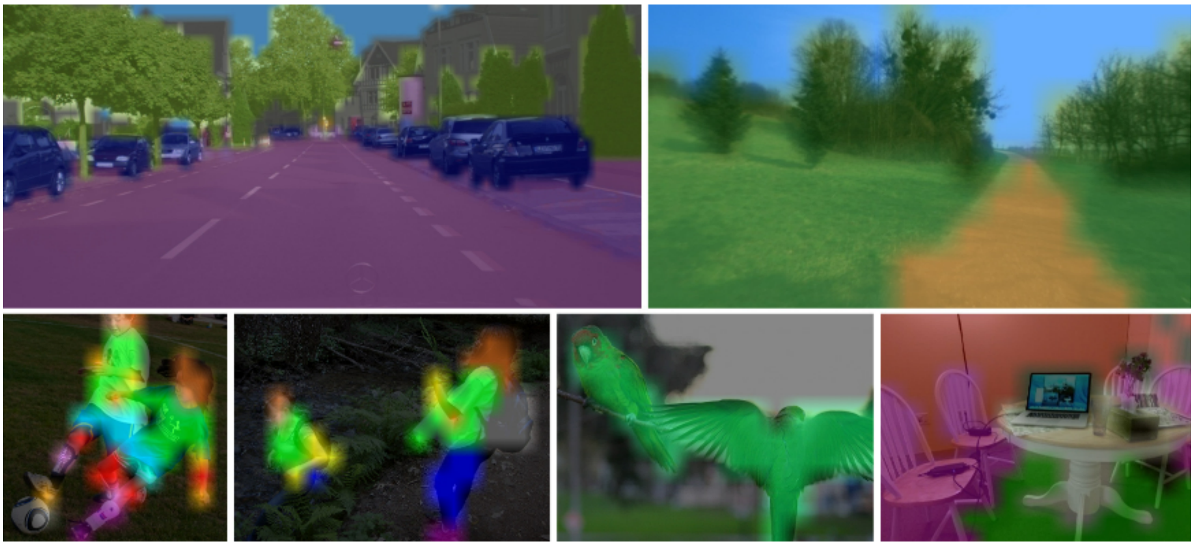
Semantic Segmentation

1. Introduction to Semantic Segmentation

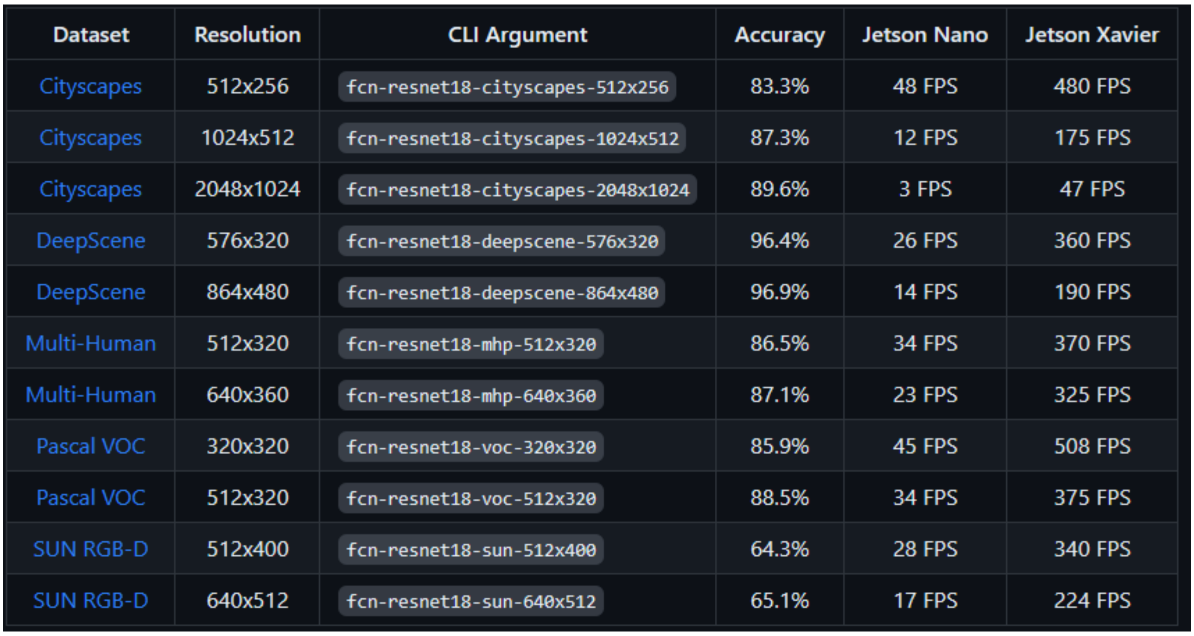
The next deep learning function we will cover in this tutorial is semantic segmentation. Semantic segmentation is based on image recognition, except that classification occurs at the pixel level, rather than the entire image. This is achieved by convolving a pre-trained image recognition backbone, which converts the model into a fully convolutional network (FCN) capable of pixel-by-pixel labeling. Segmentation is particularly useful for environmental perception, producing a dense per-pixel classification of many different potential objects for each scene, including the scene foreground and background.



