Image to Image Translation using cGans

Cheikh Tidiane Diop

Goal of the paper

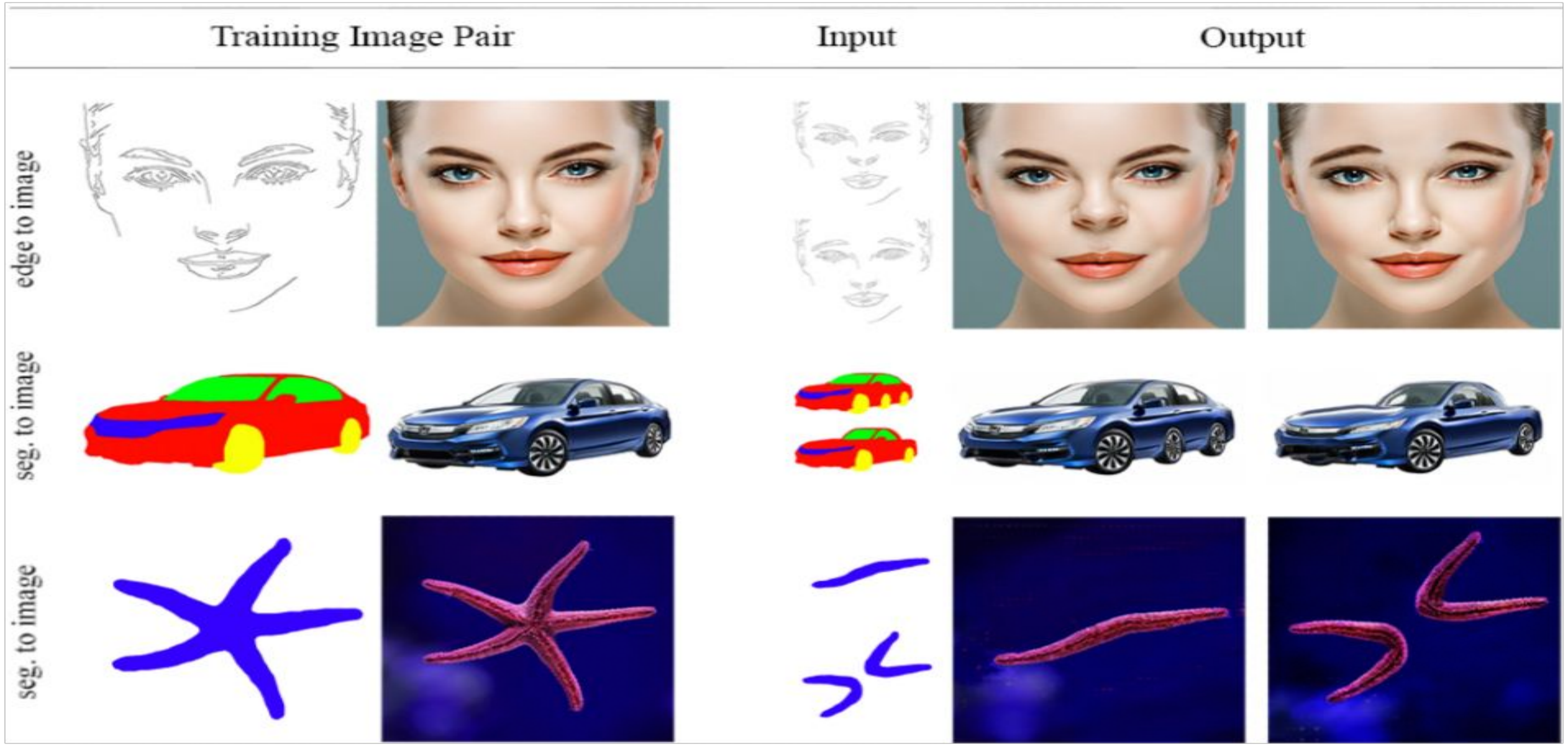
Goal of the paper

Develop a common framework for all the Image to Image translations tasks.

Image to Image Translations?

Translating one possible representation of a scene into another.

Instances


Training Image Pair
Input
Output
edge to image
seg. to image
seg. to image

Overview

- Image to Image Translation Tasks
- Photo -Segmentation Task and Cityscapes dataset
- Why CGans?
- Objective function
- General framework and Networks architectures used
- Training on the Cityscapes dataset
- Observations

Image to Image Translation Tasks

- Semantic labels↔photo
- Architectural labels→photo
- Map↔aerial photo
- BW→color photos
- Edges→photo
- Sketch→photo
- Day→night
- Thermal→color photos
- Photo with missing pixels→in painted photo

