pAInting with GANs

Andreas Engberg[1]

Abstract
This paper covers a modified implementation of NVIDIA's spatially adaptive generative adversarial network, GauGAN. The network consists of an encoder, generator, and discriminator, all working together to produce photorealistic images given a noise vector and segmentation mask. The network was trained on parts of the COCO-Stuff dataset with a smaller batch size and fewer training images compared to the original implementation. Despite the modifications the network can produce semi-realistic images with some artifacts. We compare the results of synthesized images against NVIDIA's results and conclude that we can achieve improved results by using a larger training dataset on fewer epochs. Combining an instance mask and segmentation mask provides higher quality images, and finally, training the network on multiple GPUs allows for a larger batch size which can further improve synthesized results as we get better statistics over the training data. An interactive application was implemented where you can draw a segmentation mask by hand and generate realistic images in real time.

Source code: `https://github.com/engbergandreas/GauGAN`

Demo: `https://engbergandreas-gaugan.streamlit.app/`

Authors
[1] *Media Technology Student at Linköping University, anden561@student.liu.se*

Keywords: Machine learning — GAN — SPADE — Synthesize images

Contents

1. Introduction

Automatic content generation has become a hot topic in recent years with the success of generative networks such as GauGAN [1], Pix2Pix [2], VQ-GAN [3], DALL-E [4] and Imagen [5] common in most recent networks is some form of conditional image synthesis, it refers to the task of generating photorealistic images conditioning on certain input data.

NVIDIA released GauGAN in 2019 and has had great success in generating photorealistic images of landscapes with multi modal synthesis [1]. Furthermore, they can produce high quality images in diverse settings using a network trained on COCO-Stuff and ADE20K datasets. The aim of this project is to implement GauGAN and evaluate how well it can synthesize photorealistic images by training a network on COCO-Stuff dataset using limited computing resources. Moreover, we explore what happens if you feed the network an image of objects in different shapes, e.g., a zebra-like giraffe. In addition, an interactive application[1] was created where the network transforms semantic doodles into realistic images in real time.

GauGAN is able to generate higher quality realistic images compared to Pix2Pix due to a new normalization layer using a spatially adaptive normalization technique which preserve segmentation information better than common normalization layers. As they tend to wash away semantic information when applied to uniform and flat segmentation masks [1].

[1] `https://github.com/engbergandreas/GauGAN`

2. Theory

This chapter covers background, concepts and related theory on Generative adversarial network and techniques used during training.

2.1 Generative Adversarial Network

Generative adversarial network (GAN) is a type of unsupervised learning network. In general, unsupervised learning can be described as the process of learning something useful from a given dataset containing unlabelled input data. Generative models draw training examples $\mathbf{x}$ from an unknown distribution $p_{data}(\mathbf{x})$. The goal is to learn a model $p_{model}(\mathbf{x})$ that approximates $p_{data}(\mathbf{x})$ as close as possible [6]. This process involves learning (or solving) an explicit density function, for deep generative neural networks the corresponding density function becomes complex and hard to handle. Instead of designing a tractable density function, GANs learn a manageable sample generation process, such models are called *implicit generative models*.

GANs can be seen as a game between two neural networks playing against each other. One network is the generator which implicitly define $p_{model}(\mathbf{x})$, it is able to draw samples from this distribution. In reality the generator is defined by a distribution $p(z)$ and learnable parameters $\theta^{(G)}$. Typically $p(z)$ is a Gaussian distribution and z is an input vector of random noise in a high dimensional space [6]. The generators goal is to learn a function $G(z;\theta^{(G)})$ that transform unstructured noise into realistic samples. The other network is called the discriminator (adversarial) which examines samples $\mathbf{x}$ and output some estimate $D(\mathbf{x};\theta^{(D)})$ whether the sample is real, i.e. drawn from the training data distribution or if it is a fake sample generated by the generator.

Each network (player) has a cost $C^{(G)}(\theta^{(G)},\theta^{(D)})$ and $C^{(D)}(\theta^{(G)},\theta^{(D)})$ for the generator and discriminator respectively. Each player seeks to minimize this cost; however the optimization becomes nontrivial as the cost depends not only on its own parameters but also the other player's parameters in which it does not have access to. The goal is therefore to find the Nash equilibrium[2]; a point where each player cost is at a local minimum with respect to that player's parameters. At a high level, the discriminator cost encourages it to classify data correctly as either real or fake, whereas the generator cost pushes the generator to generate samples that fool the discriminator to incorrectly classify samples as real.

