EE599 Deep Learning, Final Project

October 30, 2021

Photorealistic Image Synthesis using Spatially-Adaptive Normalization

©Charu Singh, Harsha Raidurgam Venkat, Sourabh Tirodkar

Abstract

In this project, we would be generating photorealistic images by generating segmentation maps for a set of images using Spatially Adaptive Normalization (SPADE). Other methods proposed involve normalization that tend to wash away the semantic information. In this project we have solved this problem by replacing the Instance Norm with SPADE, a learned transformation. The advantage of the proposed method is being demonstrated on Places 365 dataset. Implementation primarily involves 2 tasks. Firstly, we plan to generate the segmentation maps. The color codes will be defined for each class. The second task is feeding the semantic segmentation maps to the generator of the GANs and original images to the discriminator which will generate a photorealistic image. Here we will use the normalization technique known as Spatially Adaptive Normalization (SPADE) which preserves the semantic information against common normalization layers. The outcome would be a photorealistic image

Contents

1 Introduction

1.1 Motivation

Imagine that a kid wants to learn how to paint and start drawing on iPad or any other painting tool,he would start with a basic scene, for example mountains, trees, river, clouds. But the result produced is not photorealistic. So, to enhance the learning ability, the understanding, and the visualization of what is being drawn is very important. Now if there would be a software which would convert your painting to some photorealistic scenery as if it is a photograph. So now you can customize your painting in the way you like. Basically, we want to make the neural network model to understand how to realistically render certain objects it has been trained on. In the figure shown, whenever it sees a "dark green" pixel in the input image, it will render grass. When it sees "light blue" pixels in the image, it will render road because that is what it has been trained to do. So How does the network learn to do that? We provided thousands of training images and it is corresponding semantic segmentation mask with accurate labels for each pixel. The labels are the category to which the object in the image belong. But it is not enough to just train what labels each segment represent, it also needs to learn to put some realistic style (color, texture, lighting) onto the segment. So, it must learn what color sky is, what kind of textures a tree can render. When we provide thousands of such diverse images with its mask, the neural network can be trained to generate new pixels which will look almost exactly like the desired label. Generative Adversarial Networks (GANs) are used for this task.

Figure 1: Image Segmentation

1.2 Literature Survey

To synthesize the images, methods studied are Generative Adversarial Networks (GAN's) [4] and variational autoencoder [9]. In this project we are using conditional GAN's to perform conditional image synthesis task. A GAN architecture consists of a Generator and Discriminator where the generator aims to produce the realistic images so that it can fool the generator and it fails to determine the generated image as fake. In conditional GAN's the image synthesis depends on the type of input data. Given the category labels, class conditional models [2, 11, 12, 13, 14] learn to synthesize images. Also, image to image translation is based on conditional GAN [21] where input and output both are images. It has been observed that learning based methods run faster and produce more realistic results than non-parametric methods [5]. In our project we are segmentation mask as an input which will get converted to photorealistic images. We generated the segmentation masks for the dataset we are using. We used spatially adaptive normalization technique on our own dataset which would result in compact network and better results as compared to leading methods. Unconditional normalization does not depend on the external data. In modern deep networks use of unconditional normalization layers by various classifiers have been

an crucial aspect, which includes the Local Response Normalization in the AlexNet[10] and Batch Normalization in the Inception-v2 network[8].Other types of normalization are Instance Norm[17], layer normalization[1], the Group Normalization[20], and the weight Normalization[16]. However, conditional normalization such as Conditional Batch Normalization (Conditional BatchNorm) [6] and Adaptive Instance Normalization (AdaIN) [3] are initially used to transfer the style and eventually being used in vision applications [6]. While performing style transfer tasks [7], the affine parameters control the global style of the output and therefore are uniform across spatial coordinates. In our project we used SPADE normalization techniques which applies a spatially varies affine transformation, thus making it suitable for image synthesis form segmentation masks. Image super resolution was proposed by Wang et al. [19] which is also built on spatially adaptive modulation layers conditioned upon sematic inputs. But the difference is their goal was to incorporate the semantic information into super resolution but in our project our goal is to design the generator so that it can produce different styles and semantics disentanglement. We focus on providing the semantic information in the context of modulating normalized activations. We use semantic maps in different scales, which enables coarse-to-fine generation.