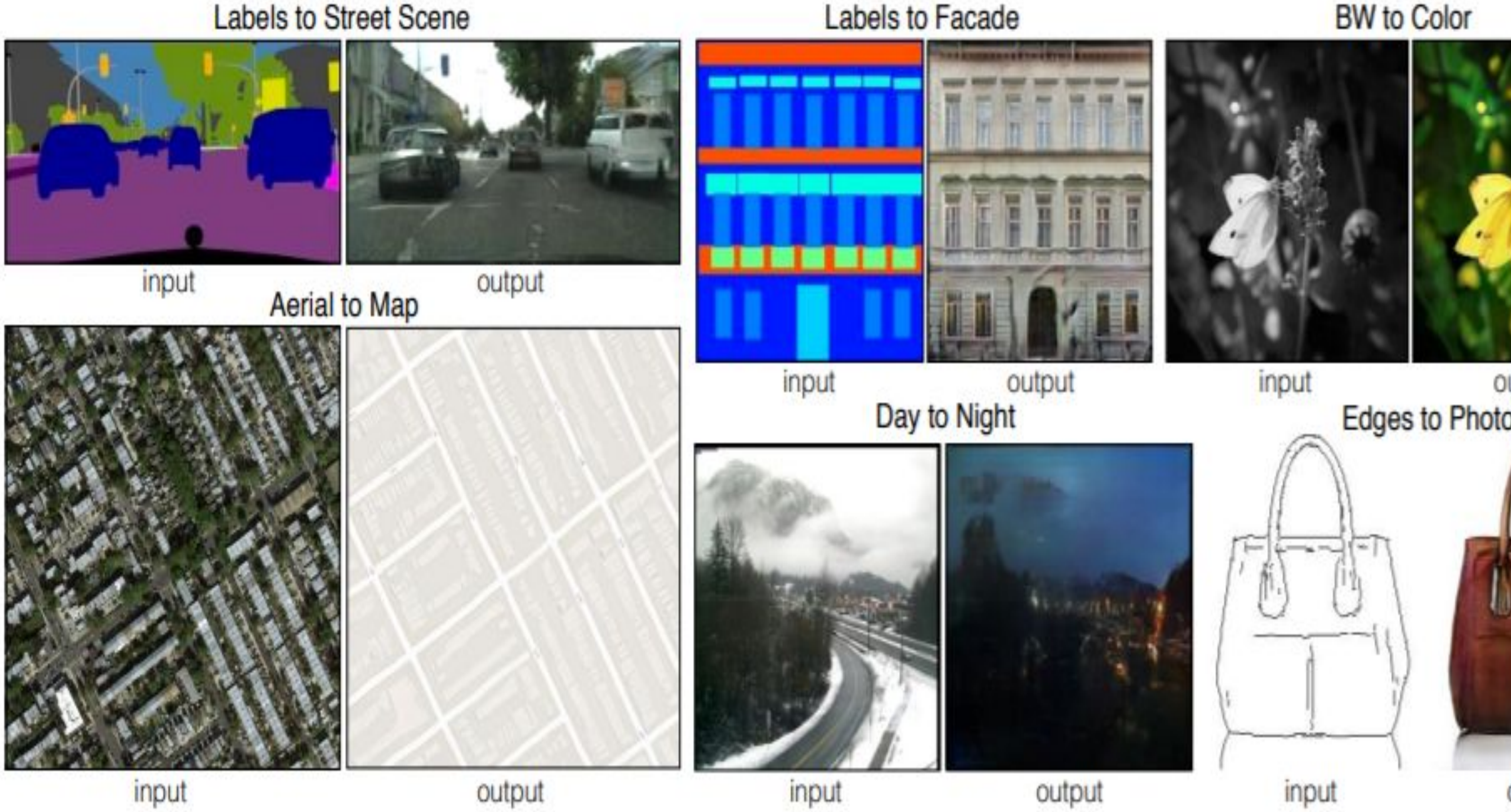
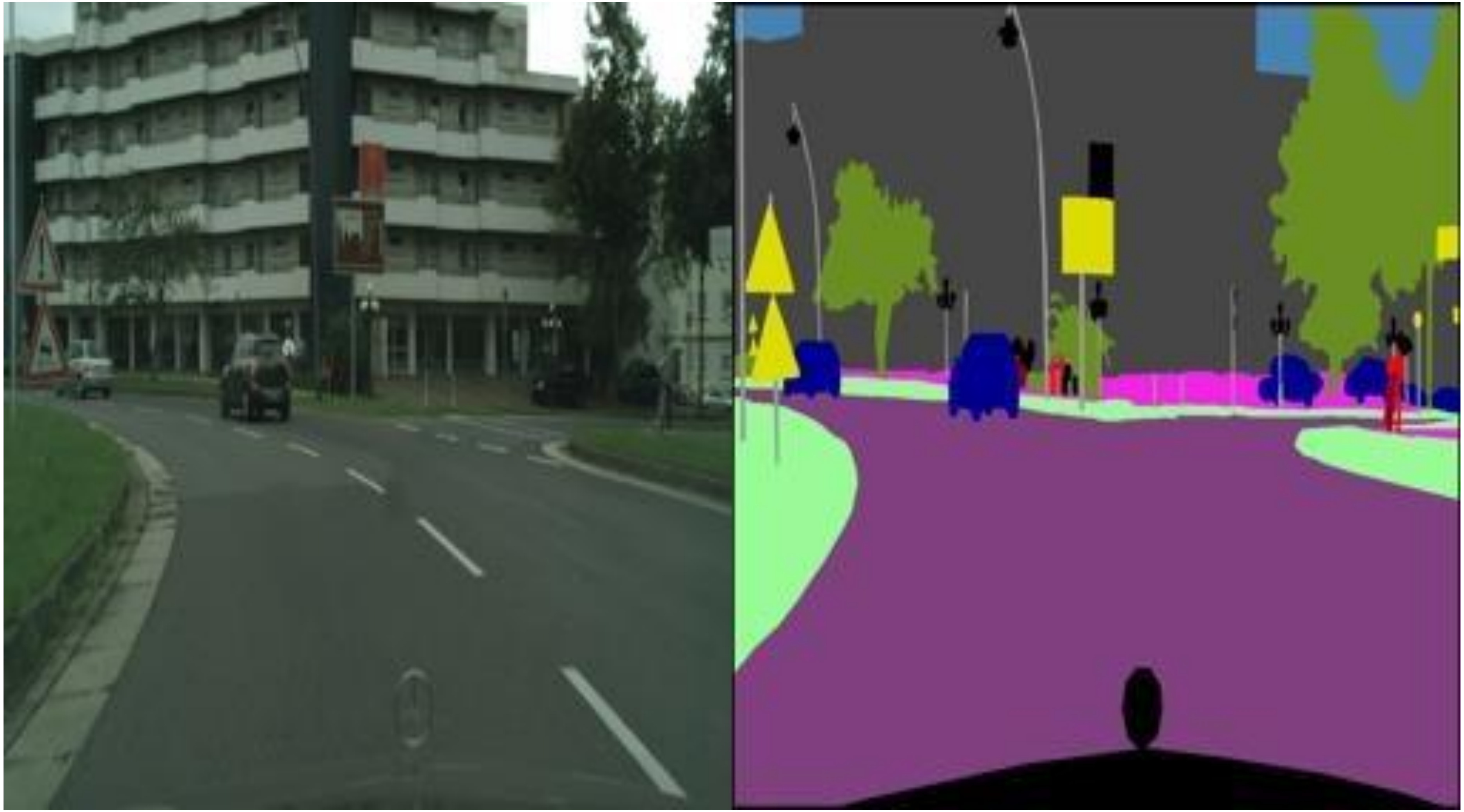