2.2 Loss

In a classification network where labelled training data is available we can usually use the L1 or L2 loss applied with some SoftMax function to determine the probability of a certain output. However, in GANs the problem is that there is not one correct answer on whether a generated sample is realistic or not. Computing the difference between a sample and some ground truth will result in a large error if for example there are illuminance differences, e.g., day and night. Moreover, suppose a blue car is generated but ground truth is composed of a red car, a distance metric like squared error will be non-zero even if the spatial arrangement of the blue car is correct, i.e. headlights, windscreen, tires etc are placed correctly.

We need a loss function that essentially determines, 'does this image look real?'. GauGAN utilize several loss functions to train the network such as prediction loss, feature matching loss, Kullback-Leibler divergence loss and hinge loss. Details of the different loss functions are presented in section 3.1.

2.3 One-Hot Encode

Most machine learning algorithms are optimized to use numerical data as input and output. Datasets in image classification often provide segmentation maps as categorical color data, e.g., road = pink, person = turquoise, car = blue or as integer data, e.g., road = 3, person = 54, car = 13. This type of representation allows the model to assume a natural ordering between classes which can result in poor performance. Thus, the data needs to be transformed to a format that can be used by the algorithms. One-Hot Encode (OHE) transforms the categorical data or integer labels where there is no ordinal relationship into a binary representation. Given the integer label example, OHE forms column vectors for each class and the rows represent input data where one indicates it belongs to a given class and zero as not belonging.

$$\begin{pmatrix} road & person & car \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

OHE is extended to 2D matrices where classes are stacked in the third dimension as channels, an example of OHE applied on an input segmentation map is shown in Figure 1.

Figure 1. Left: Integer labeled segmentation map, middle: OHE channel wise, right: Stack channels onto a single tensor.

3. Method

The GauGAN architecture consists of several networks: an encoder, a generator, and a discriminator. A segmentation map and a corresponding real image or noise vector is fed through the network. The encoder encodes the real image into a high dimensional noise vector, the noise vector is then passed to the generator together with the segmentation map which generate a photorealistic image, the real image (or noise

[2] https://en.wikipedia.org/wiki/Nash_equilibrium

vector) acts as the overall style of the output image. Multi-modal synthesis can be achieved by feeding different real images to the network using the same segmentation mask. During training the generated image and real image is fed to the discriminator to determine whether the synthesized image is realistic.

It is possible to combine the segmentation map with an instance map that helps separate overlapping classes into distinct instances. Something that NVIDIA implemented in their own network but was not used in this one.

3.1 Architecture

In this section an overview of the different networks that GauGAN consists of is given, furthermore, an introduction to spatial adaptive normalization is provided below.

3.1.1 Spatially Adaptive Normalization

Spatially adaptive normalization (SPADE) is a conditional normalization technique, conditional meaning that it is dependent on external data, i.e., the segmentation map. SPADE normalization is similar to batch normalization, the activation is normalized channel wise and then modulated with a learned scale and bias [7]. In contrast to batch normalization, SPADE is spatially dependent, this means that $\gamma_{y,x}$ and $\beta_{y,x}$ are tensors instead of vectors and produce learned parameters at every pixel.

If we simplify the original equation of SPADE [1] by only looking at the activation of a specific layer with a single channel and a batch size of one, the activation becomes

$$\gamma_{y,x}(\mathbf{m})\frac{h_{y,x}-\mu}{\sigma}+\beta_{y,x}(\mathbf{m}) \tag{1}$$

where $h_{y,x}$ is the input to SPADE layer, μ and σ are the mean and standard deviation of the activation, this correspond to batch normalization seen in Figure 2. $\gamma_{y,x}(\mathbf{m})$ and $\beta_{y,x}(\mathbf{m})$ are the learned modulation parameters that depend on the input segmentation map $\mathbf{m}$. These functions are implemented as a two-layer convolutional network.

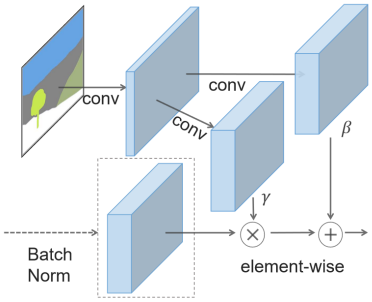

Figure 2. SPADE normalization, the mask is convolved to produce modulation parameters γ and β, (Park 2019).

3.1.2 Generator

The generator is a fully convolutional decoder which consist of several SPADE ResNet blocks followed by upsampling layers as shown in Figure 3. The residual (skip connection) is learned for layers that have different number of channels before and after the residual block. Although not shown in the figure, each convolution is followed by spectral normalization [8]. The input to the generator is a latent noise vector z in a 256-dimensional space and a segmentation map (semantic mask) which is then passed to the SPADE normalization layers. The semantic mask is interpolated to match the spatial resolution at each residual block as they operate at different scales using nearest neighbour downsampling.

