Semantic Segmentation (Driver Assistant System)

Robotics and Computer Vision: ***Final Project Report***

Team members: Ashwin Gokhale, Fawaz Tahir, Yara Hanafi

Introduction

A driver assistance system is a technology that helps a vehicle's driver while driving. These systems use a combination of sensors, cameras, and algorithms to provide the driver with information and assist with various tasks. Examples of driver assistance systems include lane departure warning, blind spot monitoring, and adaptive cruise control. These systems are designed to improve safety and make driving more convenient for the driver. We can use Semantic Segmentation to accomplish the above task. Semantic segmentation is a type of image processing technique that involves dividing an image into multiple segments, or regions, each of which corresponds to a different object or class in the image. For example, in a driver assistance system, semantic segmentation could be used to identify different types of objects in the -+vehicle's field of view, such as other vehicles, pedestrians, traffic signs, and road markings.

To use semantic segmentation to accomplish the task of providing driver assistance, the system would first need to be trained on a dataset of images that contain the types of objects and classes relevant to the task. This include images of roads and intersections, as well as images of other vehicles, pedestrians, traffic signs, and road markings. The system would then use this training data to learn to identify and classify these objects and classes in new images. For this we implemented the CamVid dataset annotated by Cambridge."The Cambridge-driving Labeled Video Database (CamVid) is the first collection of videos with object class semantic labels, complete with metadata. The database provides ground truth labels that associate each pixel with one of 32 semantic classes."

Once the system has been trained, it can be used in real-time to process the video feed from the vehicle's cameras. As the vehicle moves, the system will analyze the video feed and identify the different objects and classes in the scene. It can then provide the driver with information about these objects, such as their location and type, and use this information to assist with various tasks. For example, the system could alert the driver if another vehicle is in their blind spot, or if they are approaching a traffic sign that requires them to slow down or stop.

Methods

In order to complete the semantic segmentation, we started with importing all the necessary modules. Some of the primary libraries we used were numpy, torch, torchvision, and trainer. The next step was to create a dataset. After doing some research, we discovered that the CamVid Dataset would be perfect for our project. We started with creating a dataset class. We structured the dataset class using get_num_classes, get_class_name, and get_item. We also needed to initialize masks, transforms, class names, and then the actual path to the dataset. This was primarily given by the CamVid dataset.

After getting the images and masks from the dataset, we wanted to describe the performance of this semantic segmentation model. The metric we wanted to look at was Intersection over Union. This metric helps give the similarity between the predicted segmented region and the ground-truth region. It is defined as the size of the intersection divided by the union of two regions.

To begin this process, we had to first create a confusion matrix class. We used this class to create a confusion matrix when calculating the Intersection over Union metric. We started by creating the normalized confusion matrix:

```
       self.conf_matrix = ConfusionMatrix(num_classes=num_classes,
normalized=normalized)
```

Then, we checked whether or not the ignored classes exist since that would result in a ValueError. From, there we could actually get the metric value we wanted. To do this, we get the value of the confusion matrix using the class we just created. We check to see if the list of indices to ignore is empty and then move on to get the true positive, false positive, and false negative values. We can calculate Intersection Over Union using these values with this formula:

```
           iou = true_positive / (true_positive + false_positive +
false_negative)
```

After the Intersection over Union class is created, we can then actually calculate the metric using the CamVid Dataset we initialized earlier.

```
metric = IntersectionOverUnion(num_classes=test_dataset.get_num_classes(),
reduced_probs=True)
# add samples from test dataset and update metrics value
for sample in tqdm(test_dataset, ascii=False):
   masks = sample["mask"]
   metric.update_value(masks, masks)
```

```
# get the mean iou value
values = metric.get_metric_value()
print(values['mean_iou'])
print(values['iou'])
```

This helps us get the mean Intersection Over Union values so we can use them to plot when we end up training the semantic segmentation models.

The next step was to actually create the Semantic Segmentation model. We used pyramids of multi-scale features for pixel-wise classification to recover the spatial resolution lost in deep layers. To do this, we implemented a multi-scale encoder using a pretrained ResNet Model. We created an encoder class with methods such as get_channels to return the list of channels for each layer and get_block_size to get the size of the module. With the encoder done, we needed to create a decoder. The first step was to make a class that can set two layers to the same scale and sum them up. We needed to set up a lateral connection since that is necessary for the decoder.