Labels to Street Scene
input
output
Aerial to Map
input
output
Labels to Facade
input
output
Day to Night
input
output
BW to Color
input
ou
Edges to Photo
input

Photo segmentation Task and Cityscapes dataset

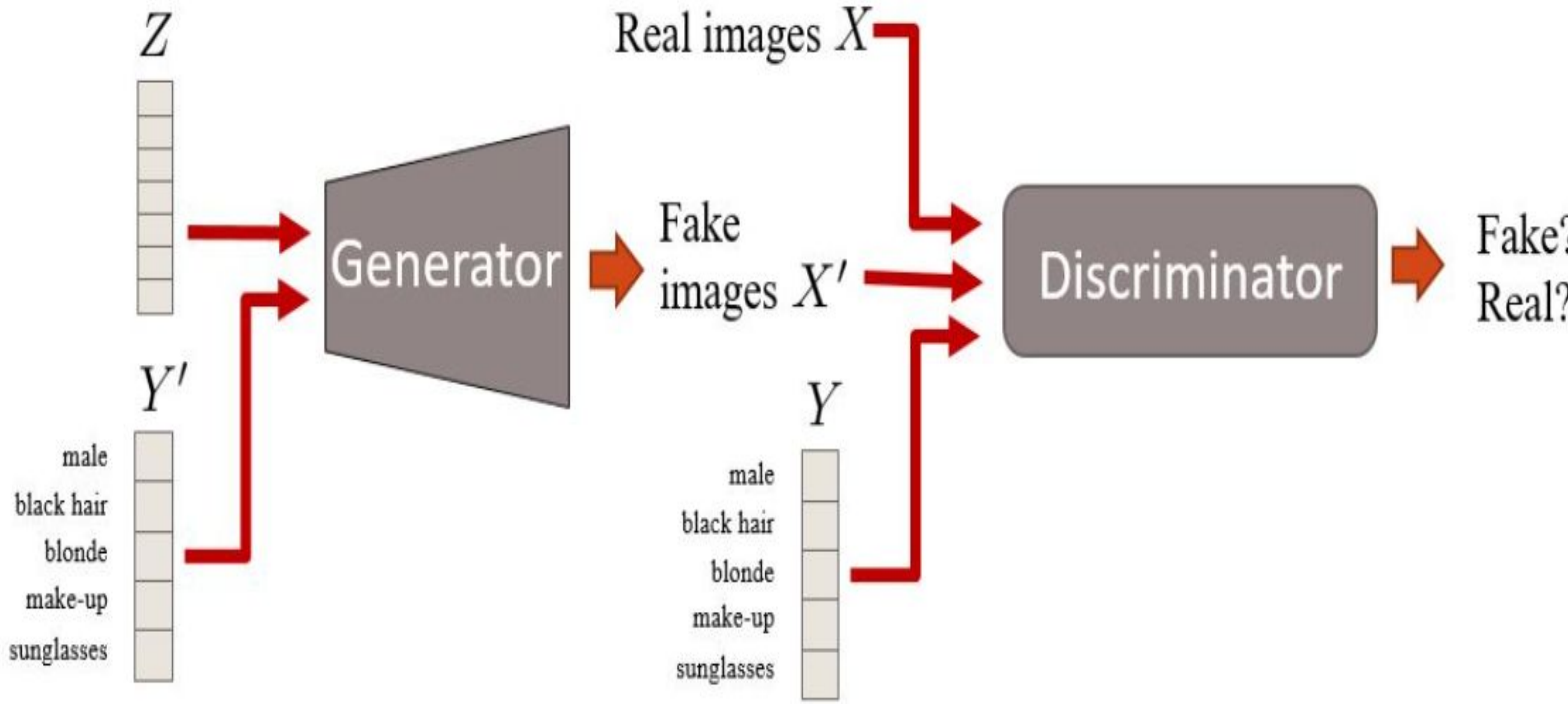
Cityscapes data (dataset home page) contains labeled videos taken from vehicles driven in Germany. This version is a processed subsample created as part of the Pix2Pix paper. The dataset has still images from the original videos, and the semantic segmentation labels are shown in images alongside the original image. This is one of the best datasets around for semantic segmentation tasks.