The generator is trained using multiple loss functions, GAN loss is the expectation over discriminator predictions and can be seen as a realness score over the generated images, it is defined as:

$$L_G = -\mathbb{E}_{z\sim p_z, y\sim p_{data}}[D(G(z),y)] \tag{2}$$

Feature matching loss motivate the generator to produce images that not only fool the generator but instead ensure that the images have similar statistics as the real image data distribution. Feature matching penalize the L1 distance between discriminator feature maps of real images to feature maps of fake images computed at different scales. Feature maps are extracted from multiple layers of the discriminator, these intermediate representations help stabilize the training [2]. Feature matching loss is defined as:

$$L_{FM}(G,D) = \mathbb{E}\sum_{i=1}^{T}[||D^{(i)}(s,x) - D^{(i)}(s,G(z))||_1] \tag{3}$$

Where T is number of feature maps in the discriminator, $D^{(i)}$ represent the i:th layer of the discriminator and s is the segmentation mask.

VGG loss is similar to feature matching loss; however, instead of using the discriminator's feature maps, we use a VGG-19 pre-trained on ImageNet to compute the feature maps. Again, we penalize the L1 distance between feature maps of real and fake images [9].

$$L_{VGG}(G,D) = \mathbb{E}\sum_{i=1}^{5} C_i[||VGG(x)_i - VGG(G(z))_i||_1] \tag{4}$$

where $C = \{\frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}, 1\}$ and $VGG(x)_i$ is the i:th feature map given input image x. If we train the network with an encoder we get the total loss for the generator as the weighted sum of (2), (3), (4) and (7) as:

$$L^{(G)} = L_G + 0.05 * L_{KLD} + 10 * (L_{FM} + L_{VGG}) \tag{5}$$

where L_{KLD} is the encoder loss given in equation (7).

3.1.3 Encoder

As previously mentioned, the generator takes a random noise vector as its input, this vector can be generated at random or by an encoder. If we add an encoder to the GauGAN it

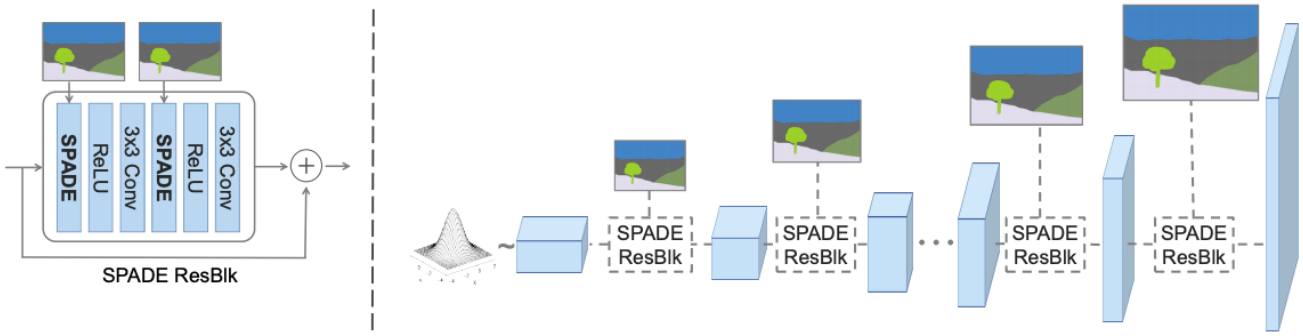

Figure 3. The generator consists of several SPADE ResNet blocks *(left)*, with upsampling layers between each residual block *(right)*, notice the input to the generator is a high dimensional latent noise vector, (Park 2019).

can be seen as a VAE (variational auto encoder) where the generator acts as the decoder. The encoder tries to capture the overall style of a real image and can be used during inferences as a guided style transform. The encoder consists of several convolutional layers that successively "shrink" an image into a compact latent representation to generate mean and variance vectors that is used to compute a noise input vector to the generator, to compute the noise vector we use a reparameterization trick [10] and calculate it as:

$$z = e^{\sigma * 0.5} \cdot \varepsilon + \mu \tag{6}$$

where $\cdot$ refers to elementwise multiplication, σ and μ is the variance and mean vectors generated by the encoder and ε is an auxiliary variable, usually some normal distribution random noise [10].