2 Image Segmentation

2.1 Datasources

We chose the Places365 dataset. It has a very diverse set of categories and it has different type of images from clear to blurry and more objects in one image. These features make it an ideal choice for training with so much variation in data for effective training of the model. In total, Places contains more than 10 million images comprising 400+ unique scene categories. The dataset features 5000 to 30,000 training images per class, consistent with real-world frequencies of occurrence. Among this diverse set of images, limited by time and resources, we chose to select 60,000 images from 12 categories which mainly includes landscapes such as cliffs, fields, ocean, coast etc...

1. **Training Dataset**: Selected 12 categories with 5000 each and considered 60,000 images for the training dataset.

2. **Test Dataset**: Considered 949 random images as our test dataset.

Data Pre-Processing: The default image size was 1024x712. All images were resized to 256x256 as the GAN would require input images to be of the size 256x256. This reduced addtional pre processing overhead to the GAN.

This dataset contained only raw images. For training the GAN, we had to generate segmentation maps and convert them to annotated label maps as the input along with original images. The models and methods used for generating different maps will be discussed in next section.

Figure 2: Original Image, Segmentation map, Annotation or Label map

2.1.1 Semantic Segmentation

The objective of semantic segmentation is to assign a category label to each pixel present in the image. We can identify different objects present in the image. The scene parsing algorithm predicts the shape, location and label of all the objects present in the image. There are different architectures available for us to use scene parsing. Notable ones are U-Net, Mask RCNN, PSPNet, etc. They use a common architecture called autoencoders. A basic autoencoder network consists of two networks connected in series. First is the encoder network and second is the decoder network. It is described in figure [3]

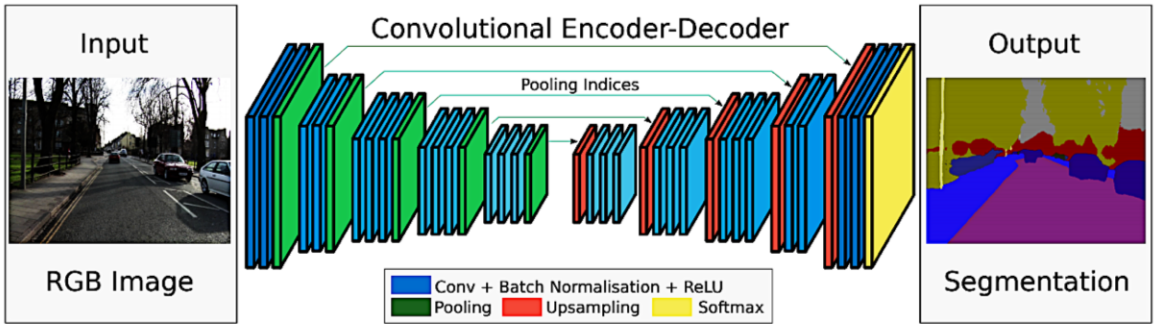


Figure 3: CSAILVision Segmentation Architecture (Autoencoder)

Encoder: An encoder network transforms the high dimensional spatial information of an image to a low dimensional representation of the same image. An encoder essentially removes noise and extracts useful features from the image. It consists of multi-layer convolutional blocks which helps to extract best features. Dimension reduction is obtained by pooling layer and convolution operation. In our case, we used a pre-trained Resnet50Dialated as the encoder.

Decoder: A decoder does the inverse operation of that of an encoder network. A decoder tries to reconstruct the original image form its representation given by the encoder network. Like encoder, it has repetitive blocks consisting of Up sampling layer and convolutional layer. We train the decoder network in our project to perform semantic segmentation from the reduced representation obtained from the encoder network. There is a cross connection between the encoder network and the decoder network which provides pooling indices to perform up sampling while retaining all the important information.

In the architecture CSAILVision team used PSPNet [22]. The PSPNet introduces a block in between the encoder and decoder network called as pyramid pooling module.

