ELEC564 – Spring 2023
Homework 5

Date: April 7th, 2023

Student's Name: Hsuan-You (Shaun) Lin / Net ID: hl116

Link to Colab notebook:
https://colab.research.google.com/drive/1-JNs185SKvZdWoCXZnoMAm-1rWygAKpd?usp=sharing

Problem 1: Semantic Segmentation (7 points)

In this problem, you will train a simple semantic segmentation network. Recall that in semantic segmentation, the algorithm must assign each pixel of an input image to one of *K* object classes. We have provided you with a Colab notebook with skeleton code to get you started.

We will use a portion of the CityScapes dataset for this problem, consisting of 2975 training images and 500 validation images. The second cell in the notebook will automatically download the dataset into your local Colab environment.

Each image also comes with annotations for 34 object classes in the form of a segmentation image (with suffix 'labelIds.png'). The segmentation image contains integer ids in [0, 33] indicating the class of each pixel. This page provides the mappings from id to label name.

a. Fill in the `init` and `forward` functions for the `Segmenter` class, which will implement your segmentation network. The network will be a convolutional encoder-decoder. The encoder will consist of the first several 'blocks' of layers extracted from the VGG16 network pretrained on ImageNet (the provided Colab notebook extracts these layers for you). You must implement the decoder with this form:

Layer	**Output channels for Conv**
3 x 3 Conv + ReLU	64
Upsample (2 x 2)	X
3 x 3 Conv + ReLU	64
Upsample (2 x 2)	X
3 x 3 Conv + ReLU	64

Upsample (2 x 2)	X
3 x 3 Conv	n_classes (input to init)

Use PyTorch's [Upsample function](). Remember that the size of the image should not change after each Conv operation (add appropriate padding).

b. Train your model for 7 epochs using the nn.CrossEntropy loss function. Using the GPU, this should take about 30 minutes.

```
-------------------------------------------------------------

/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default
  warnings.warn(
Epoch [5/7], Train Loss: 0.6060, Val Loss: 0.6198
Saved 4 epoch  model to " /content/drive/MyDrive/Colab Notebooks/COMP546/HW/HW5/models/model_4.params "

-------------------------------------------------------------

/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default
  warnings.warn(
Epoch [6/7], Train Loss: 0.5794, Val Loss: 0.6094
Saved 5 epoch  model to " /content/drive/MyDrive/Colab Notebooks/COMP546/HW/HW5/models/model_5.params "

-------------------------------------------------------------

/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default
  warnings.warn(
Epoch [7/7], Train Loss: 0.5674, Val Loss: 0.5996
Saved 6 epoch  model to " /content/drive/MyDrive/Colab Notebooks/COMP546/HW/HW5/models/model_6.params "

-------------------------------------------------------------
```