If we add the encoder to the network, we also include the Kullback–Leibler divergence (KLD) loss, which acts as a regularization for the encoder. This loss penalizes the KL divergence between the distribution predicted by our encoder and a zero mean Gaussian. Without this loss the encoder could just assign a random vector for each training sample rather than actually learning a distribution that captures modalities in the training dataset [9]. KLD loss is defined as:

$$L_{KLD} = D_{kl}(q(z|x)||p(z)) \tag{7}$$

$q(z|x)$ is called the variational distribution from which we draw random vector z given real image x and $p(z)$ is the standard Gaussian distribution.

3.1.4 Discriminator

The discriminator is a patch-based fully convolutional network. In the original paper Park et al. [1] use a multi-scale discriminator to discriminate an image at different resolutions. This is done to differentiate high-resolution real and synthesized images, and give the discriminator a larger receptive field without requiring a deeper network or larger convolutional kernels, both of which can lead to overfitting [1],[2]. Since we use smaller, fixed-size input images, we choose to only use one discriminator to simplify the implementation of the network.

The discriminator receives a channel-wise concatenation of the semantic mask and corresponding synthesized image or ground truth image, it tries to classify them as either real or fake. Both fake and real images are fed to the discriminator all at once to avoid disparity statistics [1]. It is trained using the Hinge loss [11]

$$\begin{aligned} L_D = &-\mathbb{E}_{(x,y)\sim P_{data}}[min(0, D(x,y)-1)] \\ &-\mathbb{E}_{z\sim p_z, y\sim p_{data}}[min(0, -D(G(z),y)-1)] \end{aligned} \tag{8}$$

3.2 Dataset & Training

The network was trained using COCO-Stuff dataset [12], it contains 118K semantic training images and 5000 validation images, it covers 172 classes captured in diverse scenes. Due to limited training time and computing power the 5000 validation images were chosen as training data and samples from the original training data was chosen to evaluate and test the network. It was trained for 165 epochs using a RTX 3070 Ti, because the GPU only has 8 GB of VRAM a rather small batch size of 4 images per batch was chosen. The Adam optimizer was used with beta values 0.0 and 0.999, the learning rate was set to 2^{-4} and 4^{-4} for the generator and discriminator respectively during the first 100 epochs, it was then halved for the remaining 65 epochs. PyTorch default initialization was used to initialize the network.

3.3 Interactive App

An interactive application was written in Python and hosted on *streamlit*[3]. It allows a user to draw semantic doodles in an interactive canvas and generate a photorealistic image in real time. For ease of use and for better visualization each class in COCO-Stuff is mapped to a random color in RGB space, furthermore, the number of classes is limited to 10 to make the interface more readable.
You can test the app for yourself here.

4. Result

In this section we present images generated by the network. The result of successfully synthesized images are shown in

[3] https://streamlit.io/

Figure 4; Figure 5 illustrate where the network failed to produce qualitative realistic images. Moreover a side by side comparison to NVIDIA's synthesized images can be seen in Figure 7. However, the test images NVIDIA used to synthesize was used as training data in this project, this means that the synthesized images are biased but it shows an interesting result nonetheless. Figure 6 shows a hand drawn outdoor scene using the interactive app. The image depicts a patch of grass with a tree on a beach by the water and a mountain range in the distance. Figure 8 and 9 show additional synthesized results on the COCO-Stuff dataset.

Figure 4. Successfully synthesized images on COCO-Stuff test data.

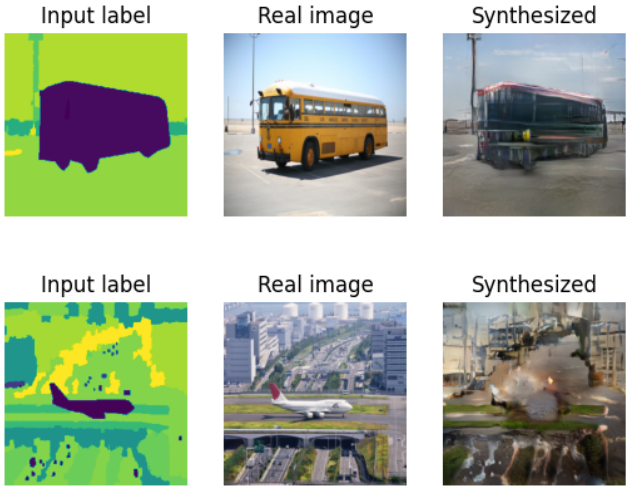

Figure 5. Failed synthesized images on COCO-Stuff test data.