Pyramid Pooling Module: It is described in figure [4] Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d)

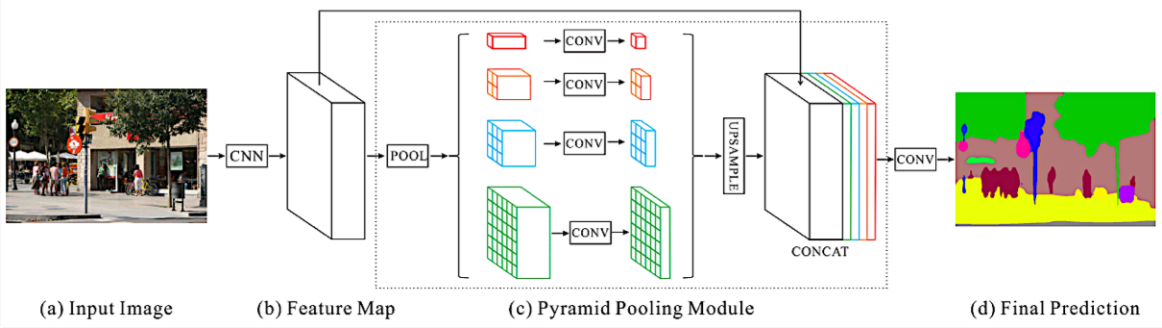


Figure 4: Pyramid Pooling Module

2.1.2 Annotation Maps

SPADE requires one-channel images. So the RGB or 3 channel semantic maps are again converted to gray-scale images as required for the SPADE input. This is done by masking all the three channels and replacing each pixel value with a class index. So in the annotated map or label map each pixel represents the index of the semantic class it belongs to. Class indices are continuous integers from 0 to n-1 where n is the number of classes.

Figure 5: Semantic Map

Figure 6: Class Annotated map or Label Map

3 Generative Adversarial Network (GAN)

3.1 Introduction

GAN allow us to generate images which are indistinguishable from original training images. During the training of GAN it can learn to distinguish between the real and the generated images. It will synthesize randomly sampled images from a manifold mapped with latent features of its training set. It mainly has two components:

1. **Generator**: They will synthesize an image from random pixels.
2. **Discriminator**: It learns to classify the generated image as "real" which is the image coming from the dataset or "fake" that is the generated image.

When the training starts, both Generator and Discriminator are bad at their jobs. But as the training proceeds, when the Discriminator classifies a generated image as "real", it is wrong and is punished using back-propagation. Discriminator will minimize the loss by calculating the incorrect classifications across a batch. The job of Generator is to fool the Discriminator, by generating images which appear to be coming from the training dataset. Using specific advanced loss functions, Discriminator will guide the Generator to generate specific kinds of images. Here we are using conditional GAN which would determine the data to be generated upon the input we provide. Instead of giving random pixels to the generator we will give semantic segmentation map and the GAN will generate a real image conditioned upon the input image

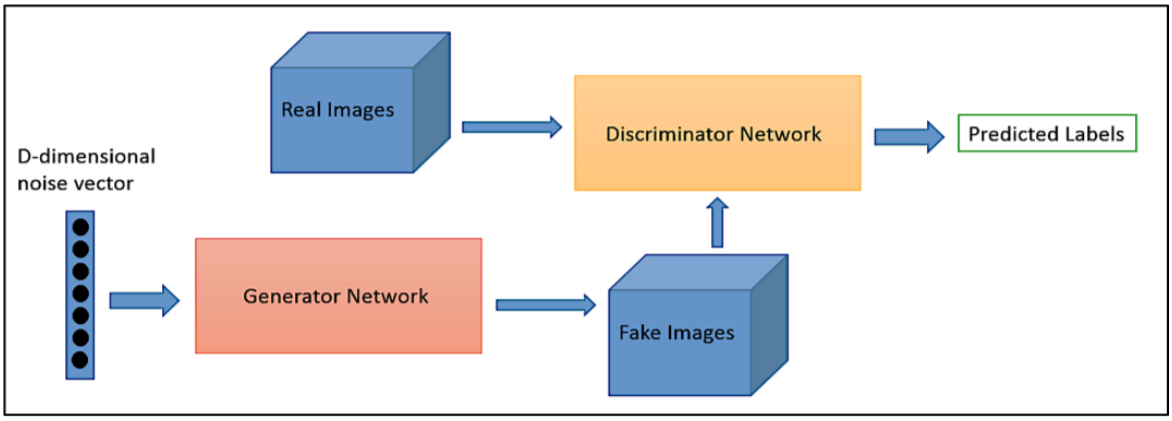


Figure 7: Simple GAN Architecture

3.2 Existing Architectures and its challenges

In existing architecture such as Pix2PixHD [18], the parameters of Instance Norm layer are non-learnable that is (γ set to 1 and β set to 0).

