DetectNet

조 영 혁

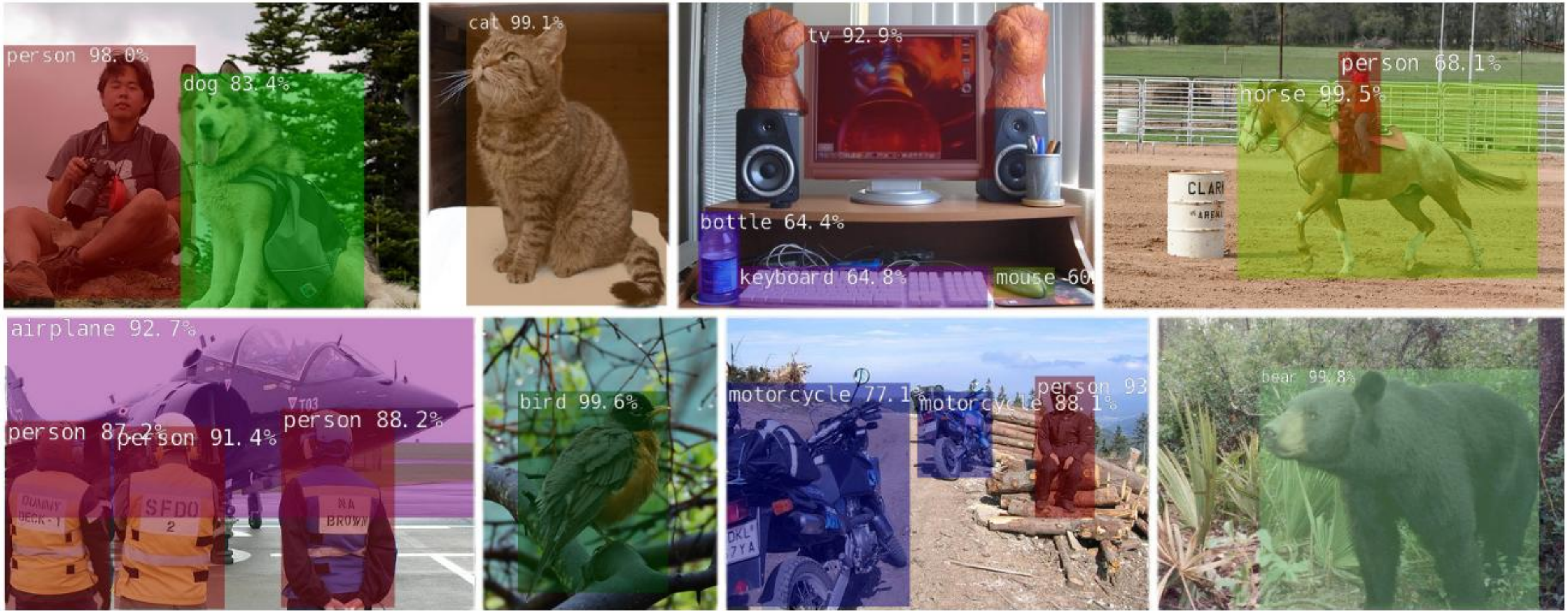
노다시스템

DetectNet

Locating Objects with DetectNet

DetectNet=> Image recognition

- 이미지에서 물체 감지하고 물체이 위치를 찾음



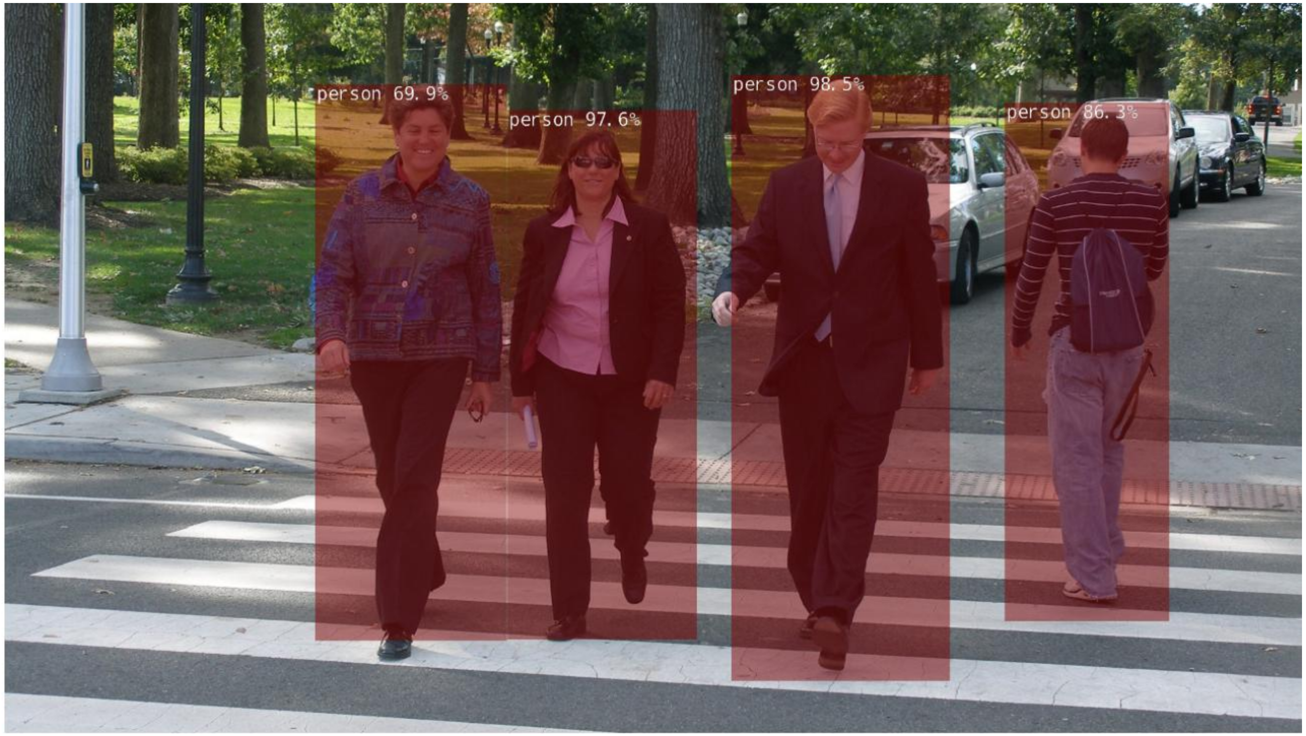
DetectNet

1. console 프로그램을 이용한 jetson 사용하기

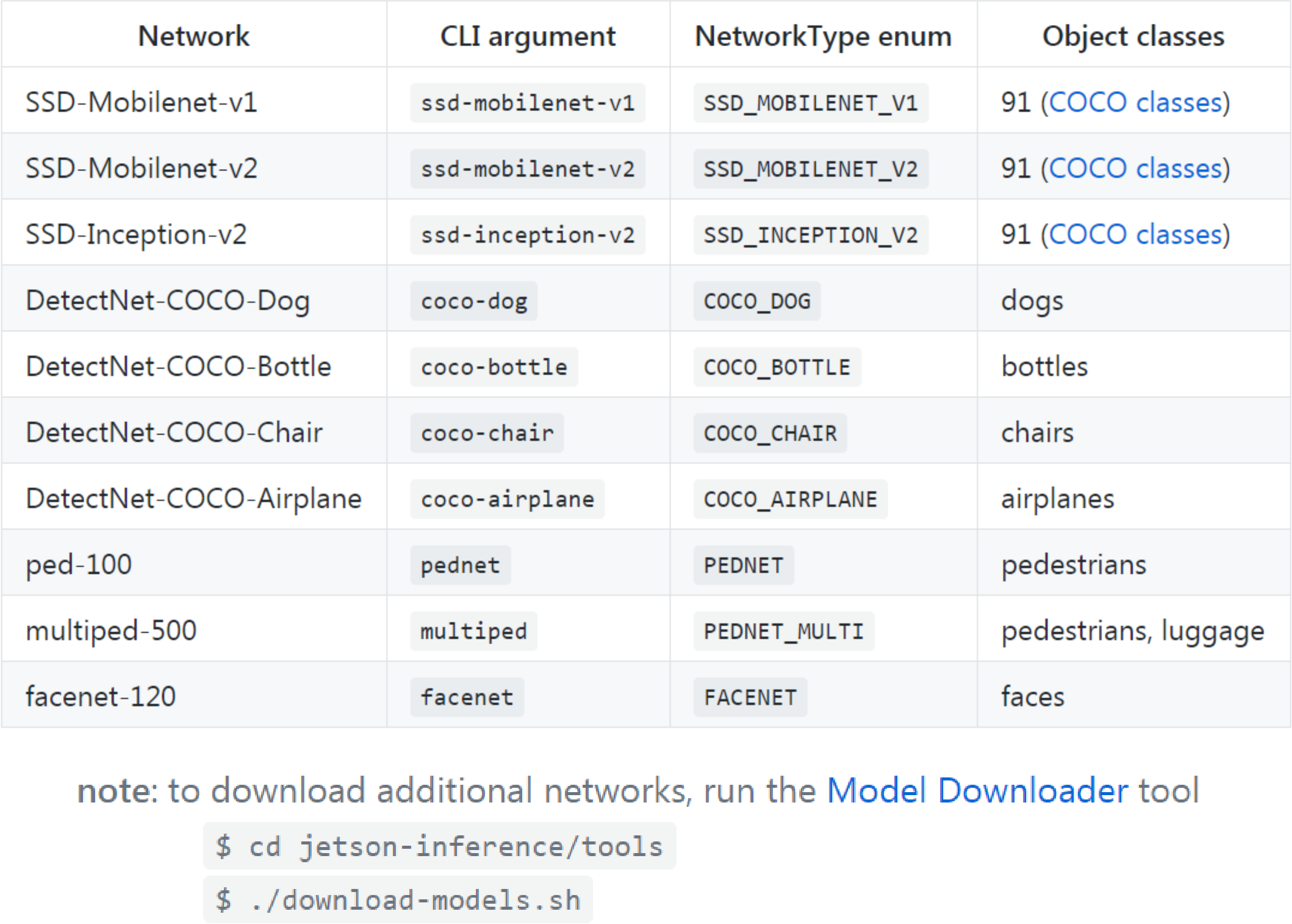
- classifcation networks that have been trained on large datasets to identify scenes and object