5. Discussion

As we can see in Figure 4 and 8, the network produces somewhat realistic images in most cases. The overall gist is in many

Figure 6. Synthesized image painted in the interactive app.

cases quite accurate, but there are some critical problems. First of all, the network has problems generating realistic details, if we zoom in on one of the images we can clearly see that details are missing and it becomes obvious that the image is synthesized, see Figure 4a where the elephant in the middle is blurred and lack a lot of the details compared to the real image (middle). Secondly the network has trouble producing spatially coherent structured objects as shown in Figure 5a. There is no real front and back of the bus, and it also lacks important details such as window, headlights, and blinkers. Lastly, Figure 5b shows some of the recurring visual artifacts with broken structure and textures, similar textural artifacts can also be seen in 6.

A possible explanation for these artifacts could be the limited training data used. The dataset contains many classes but since only 5000 images were used during training, it is possible that the model cannot generalize properly. Another problem is the small batch size used as this produce worse statistic over training data compared to a larger batch size. But it could also be as simple as overfitting, e.g., the generator has learned to compress data into one small area in order to fool the discriminator which would produce these noisy textures and artifacts.

Comparing hand drawn segmentation mask to pre-generated ones, we can see coarser artifacts in the synthesized image, see Figure 9. The free drawn segmentation mask give rise to a whole new challenge for the network. It must be able to interpret any possible combination of classes in any given arrangement which may or may not lay outside the p_{model} distribution. This can negatively affect the synthesized image and could also explain the artifacts in Figure 6.

As noted in 3.2 and 3.1.4, NVIDIA trained a GauGAN on the full COCO-Stuff dataset using a multi-scale discriminator. The multi-scale discriminator gives a coarse to fine detail evaluation, and is said to generate more detailed images as well as produce globally consistent images [2]. In Figure 7 we can see that NVIDIA is able to produce significantly more detailed images, with better spatial coherency and textures compared to our own network. The larger dataset, larger batch size and multi-scale discriminator is believed to be the reason for the improved results. In addition, the use of instance maps significantly improves the synthesized result. We can see how this affect images, see for example the zebras in Figure

7, NVIDIA's network can separate the two zebras giving a higher quality image with more details compared to our own network. It should also be noted that NVIDIA's generated images also suffer from artifacts and strange output at times, however, they are less severe and frequent compared to the trained network, see for example the bus in Figure 7.

6. Conclusion

Despite modifications made to the network, it can generate somewhat photorealistic images. Overall, the image quality is adequate and at a quick glance they seem realistic enough. However, when you zoom in or start analysing the image, it becomes clear that objects have subpar textures and sometimes lack important details. The hardest part seems to be generating structural coherent objects, such as determining where the head should be on an animal or human and producing a distinct front and back of a car.

Given a smaller training dataset and batch size limited to one GPU, it is possible to train a GauGAN to produce semi-realistic images, albeit with some visual artifacts. The synthesized results can be further improved upon by incorporating a multi-scale discriminator. Using a larger batch size which will produce better statistics over the training data and improve generalization. Furthermore, we can conclude that increasing the training data is more important than training for more epochs as shown by the improved results of NVIDIA's network in Figure 7. Additionally, instance maps should be used during training if available, as this helps the network differentiate objects apart and produce more meaningful results.

Multi-modal synthesis was excluded from the result because it did not produce the same effect on COCO-Stuff dataset compared to the Landscape dataset that NVIDA trained on. One reason could be the large number of classes in COCO-Stuff compared to the landscape dataset and the fact that fewer images were used during training. This leads to fewer images per class in COCO-Stuff and impairs generalization. As the encoder learns the whole of an image, it will most likely pick up background parts of an image and generalize about those features. In the case of landscape images, this may correspond to day and night or weather features [1]. But for COCO-Stuff there is no clear connection between the images or global features to generalize, hence the effect is less clear.

References

[1] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2337–2346, 2019.

[2] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[3] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

[4] OpenAI. Dall-e 2. https://openai.com/dall-e-2/, 2021. Accesed: 2022-11-01.

[5] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[8] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[9] Ayoosh Kathuria. Understanding gaugan part 1: Unraveling nvidia's landscape painting gans. https://tinyurl.com/y6t2kypj, 2019. Accessed: 2022-10-26.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR, 2019.

[12] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Cocostuff: Thing and stuff classes in context. In *Computer vision and pattern recognition (CVPR), 2018 IEEE conference on*. IEEE, 2018.

7. Additional result

Figure 7 show additional synthesized images comparing NVIDIA's result to that of our own.