Figure 1. Training the model for 7 epochs

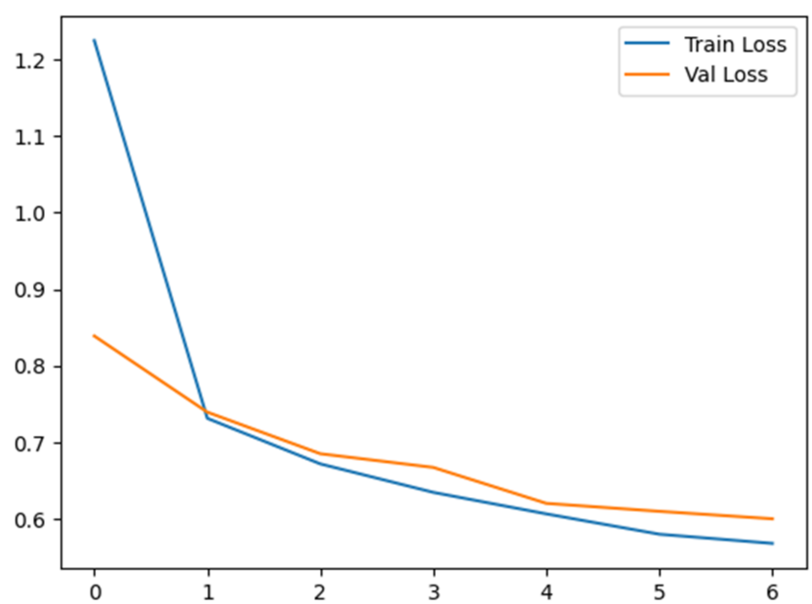

Figure 2. Training results for 7 epochs (Train Loss & Validation Loss)

```
/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default valu
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default valu
  warnings.warn(
Epoch [12/14], Train Loss: 0.4740, Val Loss: 0.5343
Saved 11 epoch  model to " /content/drive/MyDrive/Colab Notebooks/COMP546/HW/HW5/models/new_model_11.params "

-------------------------------------------------------------

/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default valu
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default valu
  warnings.warn(
Epoch [13/14], Train Loss: 0.4679, Val Loss: 0.5299
Saved 12 epoch  model to " /content/drive/MyDrive/Colab Notebooks/COMP546/HW/HW5/models/new_model_12.params "

-------------------------------------------------------------

/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default valu
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default valu
  warnings.warn(
Epoch [14/14], Train Loss: 0.4650, Val Loss: 0.5202
Saved 13 epoch  model to " /content/drive/MyDrive/Colab Notebooks/COMP546/HW/HW5/models/new_model_13.params "

-------------------------------------------------------------
```

Figure 3. Training the model for 14 epochs

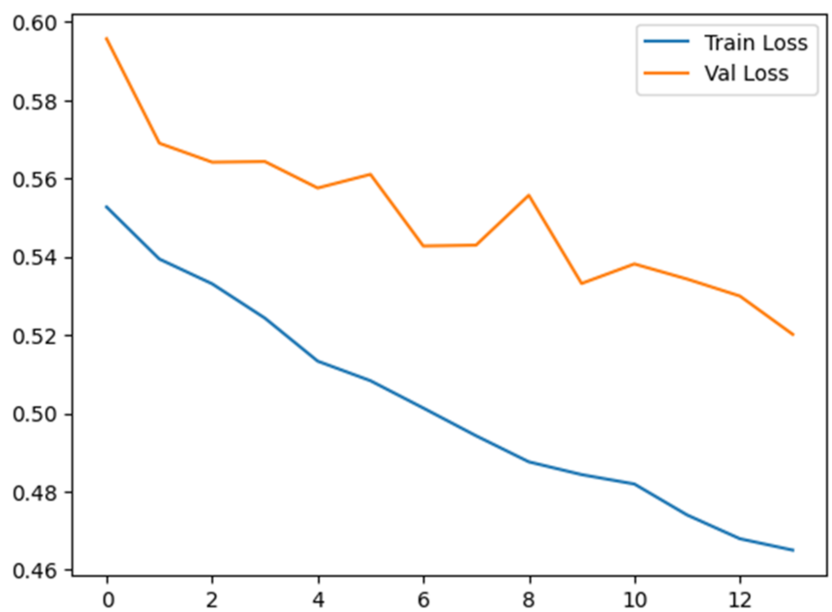

Figure 4. Training results for 14 epochs (Train Loss & Validation Loss)

c. Using the final model, report the [intersection-over-union]() (IoU) per class on the validation set in a table. For more on IoU, see [this page](). Which class has the best IoU, and which has the worst? Comment on why you think certain classes have better accuracies than others, and what factors may cause those differences.

```
/usr/local/lib/python3.9/dist-pac
  warnings.warn(
IoU per class:
unlabeled: 0.8645
ego vehicle: 0.7691
rectification border: 0.8069
out of roi: 0.0011
static: 0.0000
dynamic: 0.0000
ground: 0.9180
road: 0.5526
sidewalk: 0.0126
parking: 0.0000
rail track: 0.7105
building: 0.1099
wall: 0.1699
fence: nan
guard rail: nan
bridge: nan
tunnel: 0.0479
pole: 0.0000
polegroup: 0.0000
traffic light: 0.1672
traffic sign: 0.7645
vegetation: 0.4574
terrain: 0.7962
sky: 0.2498
person: 0.0000
rider: 0.6633
car: 0.0002
truck: 0.0000
bus: 0.0000
caravan: 0.0000
trailer: nan
train: 0.0000
motorcycle: 0.3224
bicycle: 0.8600
```