< **Using the Console Program on Jetson >**

1. Open Terminal
2. $ cd jetson-inference/build/aarch64/bin
3. $ ./detectnet-console.py --network=ssd-mobilenet-v2 images/peds_0.jpg output.jpg



DetectNet



Network	CLI argument	NetworkType enum	Object classes
SSD-Mobilenet-v1	`ssd-mobilenet-v1`	`SSD_MOBILENET_V1`	91 (COCO classes)
SSD-Mobilenet-v2	`ssd-mobilenet-v2`	`SSD_MOBILENET_V2`	91 (COCO classes)
SSD-Inception-v2	`ssd-inception-v2`	`SSD_INCEPTION_V2`	91 (COCO classes)
DetectNet-COCO-Dog	`coco-dog`	`COCO_DOG`	dogs
DetectNet-COCO-Bottle	`coco-bottle`	`COCO_BOTTLE`	bottles
DetectNet-COCO-Chair	`coco-chair`	`COCO_CHAIR`	chairs
DetectNet-COCO-Airplane	`coco-airplane`	`COCO_AIRPLANE`	airplanes
ped-100	`pednet`	`PEDNET`	pedestrians
multiped-500	`multiped`	`PEDNET_MULTI`	pedestrians, luggage
facenet-120	`facenet`	`FACENET`	faces

note: to download additional networks, run the Model Downloader tool

```
$ cd jetson-inference/tools
$ ./download-models.sh
```

DetectNet

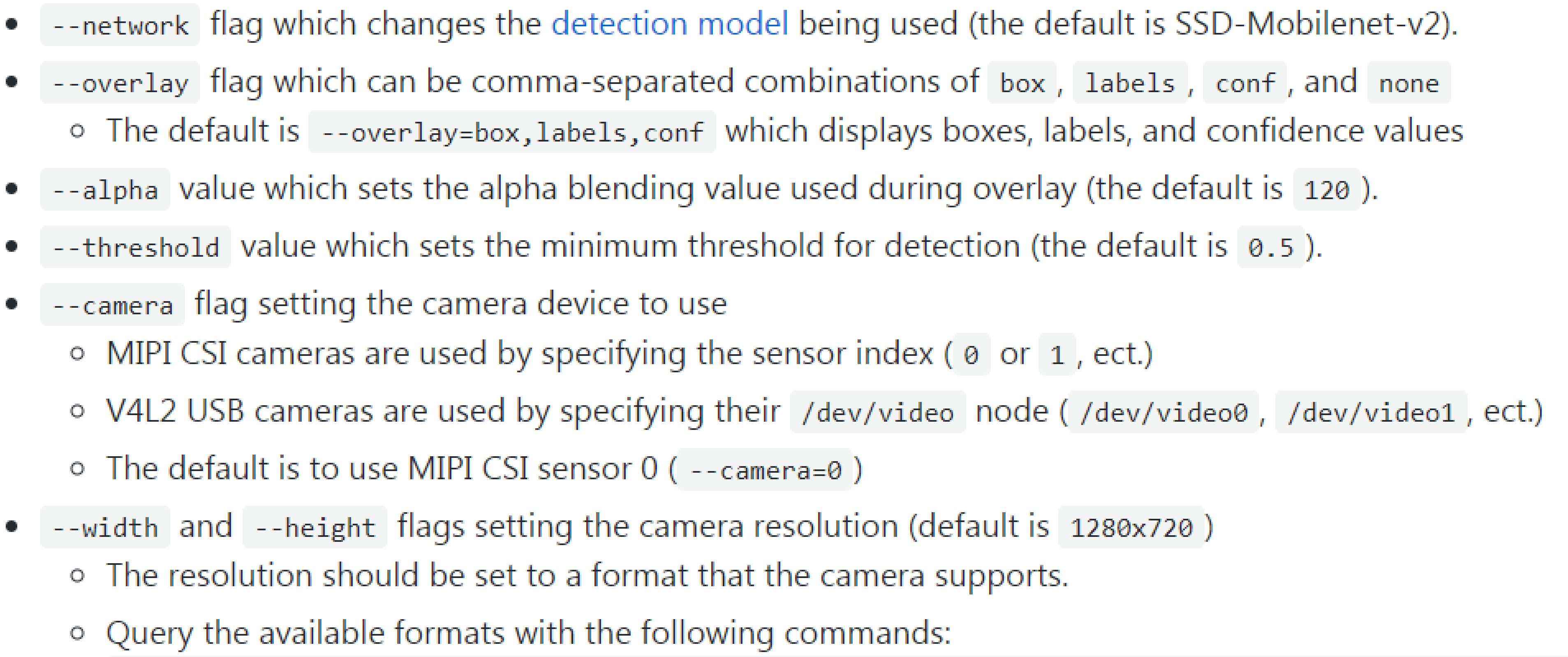
```
$ ./detectnet-console.py --network=ssd-inception-v2 input.jpg output.jpg
```

이미지	Network	Network	Network	Network	Network
	확률	확률	확률	확률	확률
개					
	(%)				
고양이					
사과					
배					
오렌지					
사람					

DetectNet

2. **Running the Live Camera Detection Demo**

```
$ ./detectnet-camera.py # using SSD-Mobilenet-v2, default MIPI CSI camera (1280x720)
$ ./detectnet-camera.py --network=ssd-inception-v2 # using SSD-Inception-v2, default MIPI CSI camera (1280x720)
$ ./detectnet-camera.py --width=640 --height=480 # using SSD-Mobilenet-v2, default MIPI CSI camera (640x480)
$ ./detectnet-camera.py --network=coco-dog
```

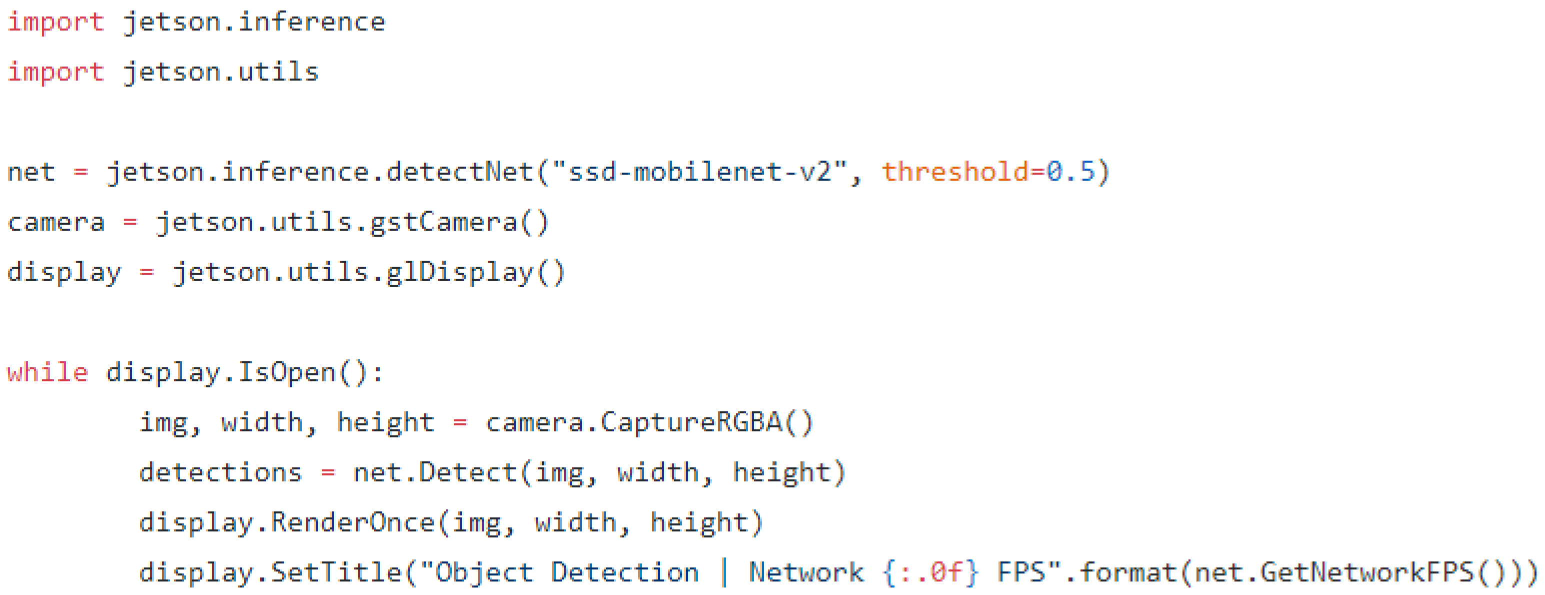


- `--network` flag which changes the detection model being used (the default is SSD-Mobilenet-v2).
- `--overlay` flag which can be comma-separated combinations of `box`, `labels`, `conf`, and `none`
  - The default is `--overlay=box,labels,conf` which displays boxes, labels, and confidence values
- `--alpha` value which sets the alpha blending value used during overlay (the default is `120`).
- `--threshold` value which sets the minimum threshold for detection (the default is `0.5`).
- `--camera` flag setting the camera device to use
  - MIPI CSI cameras are used by specifying the sensor index (`0` or `1`, ect.)
  - V4L2 USB cameras are used by specifying their `/dev/video` node (`/dev/video0`, `/dev/video1`, ect.)
  - The default is to use MIPI CSI sensor 0 (`--camera=0`)
- `--width` and `--height` flags setting the camera resolution (default is `1280x720`)
  - The resolution should be set to a format that the camera supports.
  - Query the available formats with the following commands:

DetectNet

3. **Coding Your Own Object Detection Program (my-detection.py)**

#!/usr/bin/python



```
import jetson.inference
import jetson.utils

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.gstCamera()
display = jetson.utils.glDisplay()

while display.IsOpen():
        img, width, height = camera.CaptureRGBA()
        detections = net.Detect(img, width, height)
        display.RenderOnce(img, width, height)
        display.SetTitle("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

DetectNet

<Jetson.utils>
https://rawgit.com/dusty-nv/jetson-inference/python/docs/html/python/jetson.utils.html

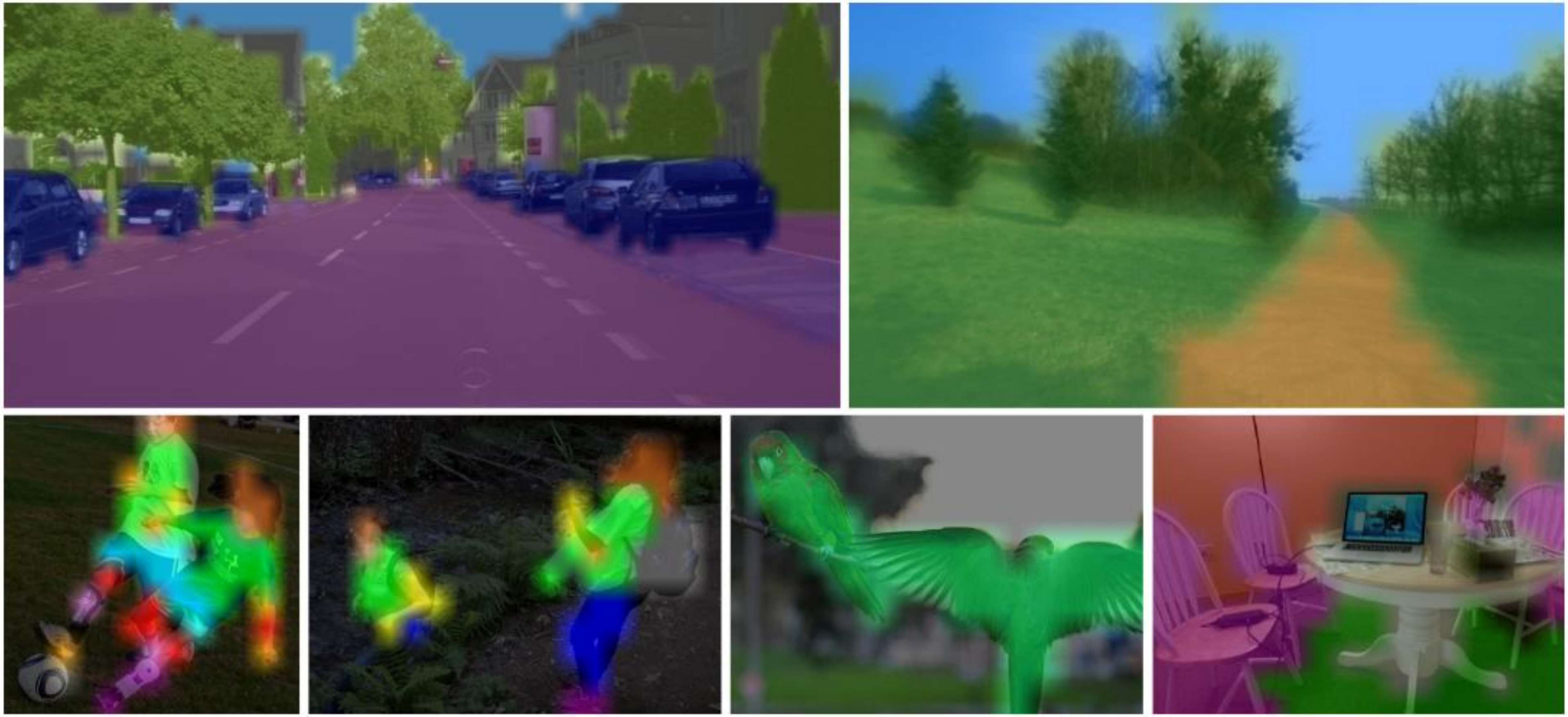
<Jetson.inference>
https://rawgit.com/dusty-nv/jetson-inference/python/docs/html/python/jetson.inference.html

SegNet

Semantic Segmentation with SegNet

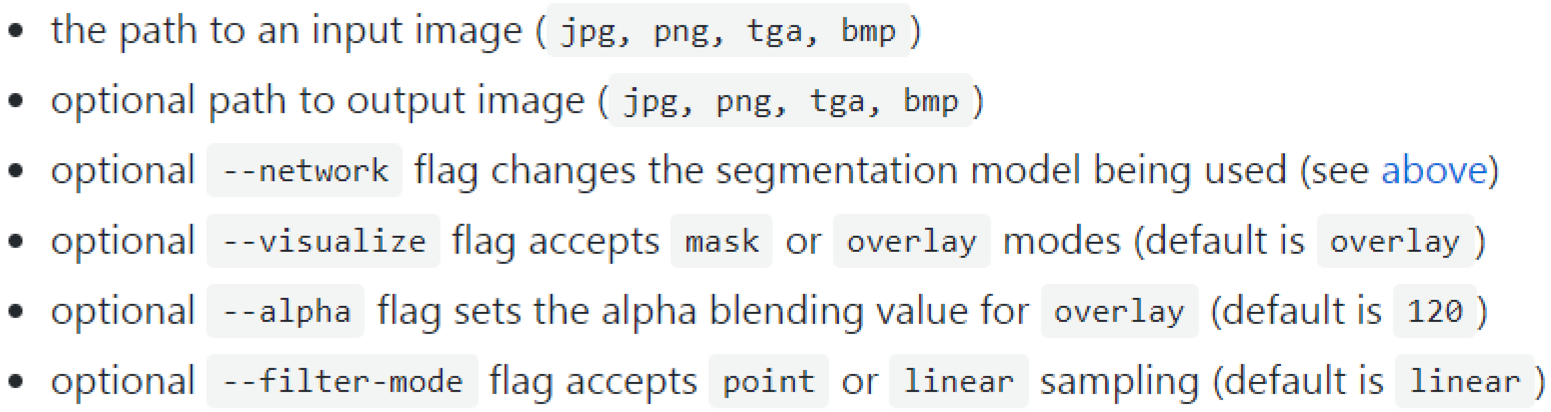
semantic segmentation

-물체를 segmentation



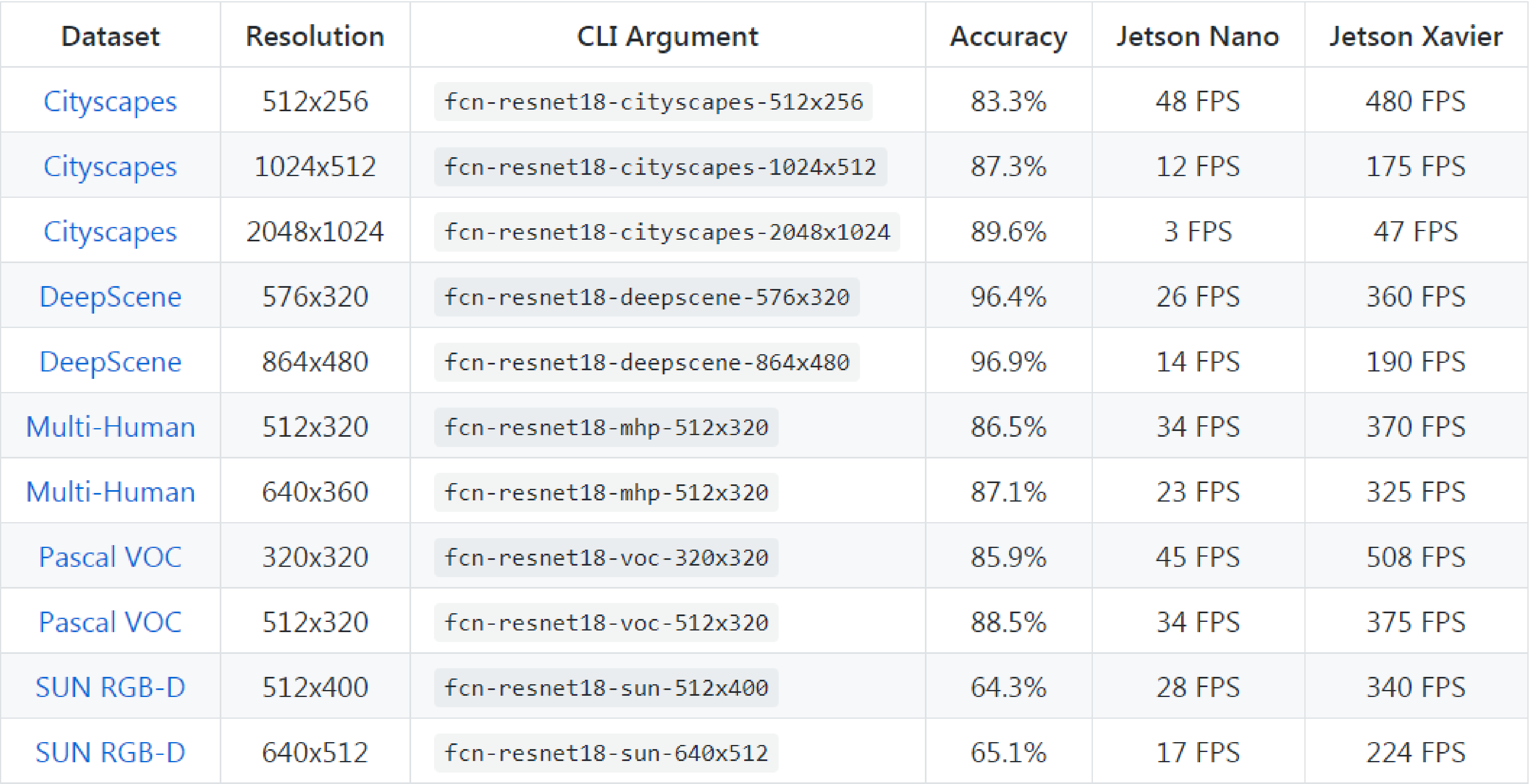
SegNet