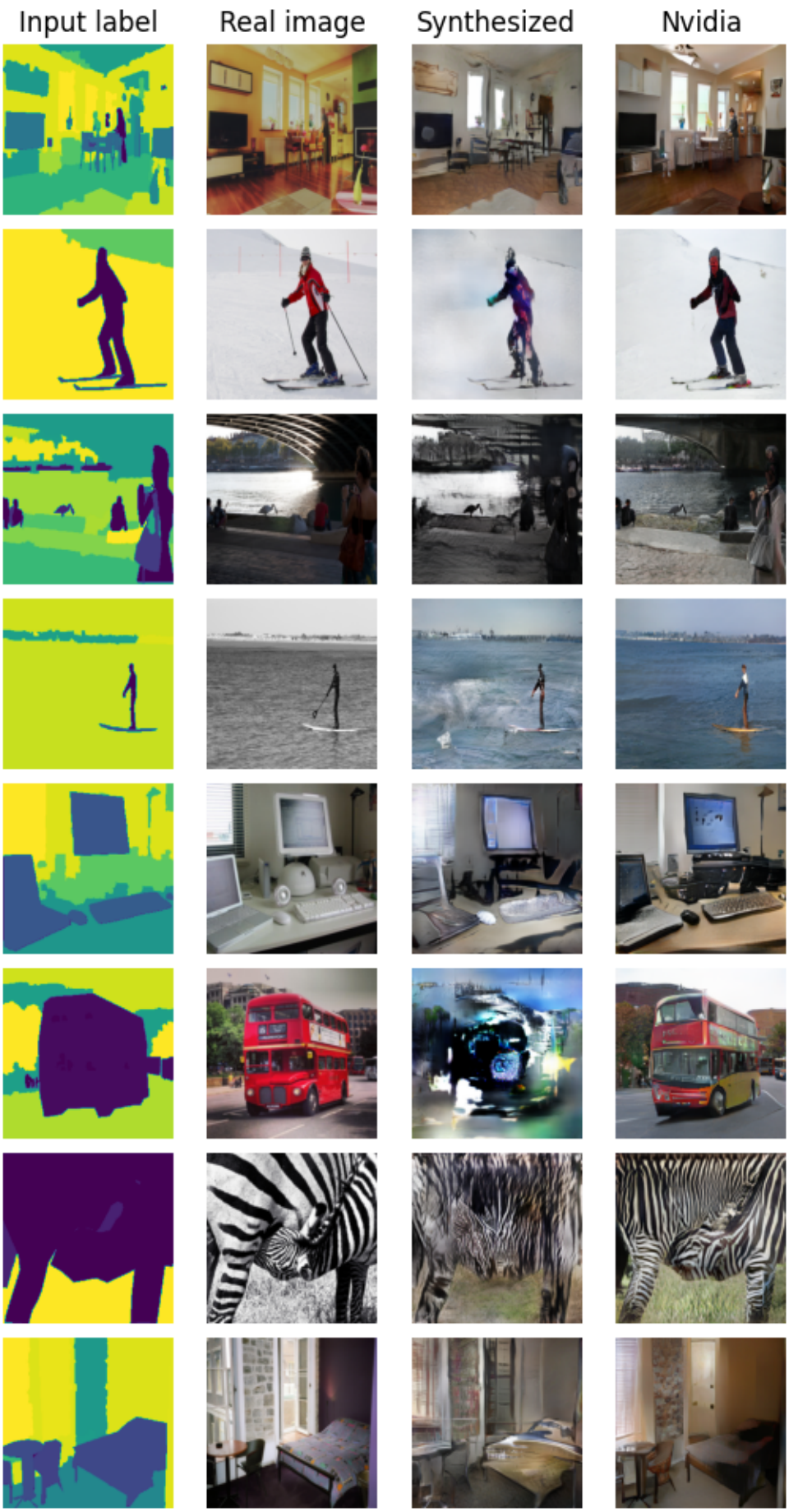

Figure 7. Comparison between implemented network and NVIDIA's network, note the input images have been used during training in our own network.