This dataset has 2975 training images files and 500 validation image files. Each image file is 256x512 pixels, and each file is a composite with the original photo on the left half of the image, alongside the labeled image (output of semantic segmentation) on the right half.



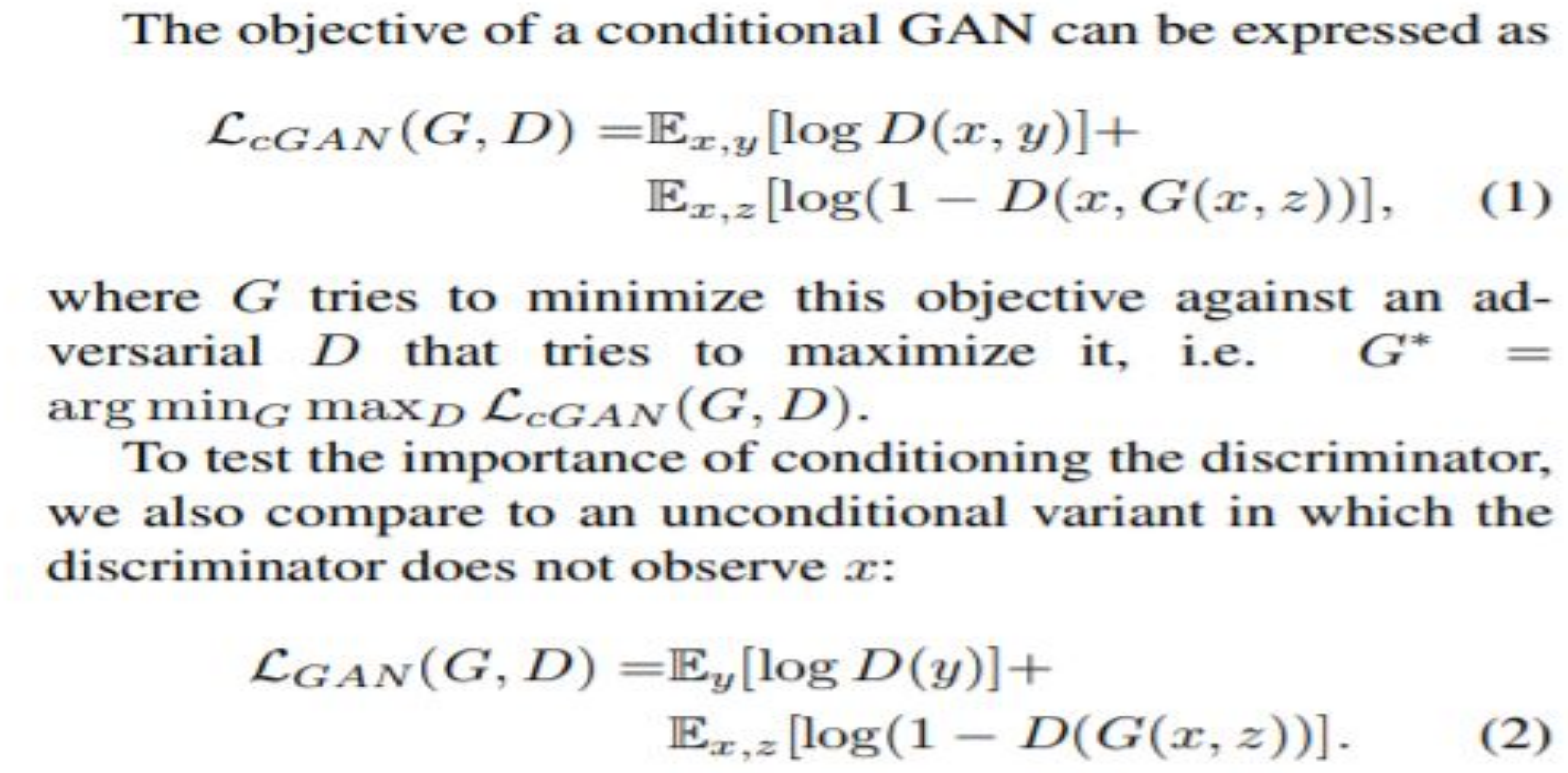
Why CGANS?

- Learn the mapping from input image to output image and a loss function to train this mapping.
- Generic approach to problems that traditionally would require very different loss formulations
- Wide applicability and ease of adoption without the need for parameter tweaking. As a community, we no longer hand-engineer our mapping functions, and this work suggests we can achieve reasonable results without hand-engineering our loss functions either



Overview of a conditional GAN with face attributes information.