```
$ ./segnet-console.py --network=<model> input.jpg output.jpg
$ ./segnet-console.py --network=<model> --alpha=200   input.jpg   output.jpg
$ ./segnet-console.py --network=<model> --visualize=mask   input.jpg  output.jpg
$ ./segnet-console.py --network=<model> --filter-mode=mask   input.jpg  output.jpg
```



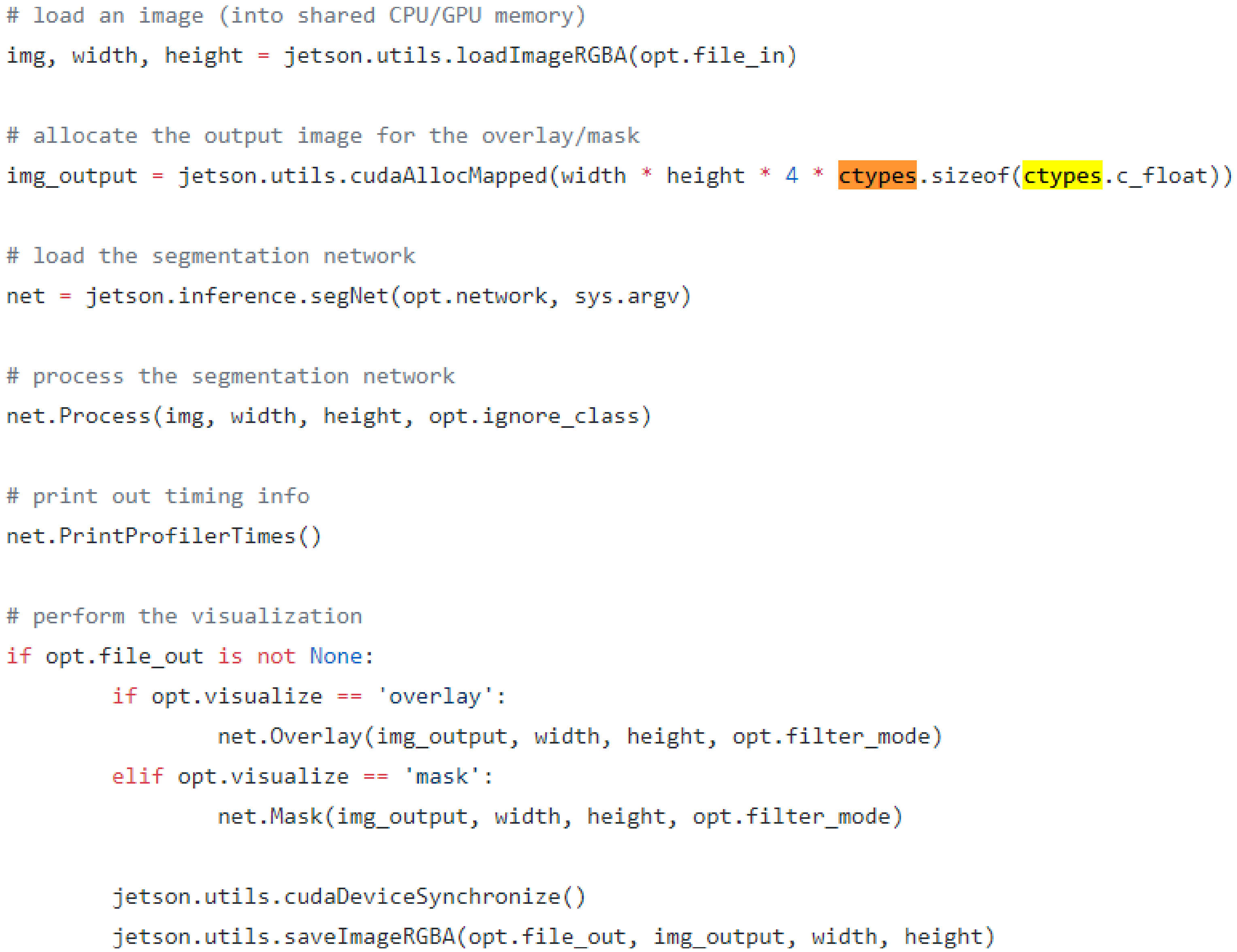
- the path to an input image (`jpg, png, tga, bmp`)
- optional path to output image (`jpg, png, tga, bmp`)
- optional `--network` flag changes the segmentation model being used (see above)
- optional `--visualize` flag accepts `mask` or `overlay` modes (default is `overlay`)
- optional `--alpha` flag sets the alpha blending value for `overlay` (default is `120`)
- optional `--filter-mode` flag accepts `point` or `linear` sampling (default is `linear`)

SegNet



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-512x320`	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

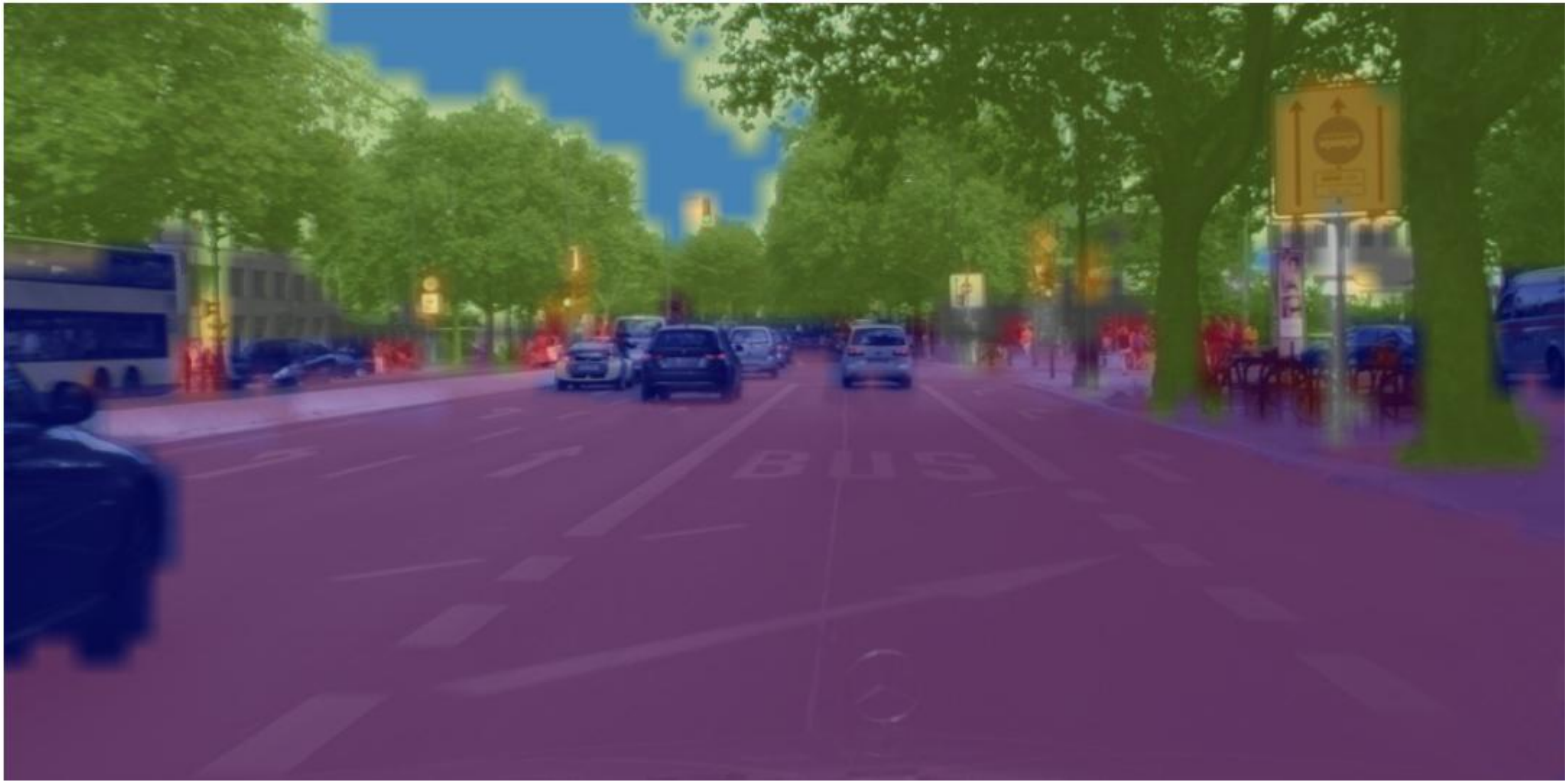
SegNet



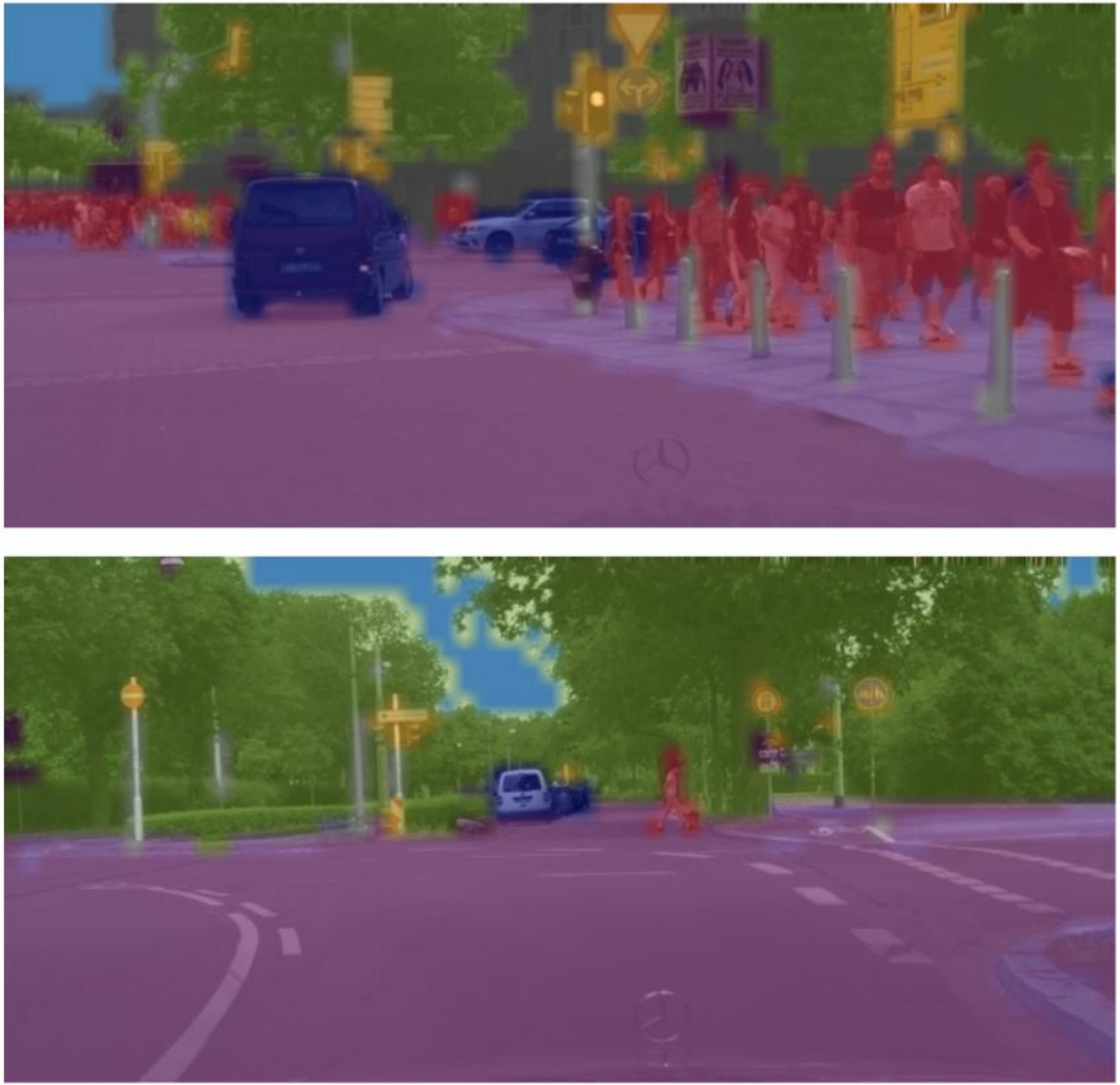
```
# load an image (into shared CPU/GPU memory)
img, width, height = jetson.utils.loadImageRGBA(opt.file_in)