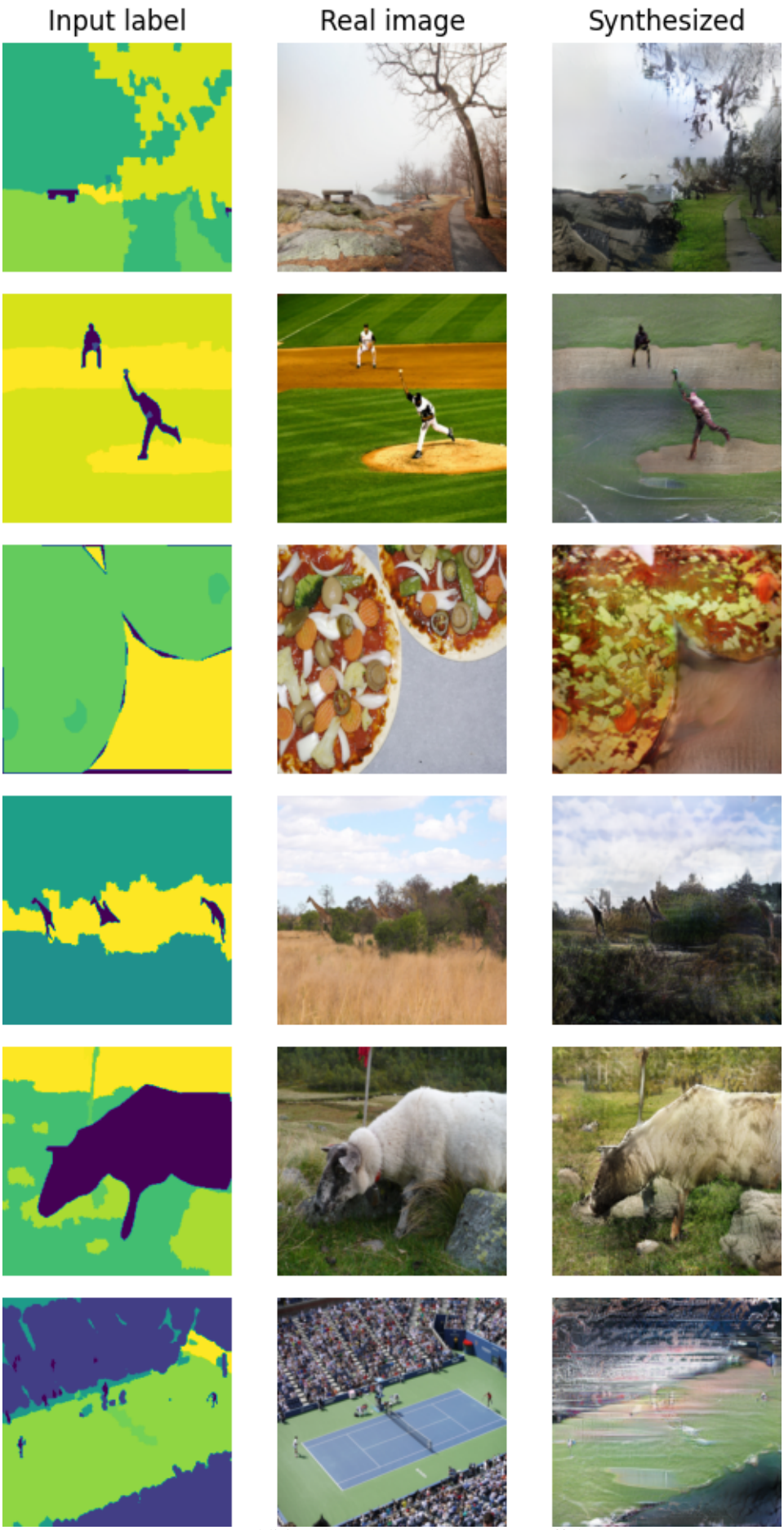

Figure 8. Additional results on the COCO-Stuff dataset.