Objective function



The objective of a conditional GAN can be expressed as

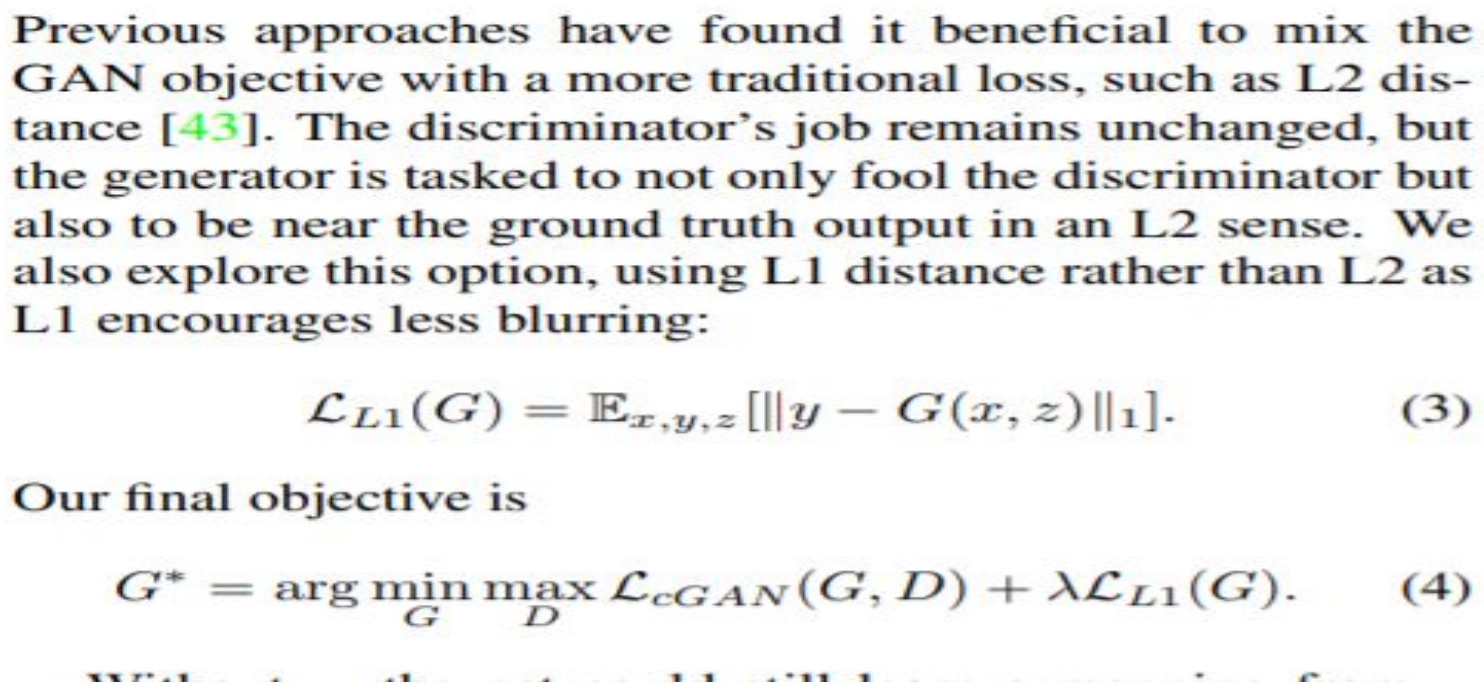
$$\begin{aligned}\mathcal{L}_{cGAN}(G, D) =& \mathbb{E}_{x,y}[\log D(x, y)] + \\ & \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))], \end{aligned} \tag{1}$$

where G tries to minimize this objective against an adversarial D that tries to maximize it, i.e. $G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D)$.

To test the importance of conditioning the discriminator, we also compare to an unconditional variant in which the discriminator does not observe x:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D) =& \mathbb{E}_{y}[\log D(y)] + \\ & \mathbb{E}_{x,z}[\log(1 - D(G(x, z)))]. \end{aligned} \tag{2}$$

Objective function



Previous approaches have found it beneficial to mix the GAN objective with a more traditional loss, such as L2 distance [43]. The discriminator's job remains unchanged, but the generator is tasked to not only fool the discriminator but also to be near the ground truth output in an L2 sense. We also explore this option, using L1 distance rather than L2 as L1 encourages less blurring:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1]. \tag{3}$$