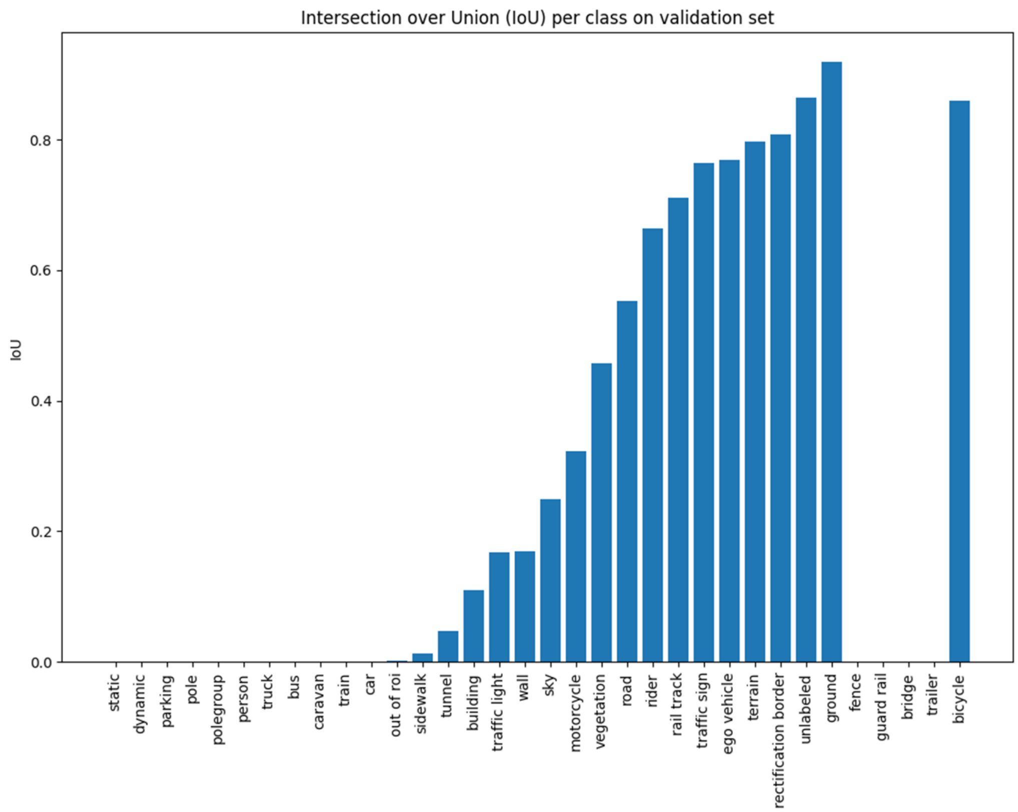

Figure 5. IoU per class on the validation set

My Answer:

The class with the best IoU is "ground" with a value of 0.9180, while the classes with the worst IoU are "fence", "guard rail", "bridge", "trailer", "person", "car", "truck", "bus", "caravan", and "train" with a value of "nan", which means they were ignored during the calculation because they were unlabeled.

The "ground" class has a high IoU as it is relatively easy to distinguish from other classes, has a large amount of labeled training data, and appears frequently in the scene. On the other hand, classes such as "fence", "guard rail", and "bridge" may be harder to accurately distinguish due to their visual similarity to other classes or their rarity in the scene.

d. For each of the following validation images, show three images side-by-side: the image, the ground truth segmentation, and your predicted segmentation. The segmentation images should be in color, with each class represented by a different color.
    i. frankfurt_000000_015389_leftImg8bit.jpg
    ii. frankfurt_000001_057954_leftImg8bit.jpg
    iii. lindau_000037_000019_leftImg8bit.jpg
    iv. munster_000173_000019_leftImg8bit.jpg