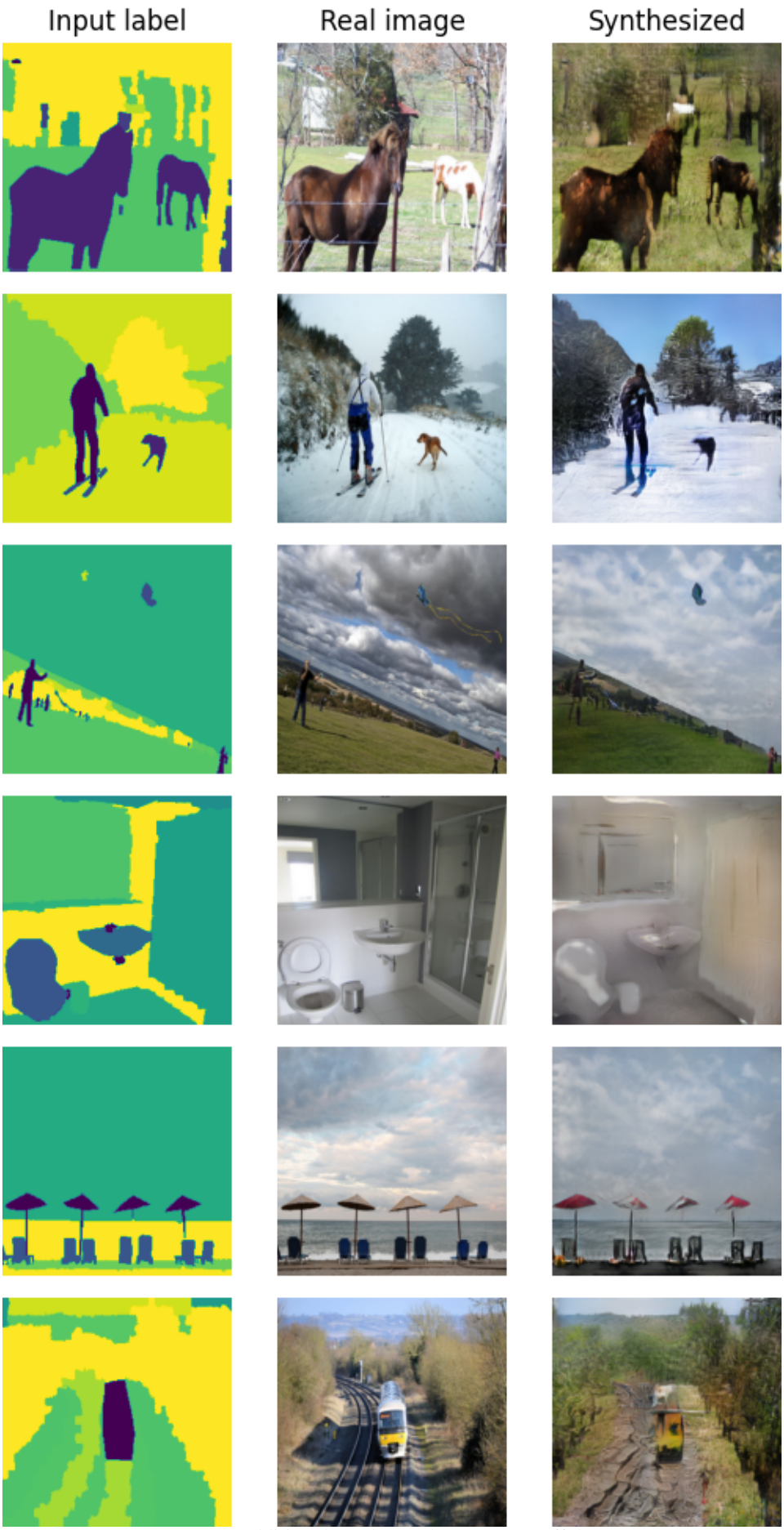

Figure 8. Additional results on the COCO-Stuff dataset.

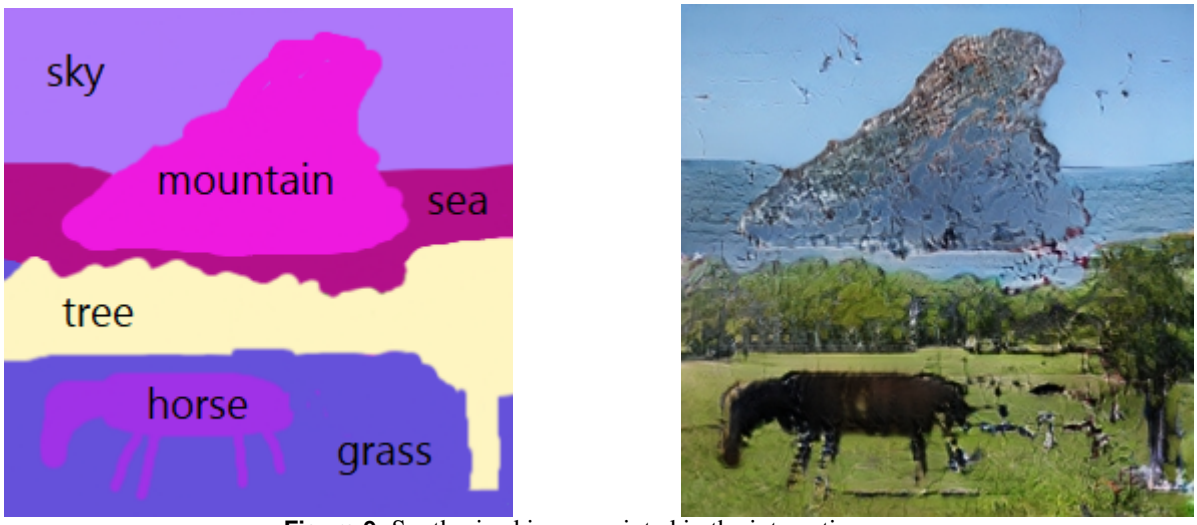

Figure 9. Synthesized image painted in the interactive app.