Why we need Normalization?

A learned convolutional feature map can activate matching filters based an input image. But it tends to overfit to the training sample and can activate highly on minor artifacts in the image, which are not important for a given task. So, they do not work well in real worlds applications with diverse, never-before-seen data. So, for the purpose of generalization, the learned filters are normalized that

is by getting the mean of the set of features and its standard deviation. Normalization helps in capturing important information across the training set and the neurons do not activate at high, unstable levels. More technically, we do this by increasing the capacity of an excited neuron to subdue its neighbors. We basically want a significant peak so that we have a form of local maxima.

$$h^{(l)}_{n,c,y,x} = \gamma^{(l)}_{n,c,y,x} \frac{h^{(l)}_{n,c,y,x} - \mu^{(l)}}{\sigma^{(l)}_c} + \beta^{(l)}_c$$

Batch Normalization:

During training, normalization is done for each layer's inputs by using the mean and standard deviation of the values in the current batch. The overall idea is that even when the exact values of inputs to hidden layers change, their mean and standard deviation will still almost remain same thus reducing the covariate shift. This weakens the coupling between parameters of early layer and that of later layers hence, allowing each layer of the network to learn by itself i.e., more independent of each other. This has the effect of speeding up the learning process. Hence, batch normalization ensures that the inputs to the hidden layers are normalized, where the normalization mean and standard deviation are controlled by two parameters, γ and β.

Instance Normalization:

It is subset of BatchNorm that normalizes features within each channel of the feature map over the mini batch.

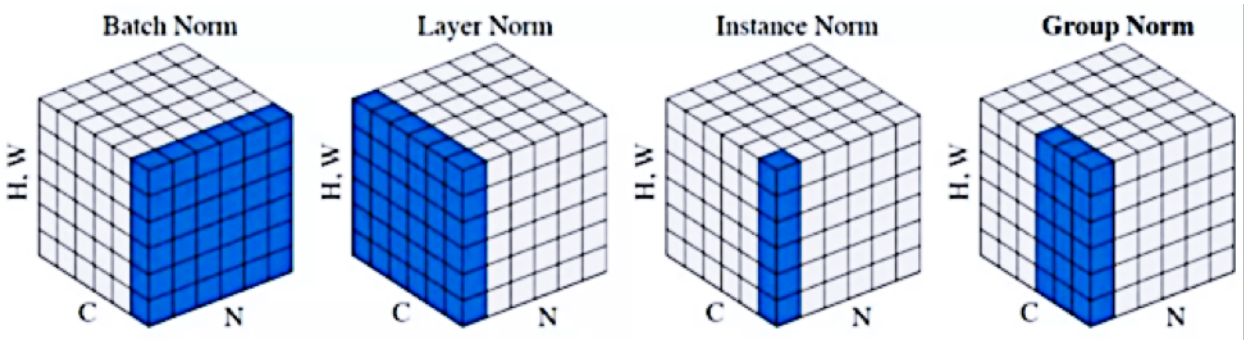
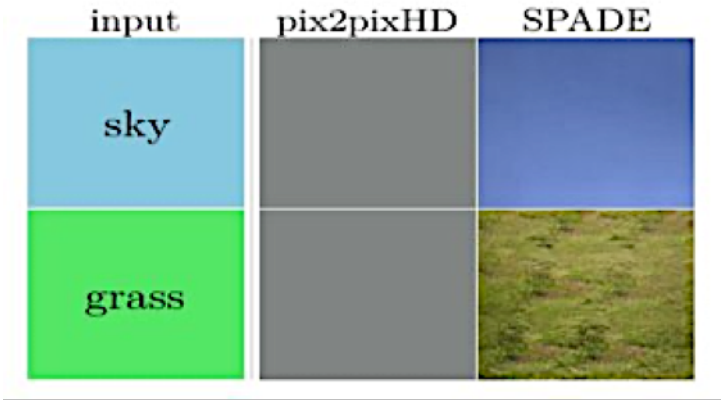