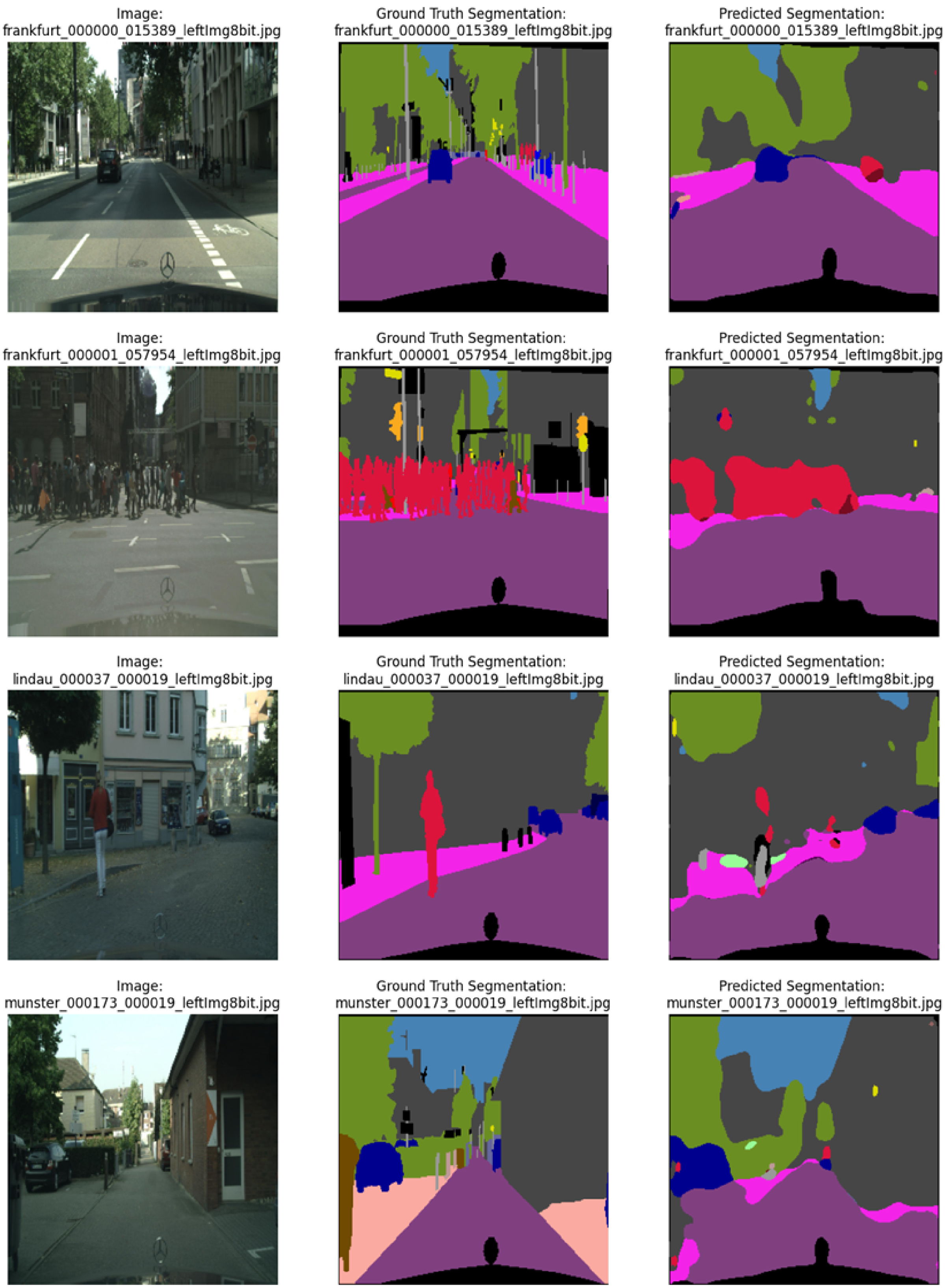

Figure 6. Original images, Ground truth segmentation and predicted segmentation results for 7 epochs model