segNet takes a 2D image as input and outputs a second image with a per-pixel classification mask overlaid. Each pixel of the mask corresponds to the classified object category. segNet can be used from Python and C++.

Download Other Models

Various pre-trained segmentation models for the FCN-ResNet18 network with real-time performance on Jetson. Below is a table of pre-trained semantic segmentation models available for use, and the associated --network parameters of segnet used to load them. They are based on a 21-class FCN-ResNet18 network, trained on various datasets and resolutions using PyTorch, and exported to ONNX format for loading into TensorRT.



Dataset	Resolution	CLI Argument	Accuracy	Jetson Nano	Jetson Xavier
Cityscapes	512x256	`fcn-resnet18-cityscapes-512x256`	83.3%	48 FPS	480 FPS
Cityscapes	1024x512	`fcn-resnet18-cityscapes-1024x512`	87.3%	12 FPS	175 FPS
Cityscapes	2048x1024	`fcn-resnet18-cityscapes-2048x1024`	89.6%	3 FPS	47 FPS
DeepScene	576x320	`fcn-resnet18-deepscene-576x320`	96.4%	26 FPS	360 FPS
DeepScene	864x480	`fcn-resnet18-deepscene-864x480`	96.9%	14 FPS	190 FPS
Multi-Human	512x320	`fcn-resnet18-mhp-512x320`	86.5%	34 FPS	370 FPS
Multi-Human	640x360	`fcn-resnet18-mhp-640x360`	87.1%	23 FPS	325 FPS
Pascal VOC	320x320	`fcn-resnet18-voc-320x320`	85.9%	45 FPS	508 FPS
Pascal VOC	512x320	`fcn-resnet18-voc-512x320`	88.5%	34 FPS	375 FPS
SUN RGB-D	512x400	`fcn-resnet18-sun-512x400`	64.3%	28 FPS	340 FPS
SUN RGB-D	640x512	`fcn-resnet18-sun-640x512`	65.1%	17 FPS	224 FPS

Here you can download the model you want from https://github.com/dusty-nv/jetson-inference

2. Image semantic segmentation

Here is an example of segmenting a city street scene using the Cityscapes model:

After building the project, make sure your terminal is in the aarch64/bin directory:

