnet_new"
write_to_dir(model_load, load_dir, save_dir)

load_dir = camvid_path+"/val"
save_dir = camvid_path+"/val_segnet_new"
write_to_dir(model_load, load_dir, save_dir)
```