# allocate the output image for the overlay/mask
img_output = jetson.utils.cudaAllocMapped(width * height * 4 * ctypes.sizeof(ctypes.c_float))

# load the segmentation network
net = jetson.inference.segNet(opt.network, sys.argv)

# process the segmentation network
net.Process(img, width, height, opt.ignore_class)

# print out timing info
net.PrintProfilerTimes()

# perform the visualization
if opt.file_out is not None:
        if opt.visualize == 'overlay':
                net.Overlay(img_output, width, height, opt.filter_mode)
        elif opt.visualize == 'mask':
                net.Mask(img_output, width, height, opt.filter_mode)

        jetson.utils.cudaDeviceSynchronize()
        jetson.utils.saveImageRGBA(opt.file_out, img_output, width, height)
```

SegNet - Cityscapes

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



SegNet- Cityscapes

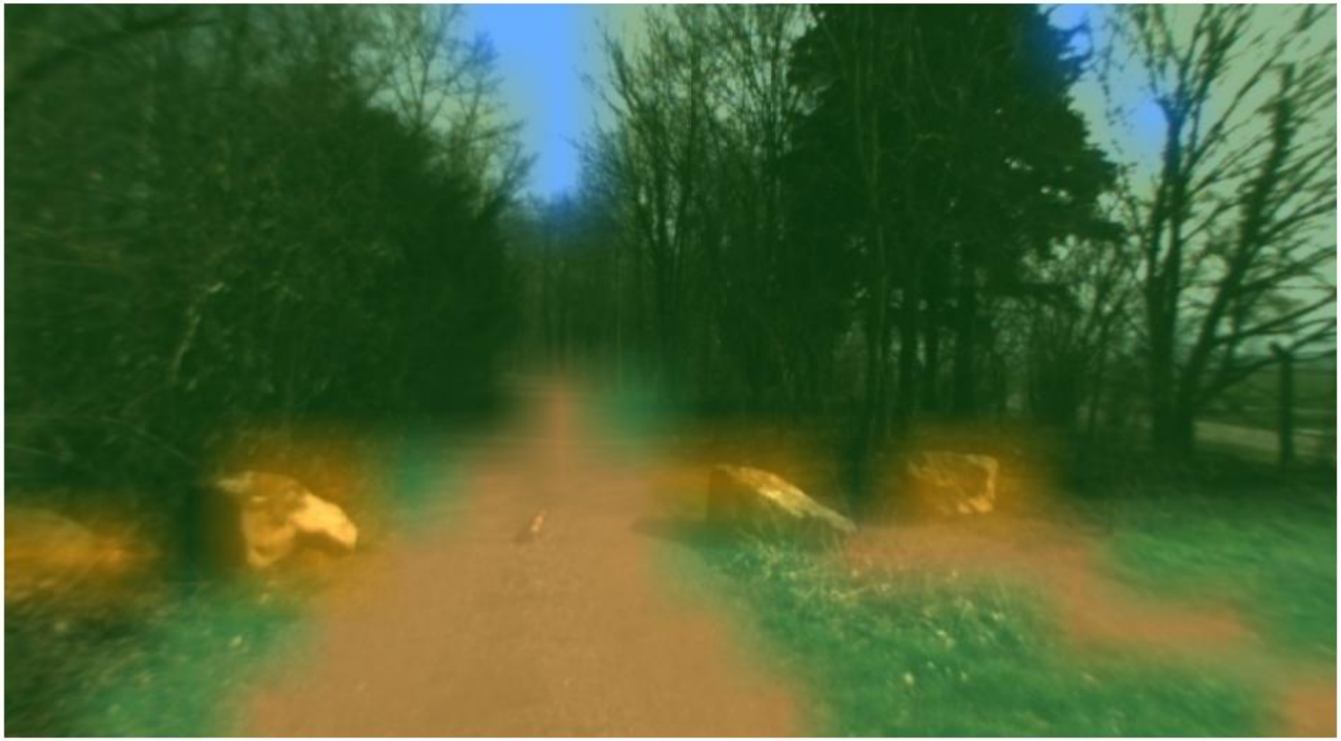


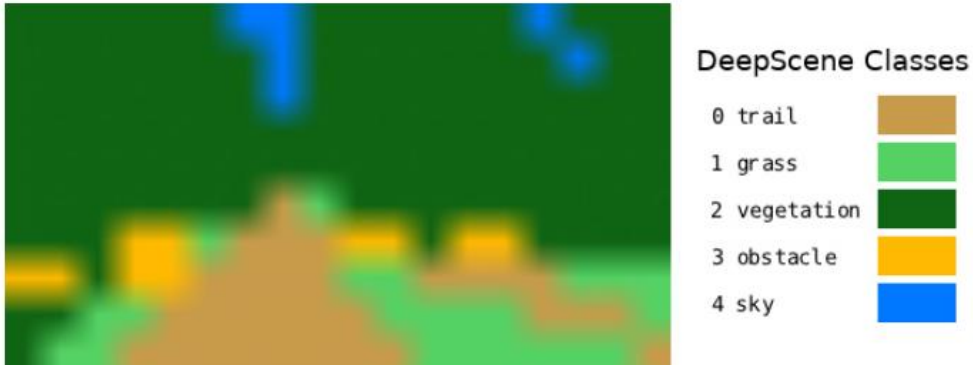
Cityscapes Classes

0	void	⬛
1	ego_vehicle	🟪
2	ground	🟪
3	road	🟪
4	sidewalk	🟪
5	building	⬛
6	wall	🟦
7	fence	🟫
8	pole	⬜
9	traffic_light	🟧
10	traffic_sign	🟧
11	vegetation	🟩
12	terrain	🟩
13	sky	🟦
14	person	🟥
15	car	🟦
16	truck	🟦
17	bus	🟦
18	train	🟦
19	motorcycle	🟦
20	bicycle	🟥

SegNet- DeepScene

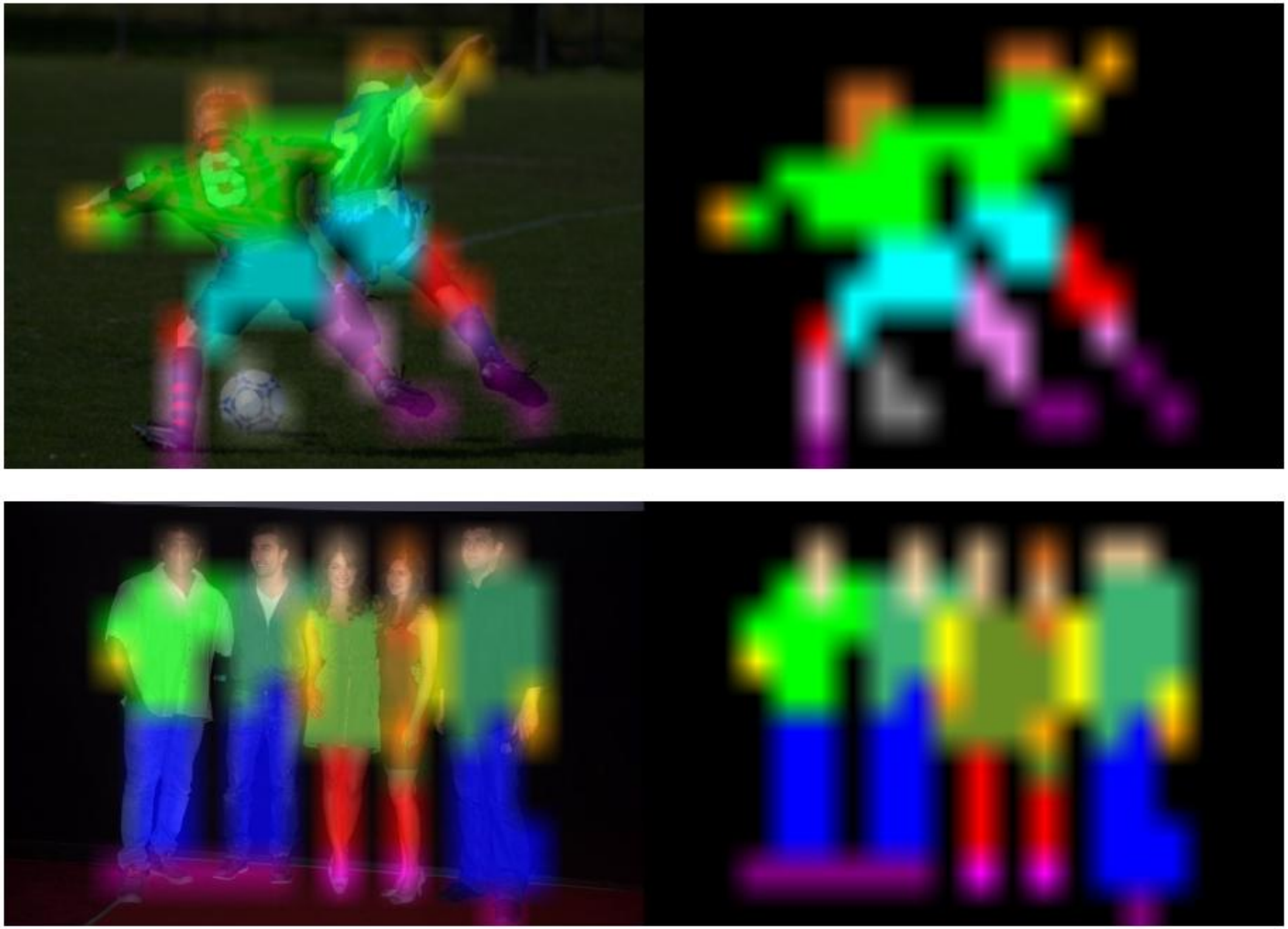
```
$ ./segnet-console.py --network=fcn-resnet18-deepscene images/trail_0.jpg output_overlay.jpg
$ ./segnet-console.py --network=fcn-resnet18-deepscene --visualize=mask images/trail_0.jpg output_mask.jpg
```

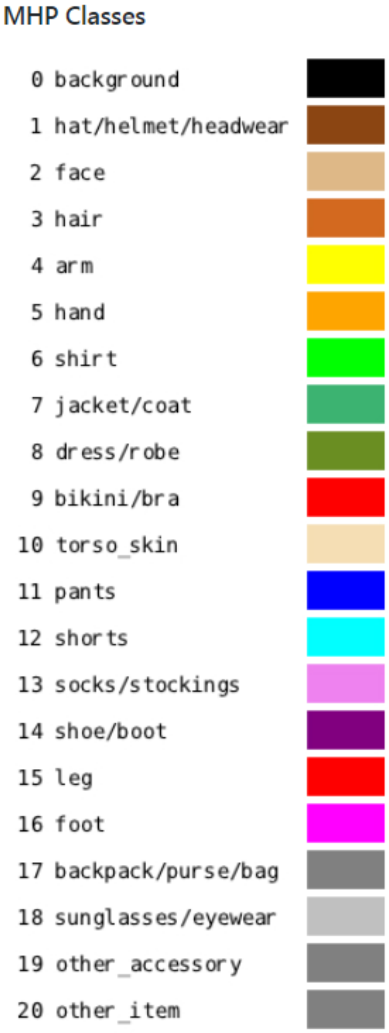




SegNet- Multi-Human Parsing (MHP)