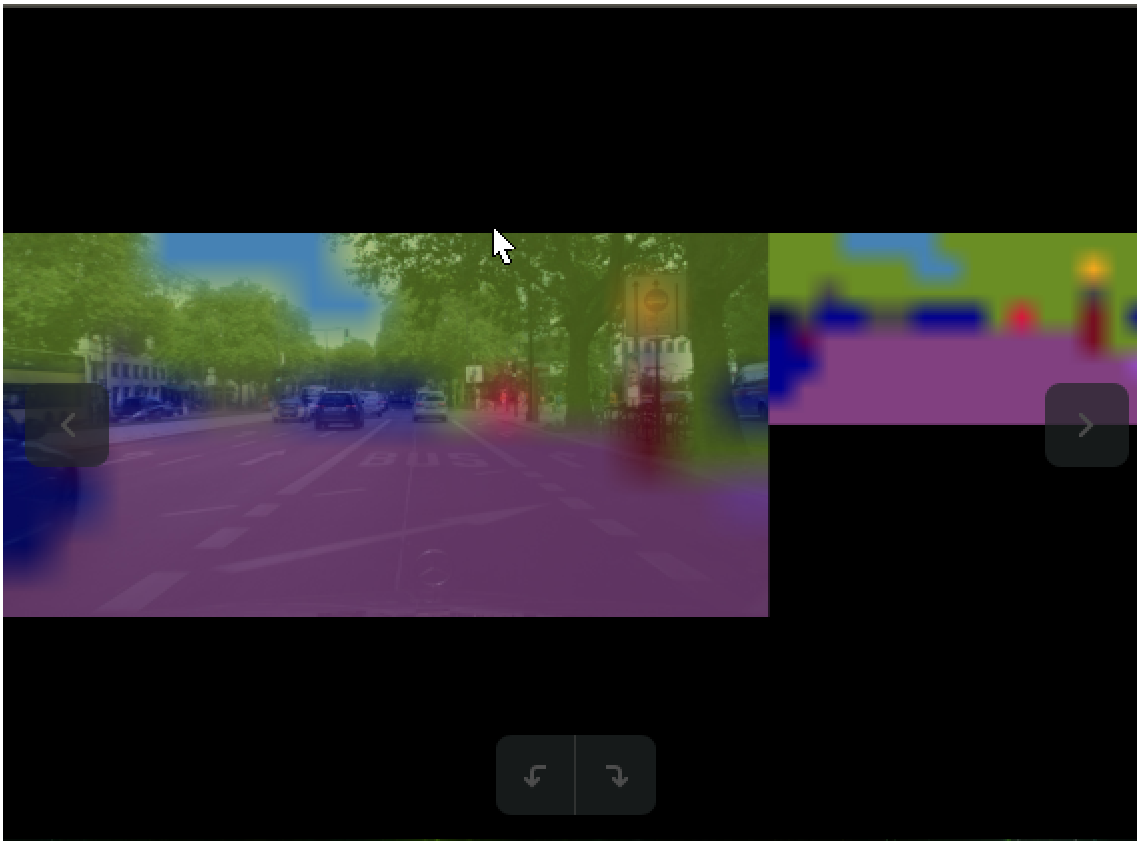
```
cd jetson-inference/build/aarch64/bin
```

Here are some examples using the fcn-resnet18-cityscapes model:

--network= You can put your downloaded model file in here. For example, here is fcn-resnet18-cityscapes

```
# C++
$ ./segnet --network=fcn-resnet18-cityscapes images/city_0.jpg
images/test/output.jpg

# Python
$ ./segnet.py --network=fcn-resnet18-cityscapes images/city_0.jpg
images/test/output.jpg
```