Our final objective is

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G). \tag{4}$$

General framework and Network architectures used

Generator: UNET

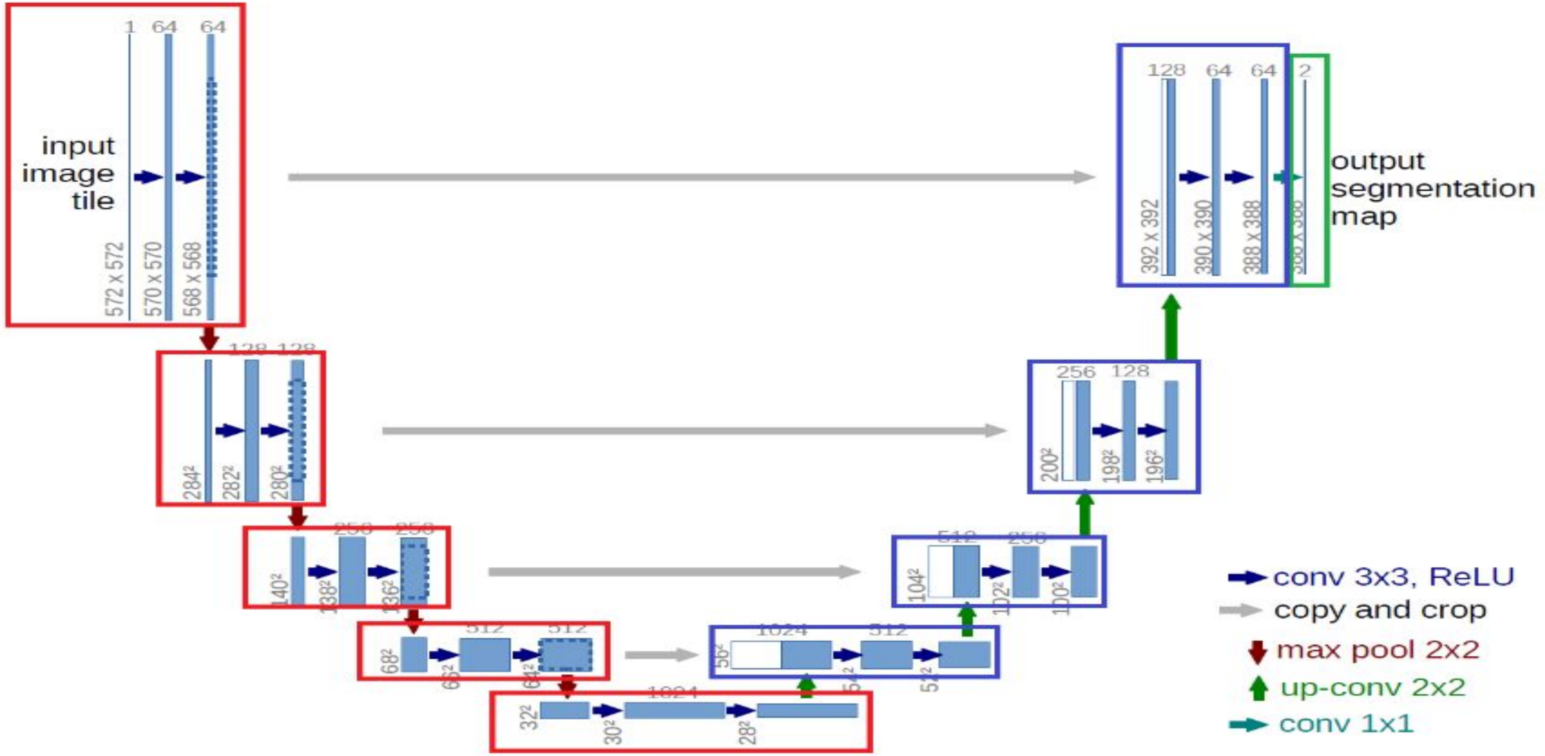
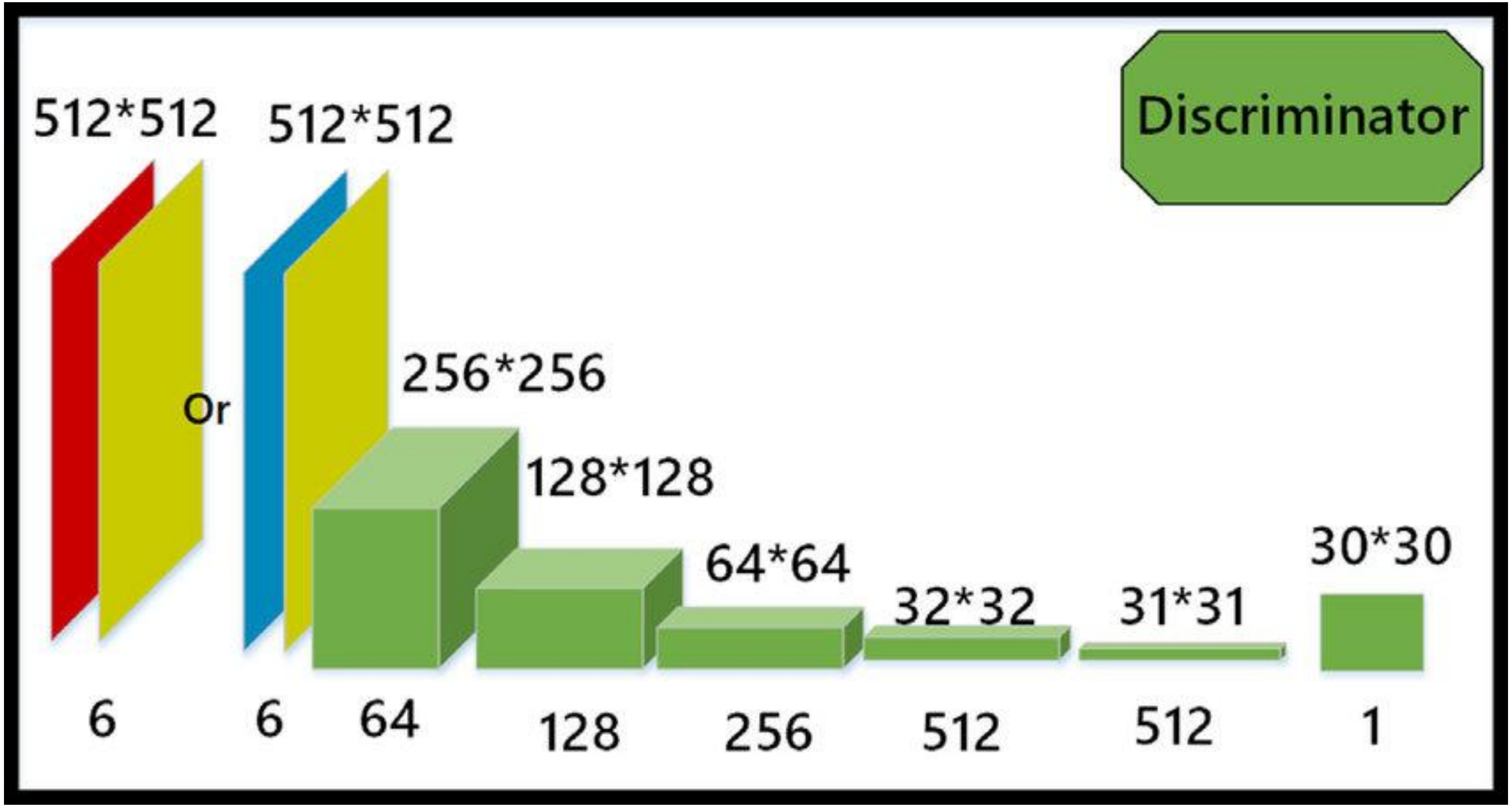