```
$ ./segnet-console.py --network=fcn-resnet18-mhp images/humans_0.jpg output.jpg
```



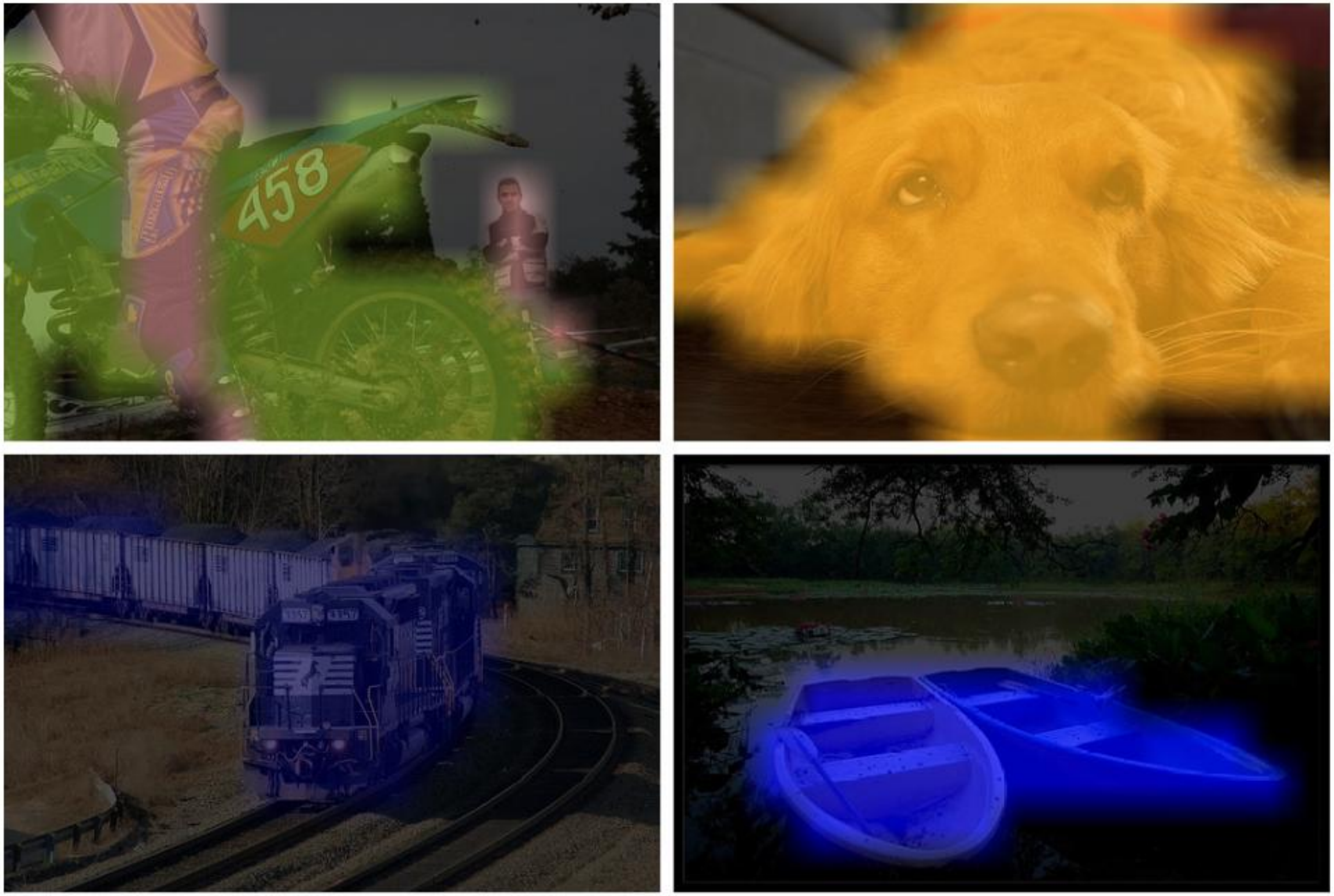


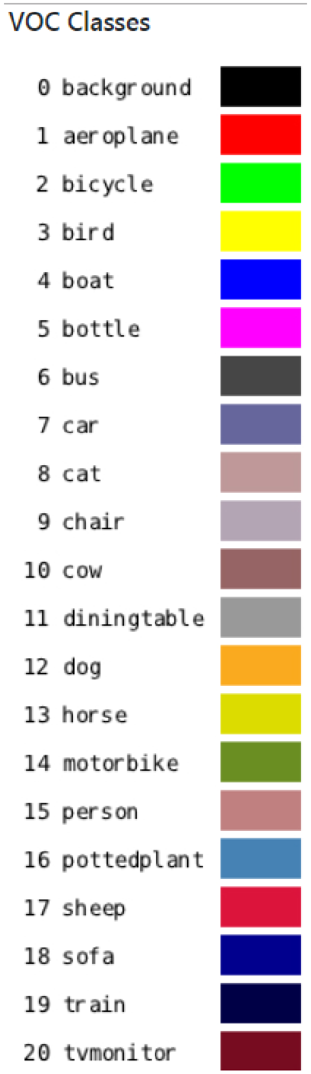
MHP Classes

0	background	⬛
1	hat/helmet/headwear	🟫
2	face	🟫
3	hair	🟧
4	arm	🟨
5	hand	🟧
6	shirt	🟩
7	jacket/coat	🟩
8	dress/robe	🟩
9	bikini/bra	🟥
10	torso_skin	🟨
11	pants	🟦
12	shorts	🟦
13	socks/stockings	🟪
14	shoe/boot	🟪
15	leg	🟥
16	foot	🟪
17	backpack/purse/bag	⬜
18	sunglasses/eyewear	⬜
19	other_accessory	⬜
20	other_item	⬜

SegNet-Pascal VOC