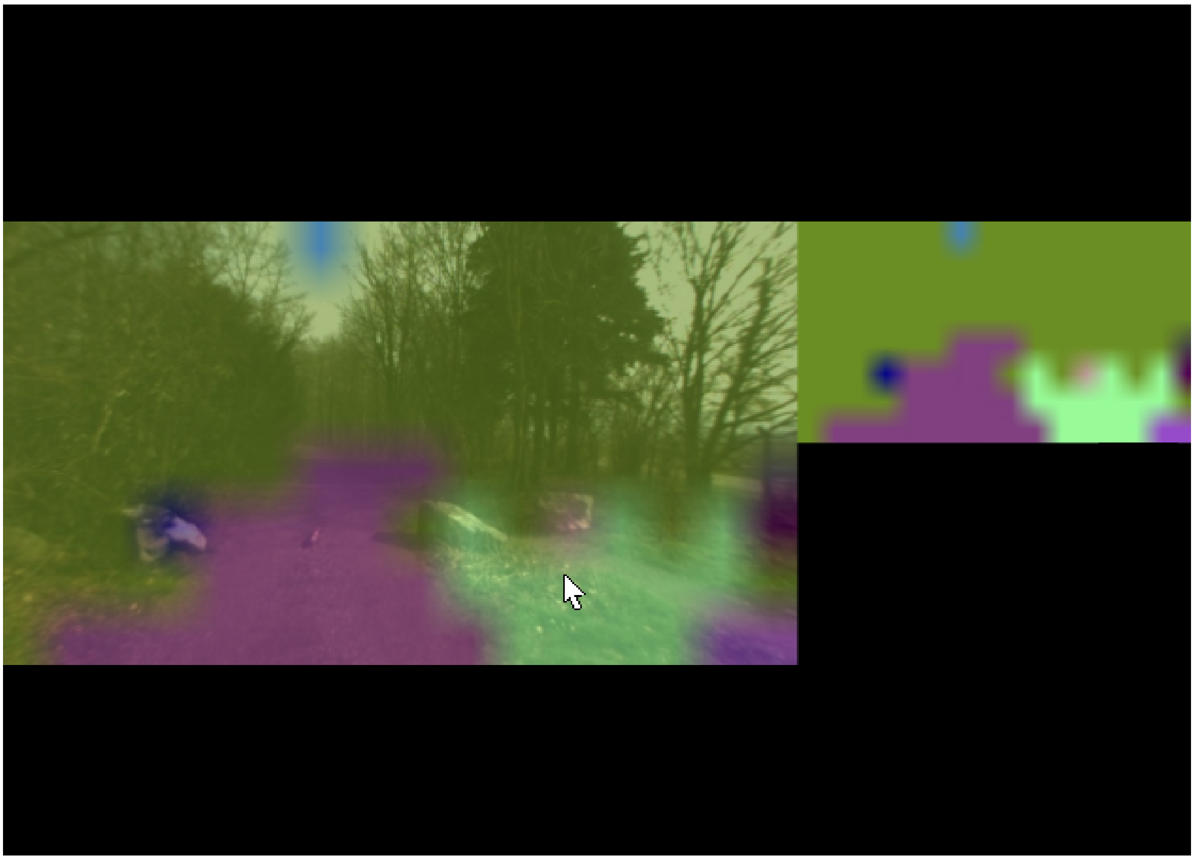
The following example is a DeepScene dataset consisting of off-road forest trails and vegetation, which helps path tracking for outdoor robots.

C++

```
$ ./segnet --network=fcn-resnet18-deepscene images/trail_0.jpg
images/test/output_overlay.jpg # overlay
$ ./segnet --network=fcn-resnet18-deepscene --visualize=mask images/trail_0.jpg
images/test/output_mask.jpg # mask
```

python

```
$ ./segnet.py --network=fcn-resnet18-deepscene images/trail_0.jpg
images/test/output_overlay.jpg # overlay
$ ./segnet.py --network=fcn-resnet18-deepscene --visualize=mask
images/trail_0.jpg images/test/output_mask.jpg # mask
```



Note: If you build your own environment, you need to download the model file to the network folder to run the above program. You can directly enter the above program using the image we provide

3. Run the real-time camera segmentation demonstration

The `segnet.cpp` / `segnet.py` samples we used before can also be used for real-time camera streams. Supported camera types include:

- MIPI CSI cameras (csi://0)
- V4L2 cameras (/dev/video0)
- RTP/RTSP streams (rtsp://username:password@ip:port)

Here are some typical scenarios for launching the program - for available models

C++

```
$ ./segnet --network=<model> csi://0 # MIPI CSI camera
$ ./segnet --network=<model> /dev/video0 # V4L2 camera
$ ./segnet --network=<model> /dev/video0 output.mp4 # save to video file
```

python

```
$ ./segnet.py --network=<model> csi://0 # MIPI CSI camera
$ ./segnet.py --network=<model> /dev/video0 # V4L2 camera
$ ./segnet.py --network=<model> /dev/video0 output.mp4 # save to video file
```

Where model is something we can choose. The model I use here is fcn-resnet18-mhp.

The OpenGL window shows the live camera stream with the segmentation output superimposed, and a solid segmentation mask for clarity. Here are some examples to try with different models:

```
# C++
$ ./segnet --network=fcn-resnet18-deepscene /dev/video0

# Python
$ ./segnet.py --network=fcn-resnet18-deepscene /dev/video0
```