Once the decoder and encoder classes were complete, we created a network using the Encoder-Decoder architecture. This was how the semantic segmentation class was set up:

```
class SemanticSegmentation(nn.Module):
   def __init__(self, num_classes, encoder_type=ResNetEncoder,
channels_out=128, final_upsample=False):
       ...
   def forward(self, x):
       ...
```

The arguments to initialize the class were number of classes, encoder type, number of channels of the output features, and a boolean if the prediction was to be upsampled to the original resolution. In the constructor, we initialized the fields, created the decoder and formed a classifier with num_classes as the output. We also defined a forward pass as a sequence of forward passes between all modules.

Finally, we had to train the model. We used nn.CrossEntropyLoss to help train it. Specifically, we defined the loss function as cross-entropy loss and we used a pre-set learning rate and weight decay for the optimizer configuration. For the sake of time and CPU size, we only set it to train for 10 epochs.

Results

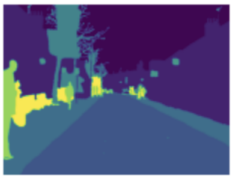
For the Results, we tested our program by first printing out some images in the dataset and the expected resulting masks that should be seen. This step helped us verify the images we have in the dataset and are using for our training. Afterwards we trained our program following the previous methods and printed out the resulting masks prediction made by the program in order to see the software's ability to detect the masks in the images made by the three different videos we're testing. We also printed out the graphs that represent test vs epochs, training, loss and learning rate.

The following three are the images from the three different videos we are training and the associated masks.

image

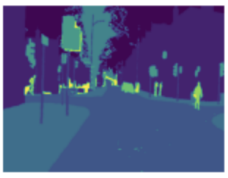
mask

image



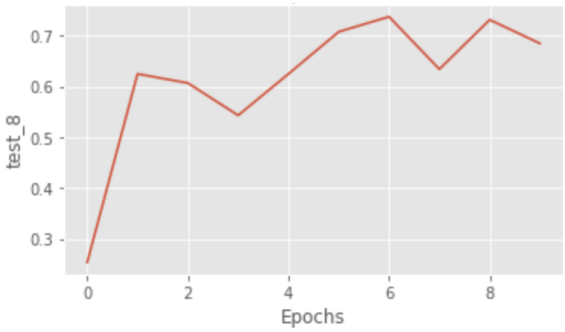
mask

image

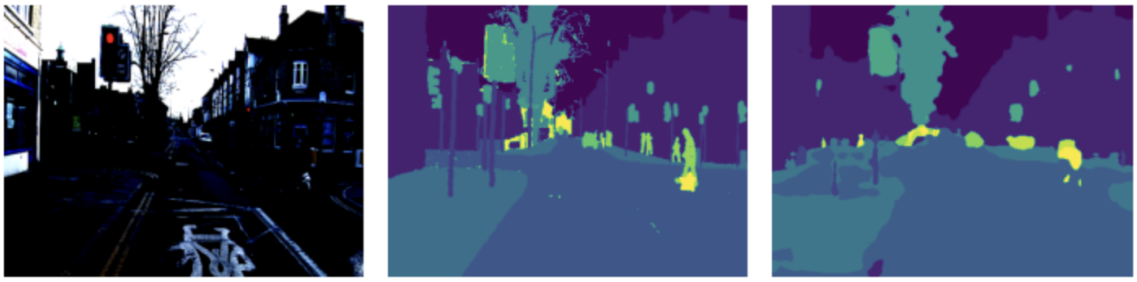


mask

The following two are some graphs printed that show the variation of "Training & Validation Loss", and "Epochs & test_8".



The following is an image and its true mask as well as the predicted mask after the program has been trained. We can see that the program has successfully identified the figures of vehicles and pedestrians we have present but not so well defined.



Conclusion

In conclusion, semantic segmentation is a powerful image processing technique that has numerous applications in the field of driver assistance systems. By dividing an image into segments that correspond to different objects and classes, semantic segmentation can provide the driver with valuable information about the environment and assist with various tasks. Through the use of machine learning algorithms and extensive training on relevant datasets, semantic segmentation can accurately identify and classify objects in real-time, making it an essential component of any driver assistance system. Overall, the use of semantic segmentation has the potential to significantly improve the safety and convenience of driving, and will likely continue to play a key role in the development of future driver assistance systems.