```
$ ./segnet-console.py --network=fcn-resnet18-voc images/object_0.jpg output.jpg
```





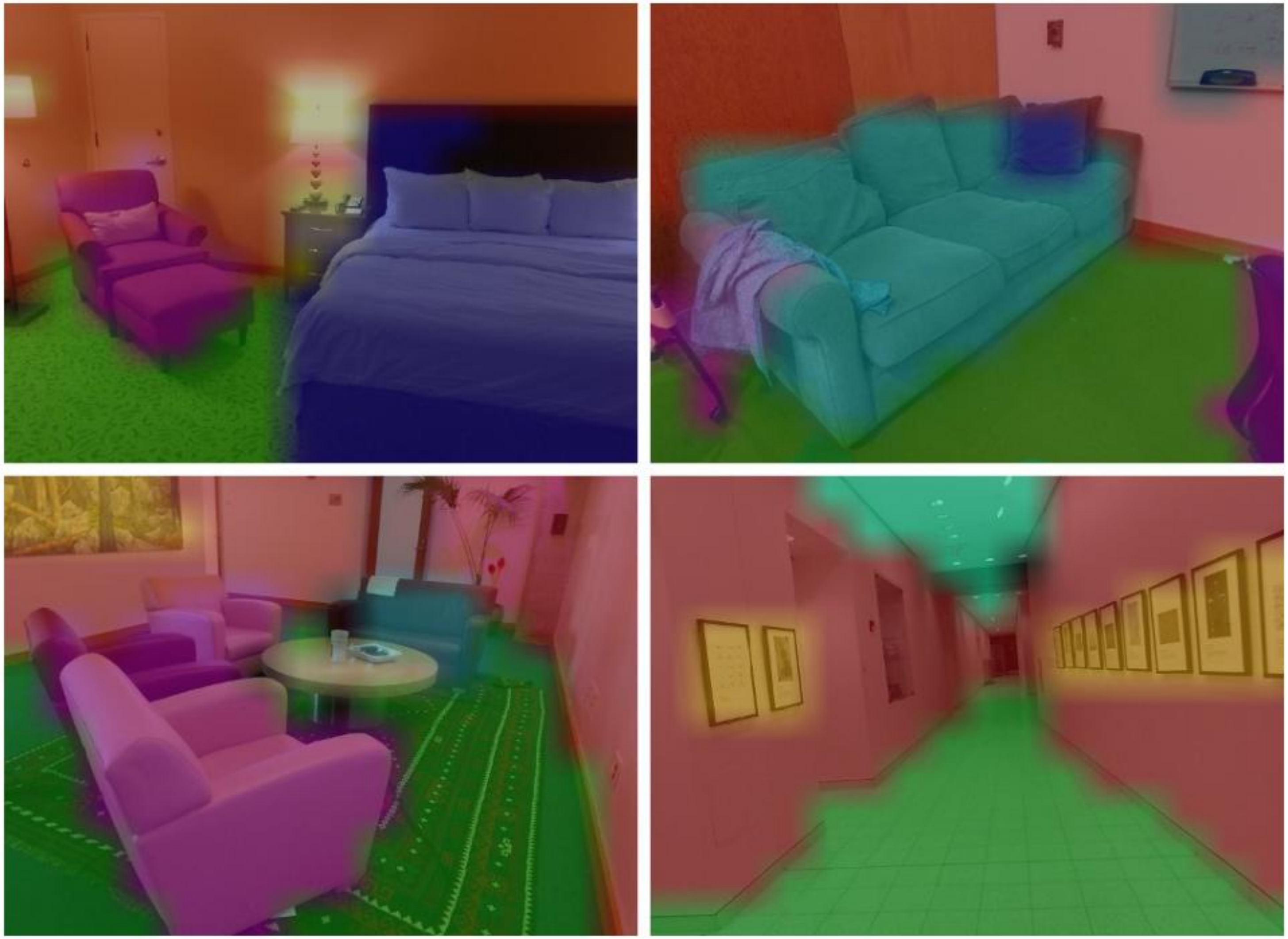
VOC Classes

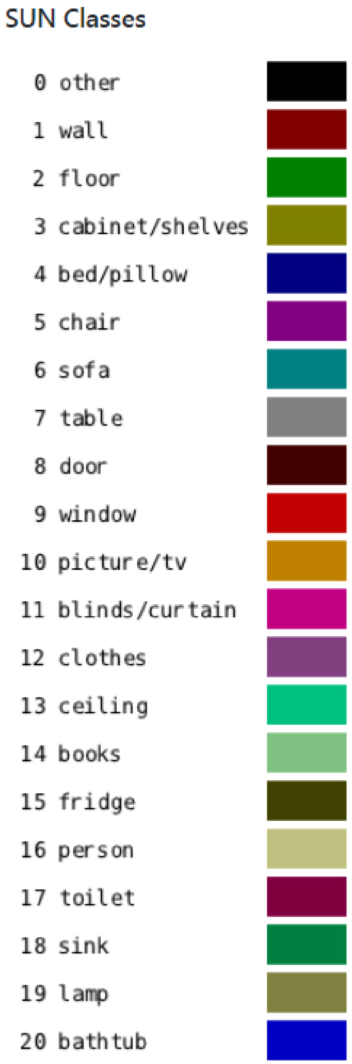
0	background	■
1	aeroplane	■
2	bicycle	■
3	bird	■
4	boat	■
5	bottle	■
6	bus	■
7	car	■
8	cat	■
9	chair	■
10	cow	■
11	diningtable	■
12	dog	■
13	horse	■
14	motorbike	■
15	person	■
16	pottedplant	■
17	sheep	■
18	sofa	■
19	train	■
20	tvmonitor	■

- people, animals, vehicles, hosehold

SegNet-SUN RGB-D

```
$ ./segnet-console.py --network=fcn-resnet18-sun images/room_0.jpg output.jpg
```





SUN Classes

0	other	⬛
1	wall	🟥
2	floor	🟩
3	cabinet/shelves	🟨
4	bed/pillow	🟦
5	chair	🟪
6	sofa	🟦
7	table	⬜
8	door	🟫
9	window	🟥
10	picture/tv	🟧
11	blinds/curtain	🟪
12	clothes	🟪
13	ceiling	🟩
14	books	🟩
15	fridge	🟫
16	person	🟨
17	toilet	🟥
18	sink	🟩
19	lamp	🟨
20	bathtub	🟦

- 사무실,집에서의 indoor objects and scenes

SegNet

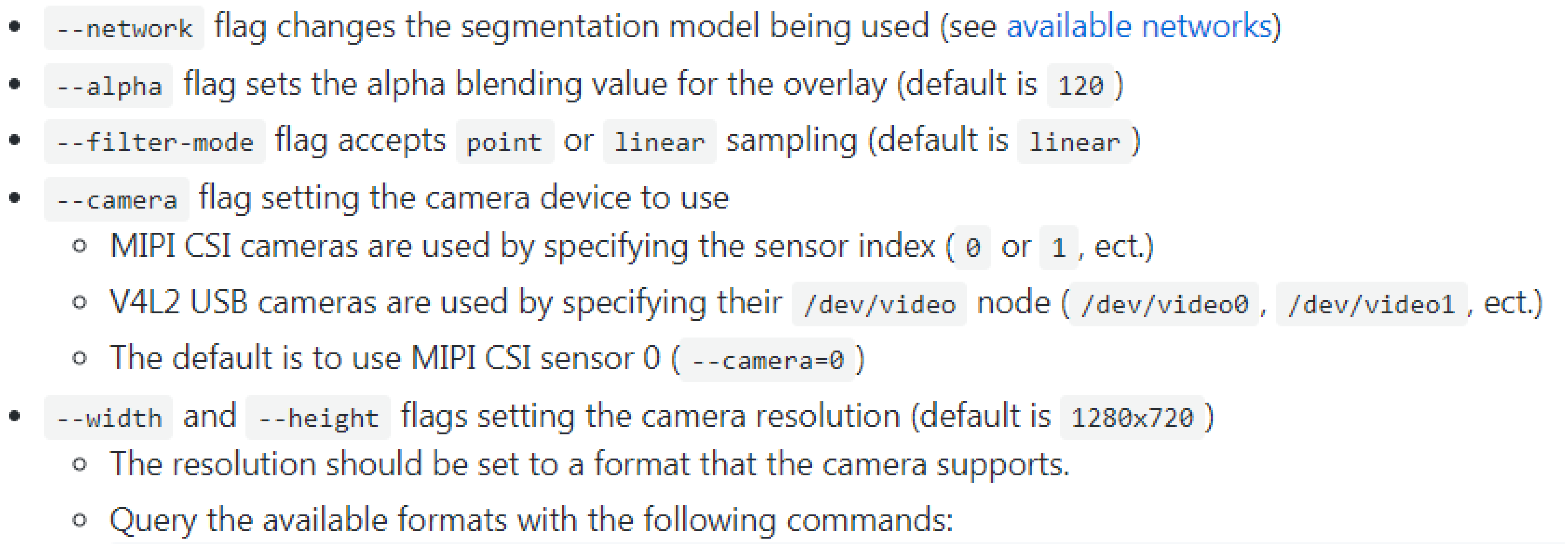
Directory 단위로 segmentation을 하는 batch 파일