Figure 8: Types of Normalization

Disadvantage of Batch Normalization:

In the extreme case of a completely single -label image like in the figure shown below which contains only one class that is either sky or grass, the output is all zeros. Now after applying convolution to the segmentation mask and then the normalization will cause the convolution output to be uniform with different labels having different uniform values. Now after applying instance norm to the output, the normalized activation will become all zeros no matter what the input semantic label is given. Therefore, semantic information is totally lost.

According to the equations given below,both are almost same except for a fact that the information lost while performing batch normalization can be recovered in SPADE adding gamma and beta. In the next slide we are going to see how this is done.



4 SPatially Adaptive DEnormalization (SPADE)

4.1 Why Spade?

With SPADE[15] the idea is to prevent the network from losing semantic information by allowing the segmentation map to control the normalization parameters and locally in each layer. Rather than having just one pair of parameters for each channel, different pairs are computed for each spatial point by feeding a downsampled version of the segmentation map through 2 convolutional layers. So, the hyper-parameters are itself a neural network which can now be trained with any condition you want. For this algorithm, the condition is to preserve the semantic information using maps.

$$h^{(l)}_{n,c,y,x} = \gamma^{(l)}_{n,c,y,x} \frac{h^{(l)}_{n,c,y,x} - \mu^{(l)}}{\sigma^{(l)}_c} + \beta^{(l)}_c$$

$$h^{(l)}_{n,c,y,x} = \gamma^{(l)}_{n,c,y,x}(s) \frac{h^{(l)}_{n,c,y,x} - \mu^{(l)}}{\sigma^{(l)}_c} + \beta^{(l)}_c(s)$$

$h^{(l)}_{n,c,y,x}$ - activation at site before normalization,
$\mu^{(l)}$ - mean of the activation channel while
$\sigma^{(l)}_c$ - standard deviation of the activation channel
$\gamma^{(l)}_{n,c,y,x}(s)$ and $\beta^{(l)}_c(s)$ - learned modulation parameters of the normalization layer

The very first reason is that the input to SPADE is a semantic map, which is further one-hot encoded. This means that the GAN must take regions of uniform values, precisely 1s, and produce pixels with diverse values so that they look like a real object. Having a different set of batch norm parameters for each pixel helps in tackling this task better than single set of batch norm parameters for each channel in the feature map.