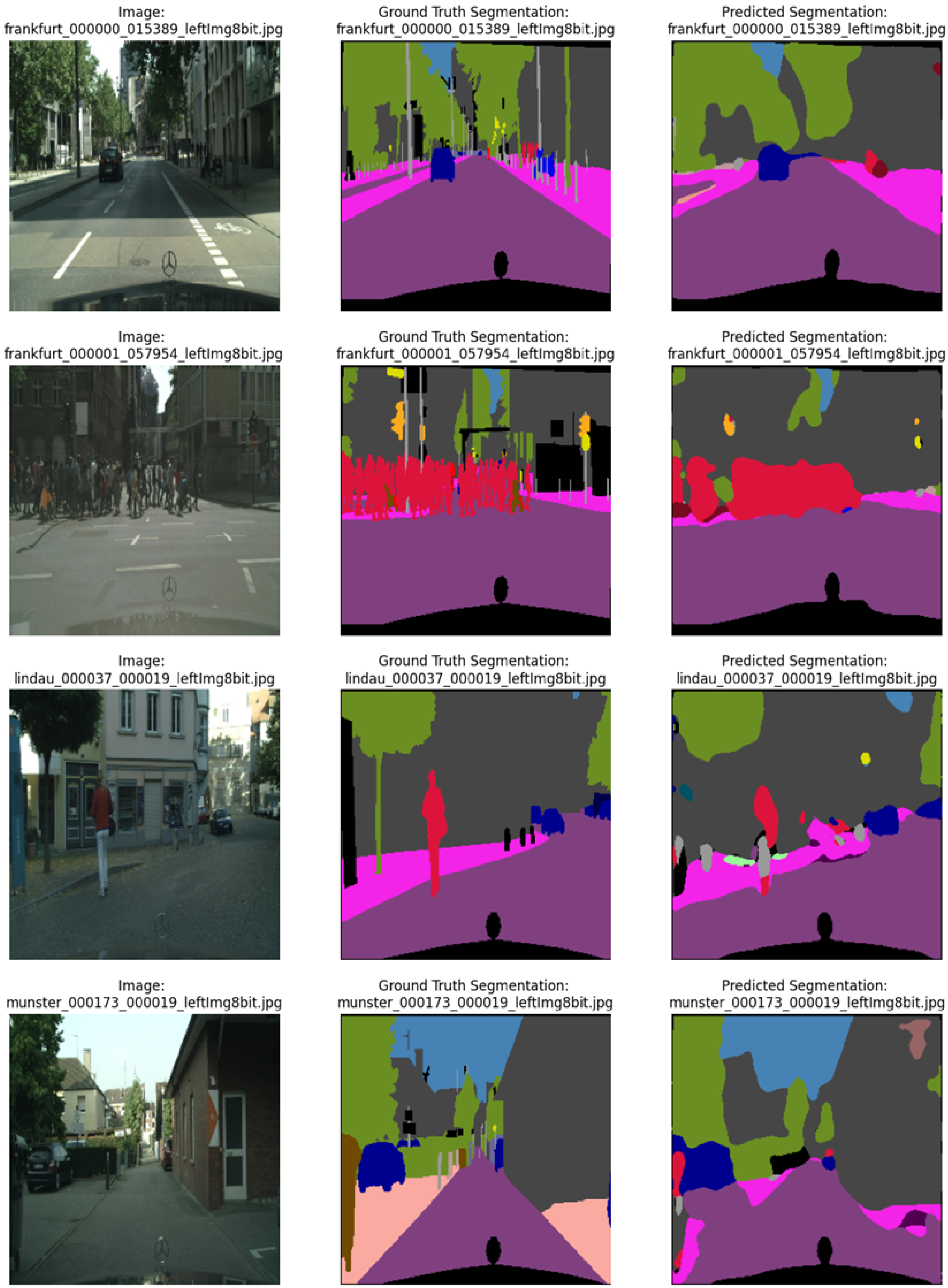

Figure 7. Original images, Ground truth segmentation and predicted segmentation results for 14 epochs model

My Observation:

After training the model for 7 and 14 epochs, I found that "lindau_000037_000019_leftImg8bit.jpg" performed poorly. Its ground truth segmentation image indicated that the entire person should be segmented, but my model failed to do so. Similarly, the image "munster_000173_000019_leftImg8bit.jpg" was unable to accurately segment the road.

e. Look at the lines of code for resizing the images and masks to 256 x 256. We use bilinear interpolation when resizing the image, but nearest neighbor interpolation when resizing the mask. Why do we not use bilinear interpolation for the mask?

My Answer:

We do not use bilinear interpolation for the mask when resizing it to 256 x 256 because the segmentation mask is a categorical map where each pixel is assigned a discrete label indicating the class to which it belongs. If we use bilinear interpolation to resize the mask, it would introduce new values that are not valid class labels and distort the mask, leading to incorrect labels in the output. Nearest neighbor interpolation preserves the integer labels in the mask and is therefore a better choice for resizing categorical maps.

f. Look at the `__getitem__` function for the CityScapesDataset class and notice that we apply a horizontal flip augmentation to the image and mask using a random number generator. Why do we apply the flip in this way instead of simply adding `T.RandomHorizontalFlip` to the sequence of transforms in `im_transform` and `mask_transform` (similar to what you did in Homework 4)

My Answer:

In this project, we apply the horizontal flip augmentation using a random number generator in the getitem function because we want to apply the same random horizontal flip to both the image and mask. If we simply added `T.RandomHorizontalFlip` to the sequence of transforms in `im_transform` and `mask_transform`, we would not be guaranteed that the same flip would be applied to both the image and mask. By applying the flip separately using the random number generator, we ensure that the same flip is applied to both.