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Discriminator: Markovian PatchGan Network


Discriminator
512*512
512*512
Or
256*256
128*128
64*64
32*32
31*31
30*30
6
6
64
128
256
512
512
1

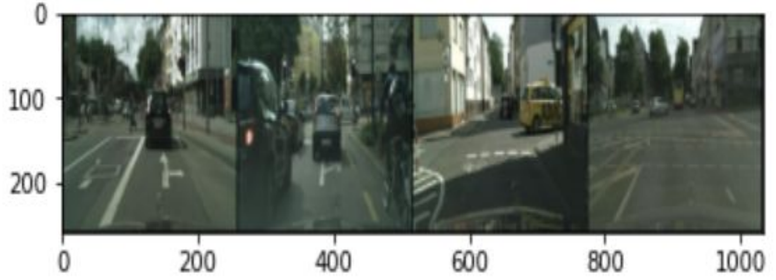
Optimization and Inference

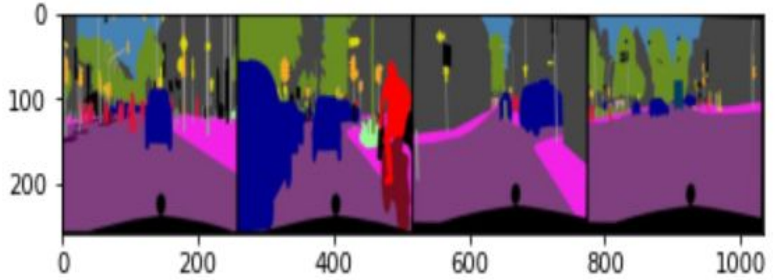
To optimize our networks, we follow the standard approach from [24]: we alternate between one gradient descent step on D, then one step on G. As suggested in the original GAN paper, rather than training G to minimize $\log(1 - D(x, G(x, z)))$, we instead train to maximize $\log D(x, G(x, z))$ [24]. In addition, we divide the objective by 2 while optimizing D, which slows down the rate at which D learns relative to G. We use mini batch SGD and apply the Adam solver [32], with a learning rate of 0.0002, and momentum parameters $\beta1 = 0.5$, $\beta2 = 0.999$. At inference time, we run the generator net in exactly the same manner as during the training phase. This differs from the usual protocol in that we apply dropout at test time, and we apply batch normalization [29] using the statistics of the test batch, rather than aggregated statistics of the training batch. This approach to batch normalization, when the batch size is set to 1, has been termed “instance normalization” and has been demonstrated to be effective at image generation tasks [54]. In our experiments, we use batch sizes between 1 and 10 depending on the experiment.

Training on the cityscapes dataset

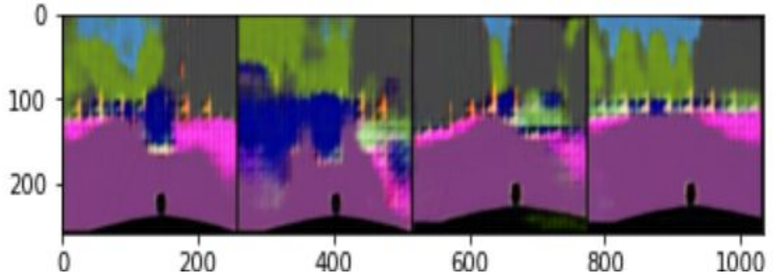
Epoch 1

```
Epoch 1: Step 1600: Generator (U-Net) loss: 17.461983327865596, Discriminator loss: 0.41389626394957296
```



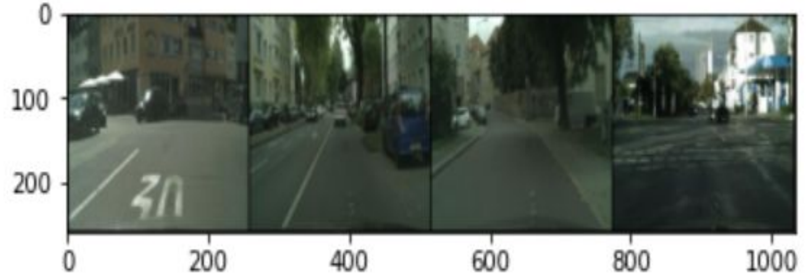


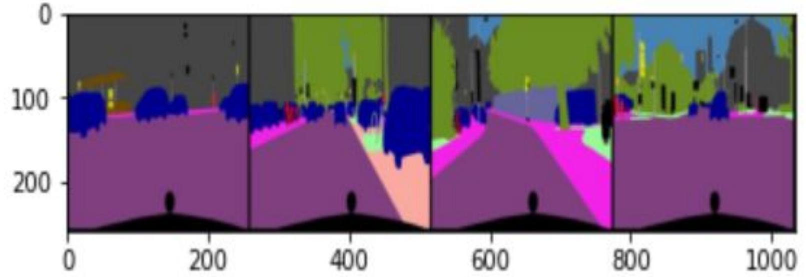
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