```
$ ./segnet-batch.py    --network=<model>    <input-dir>    <output-dir>
```

SegNet

Running the Live Camera Segmentation Demo

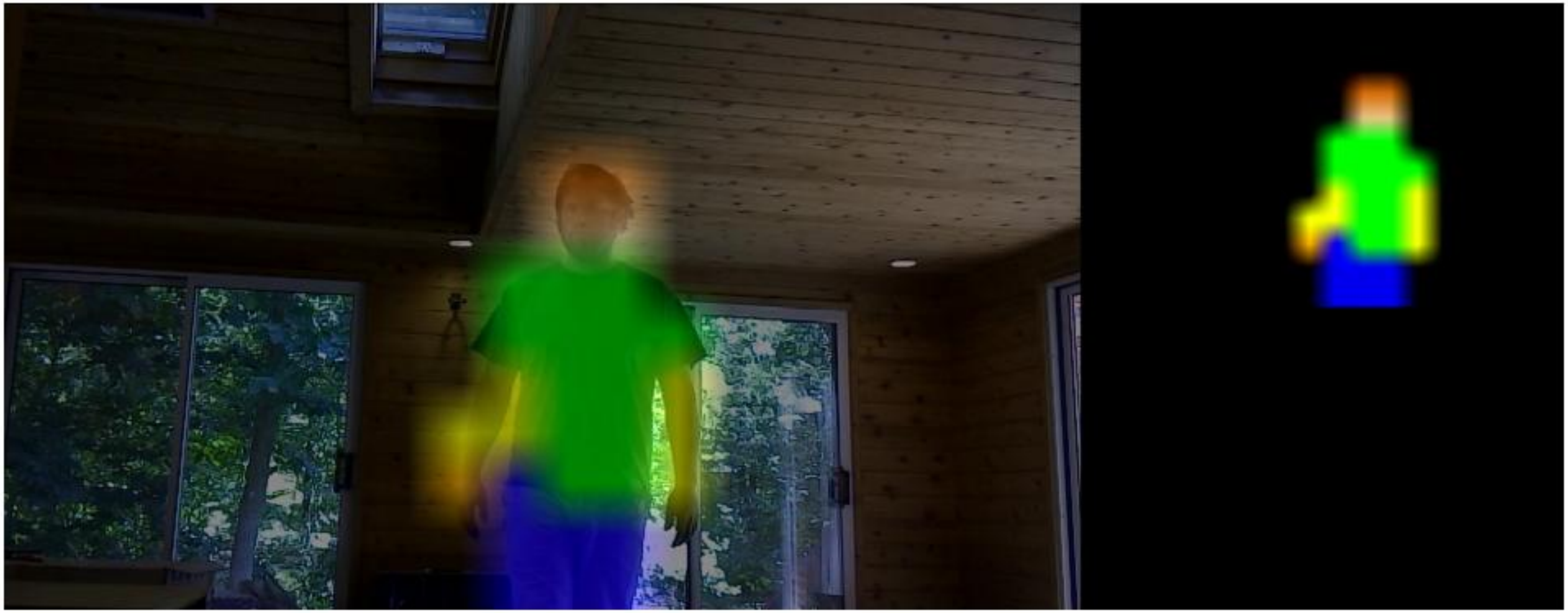
```
$ ./segnet-camera.py --network=fcn-resnet18-mhp
$ ./segnet-camera.py --width=640 --height=480
```



- `--network` flag changes the segmentation model being used (see available networks)
- `--alpha` flag sets the alpha blending value for the overlay (default is `120`)
- `--filter-mode` flag accepts `point` or `linear` sampling (default is `linear`)
- `--camera` flag setting the camera device to use
  - MIPI CSI cameras are used by specifying the sensor index (`0` or `1`, ect.)
  - V4L2 USB cameras are used by specifying their `/dev/video` node (`/dev/video0`, `/dev/video1`, ect.)
  - The default is to use MIPI CSI sensor 0 (`--camera=0`)
- `--width` and `--height` flags setting the camera resolution (default is `1280x720`)
  - The resolution should be set to a format that the camera supports.
  - Query the available formats with the following commands:

Running the Live Camera Segmentation Demo

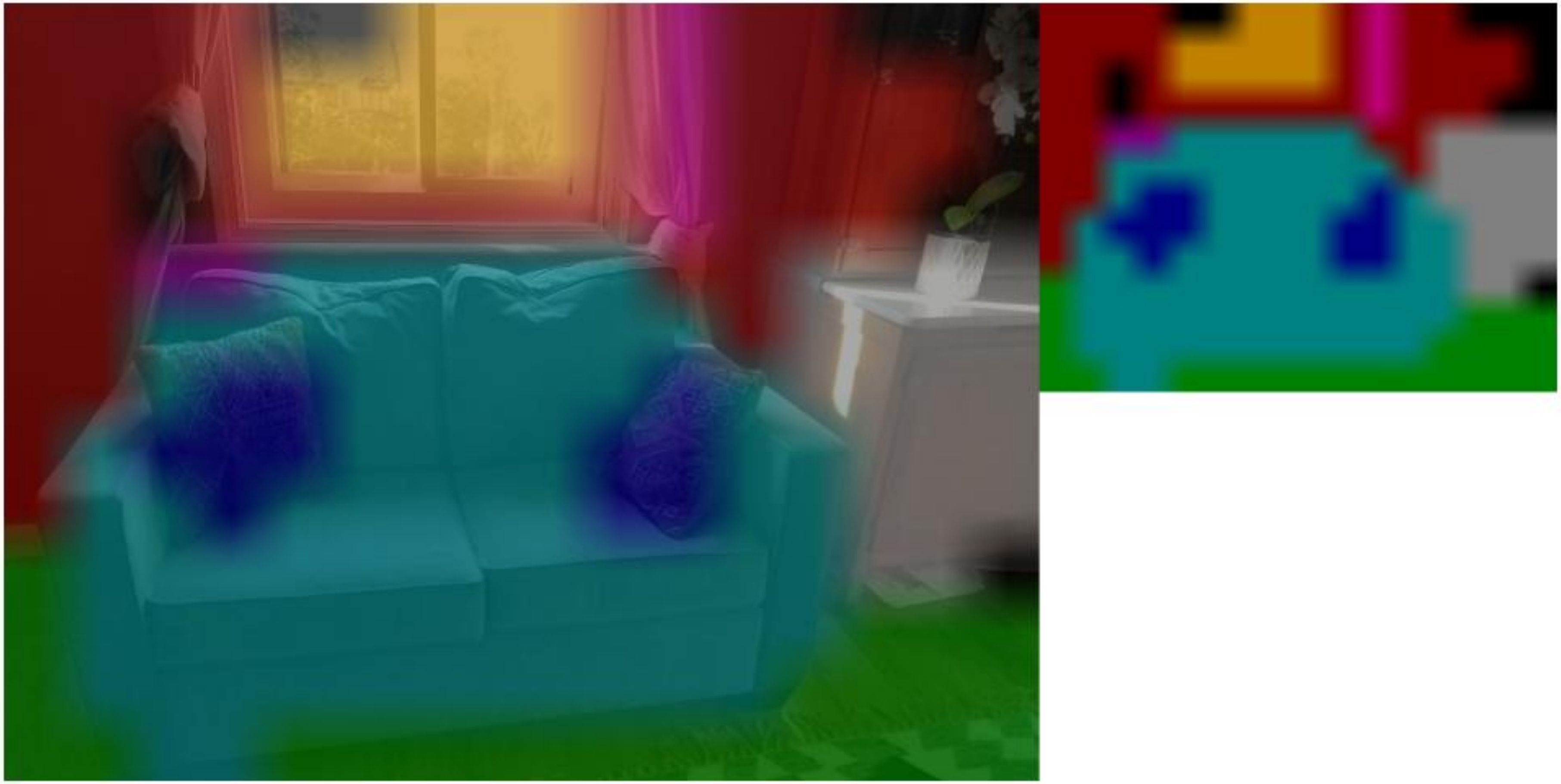
```
$ ./segnet-camera.py --network=fcn-resnet18-mhp
```



SegNet

Running the Live Camera Segmentation Demo

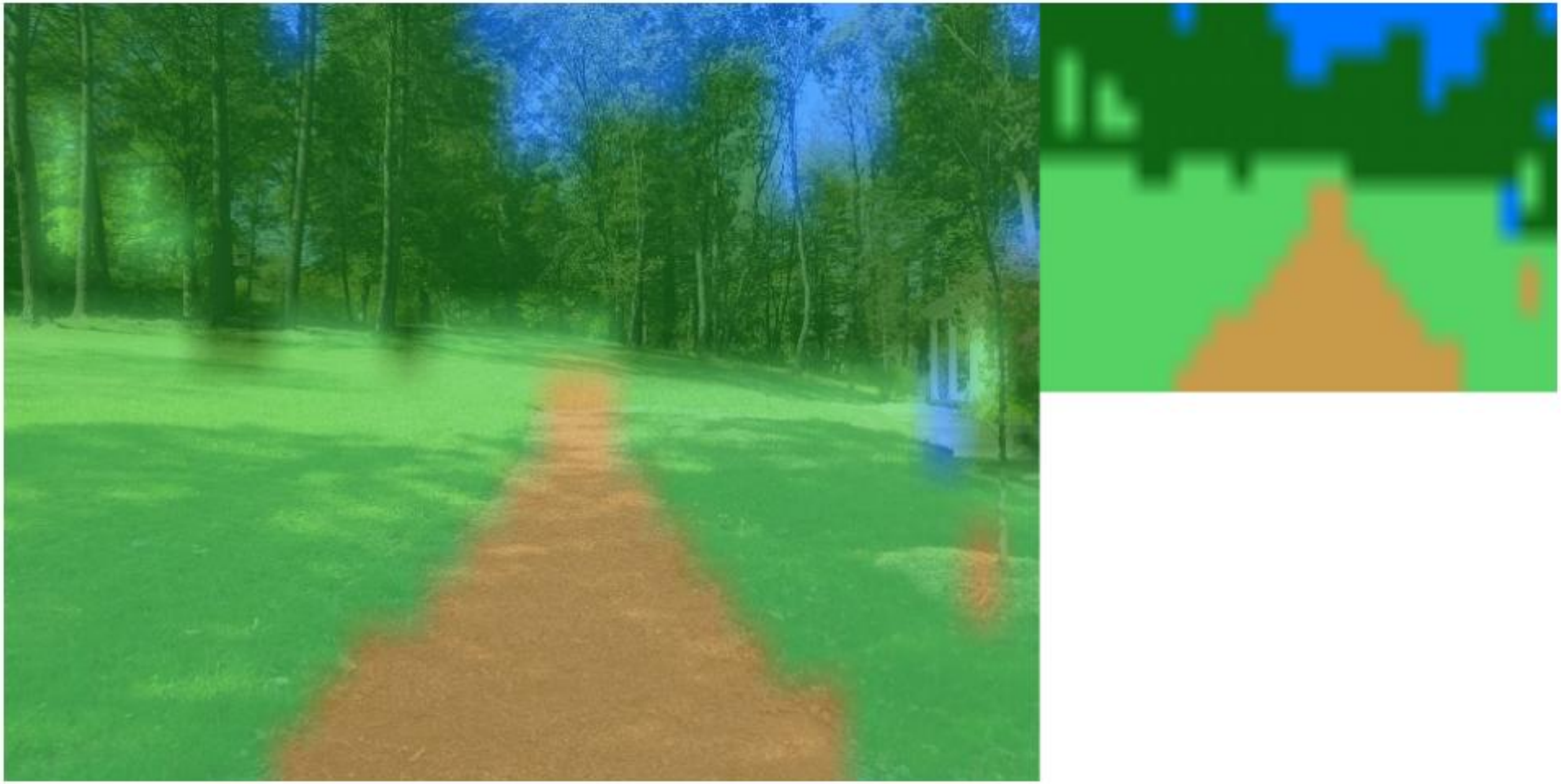
```
$ ./segnet-camera.py --network=fcn-resnet18-sun
```



SegNet

Running the Live Camera Segmentation Demo

```
$ ./segnet-camera.py --network=fcn-resnet18-deepscene
```



THANK YOU

Suggestions Questions