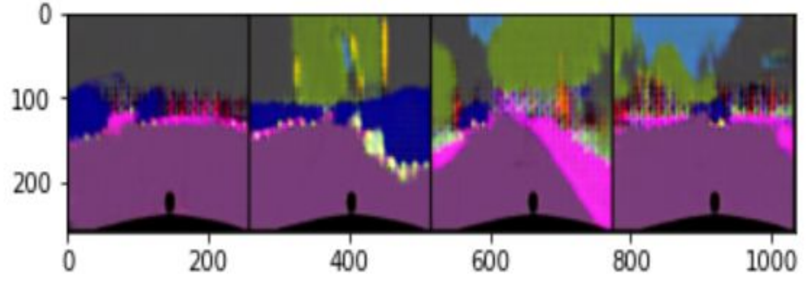
Epoch 2

```
Epoch 2: Step 2600: Generator (U-Net) loss: 15.990100579261785, Discriminator loss: 0.44308156318962555
```



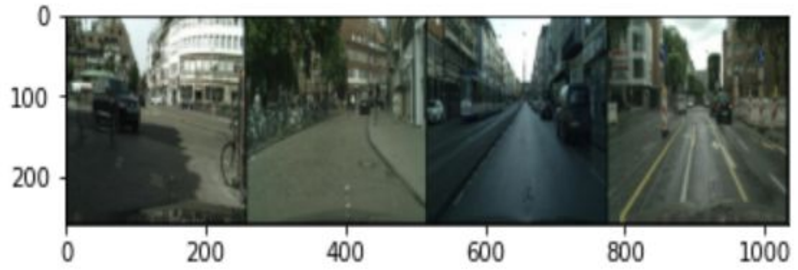


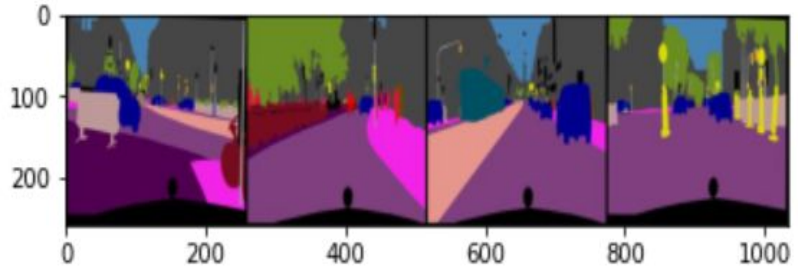
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



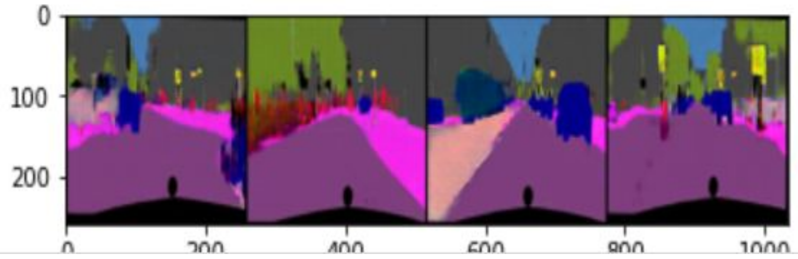
Epoch 18

```
Epoch 18: Step 16400: Generator (U-Net) loss: 11.583383512496955, Discriminator loss: 0.3228769456967712
```



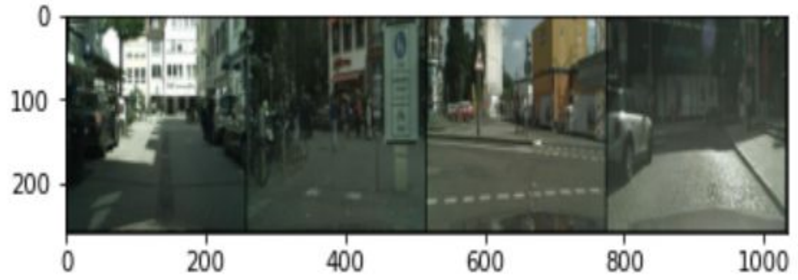


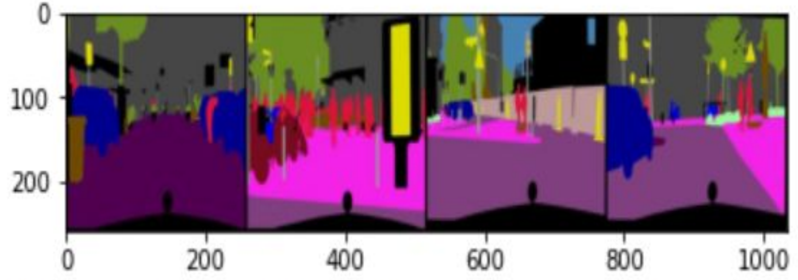
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



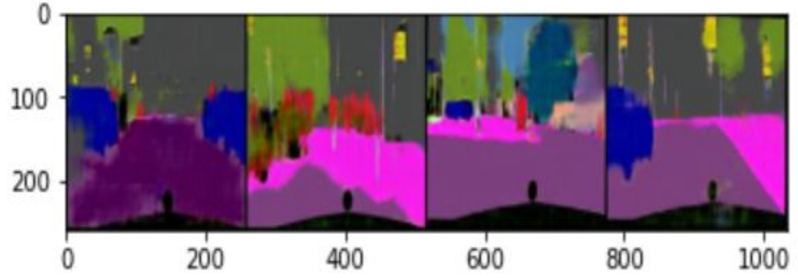
Epoch 19

```
Epoch 19: Step 17200: Generator (U-Net) loss: 11.372778725624077, Discriminator loss: 0.3719861767441034
```





```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



Observations

Going further