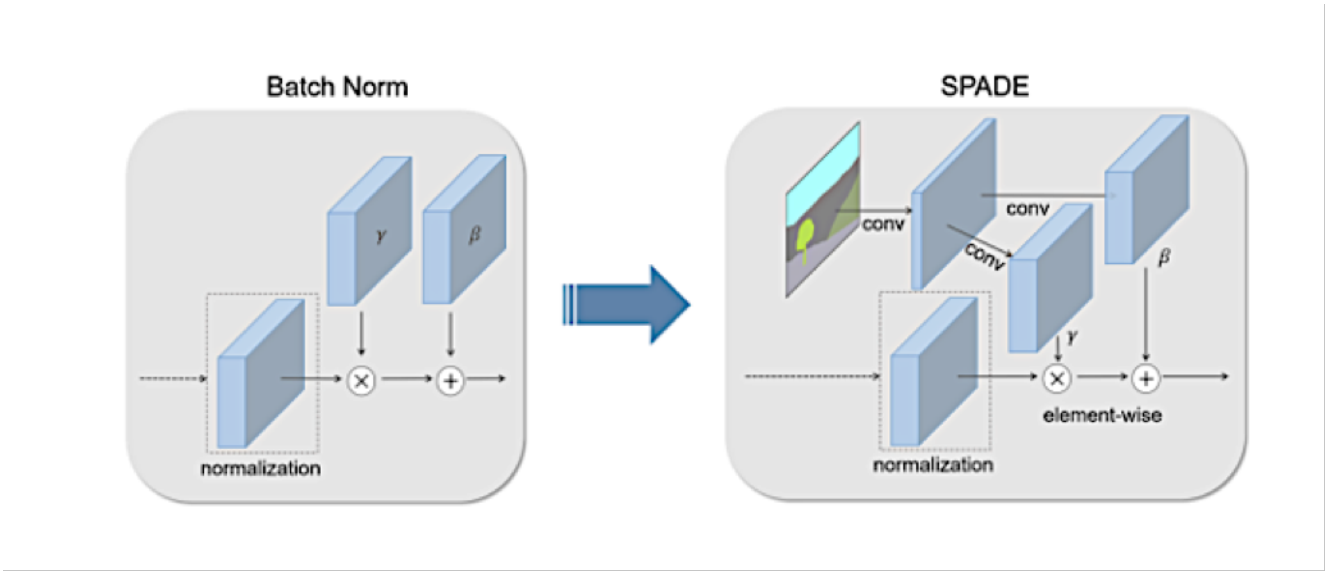


Figure 9: Batchnorm and SPADE

5 Architectures

5.1 Generator

The generator of this conditional GAN is a fully convolutional decoder consisting of various residual SPADE blocks. It also has an upsampling layer after every SPADE ResBlks. The upsampling after SPADE is done using the nearest neighbor resizing rather than the transpose convolutional layer as in PIx2PixHD[40]. Transpose convolutional layers have been susceptible to producing artifacts in the output image. Since then the upsampling has been done using non-learnable upsampling and then followed by a convolutional layer. Here we are not doing downsampling to build up the semantic information as we are directly providing semantic maps as an input to each SPADE block. The inputs to this SPADE Generator are the labels maps, which are a single channel semantic map and a random vector which is the output from the image encoder. The goal of this architecture is to generate 3 channel images which would serve as an input to the discriminator where the classification of real and fake would be done.

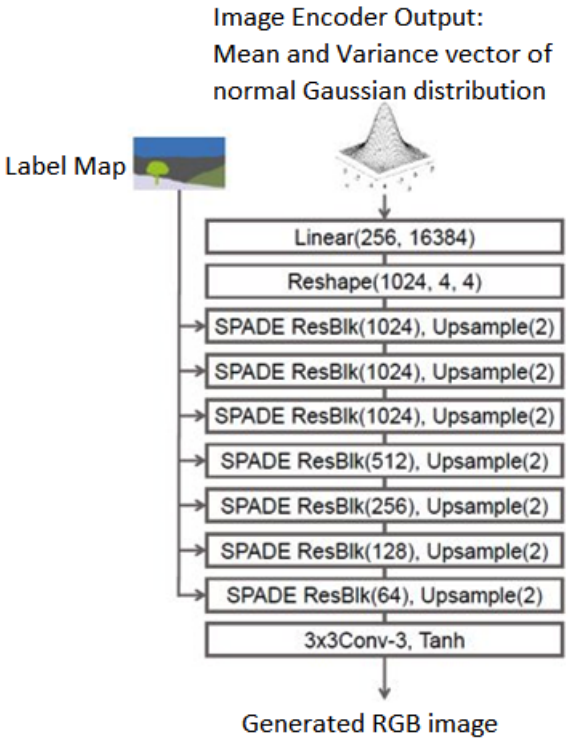


Figure 10: Generator architecture

5.2 Discriminator

It is a classifier network usually having fully connected layers at the output to predict real/ fake images. Here, the discriminator is similar to Patch GAN where it outputs a feature map which is the reason we have a convolutional layer at the end. Multi scale patch GAN is used which outputs a feature map which is then averaged to get the score which will show how real the image is. Since it consists pf FCN layers, it is size invariant.

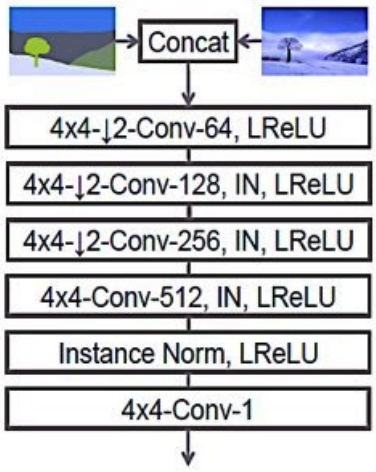


Figure 11: Discriminator Layers

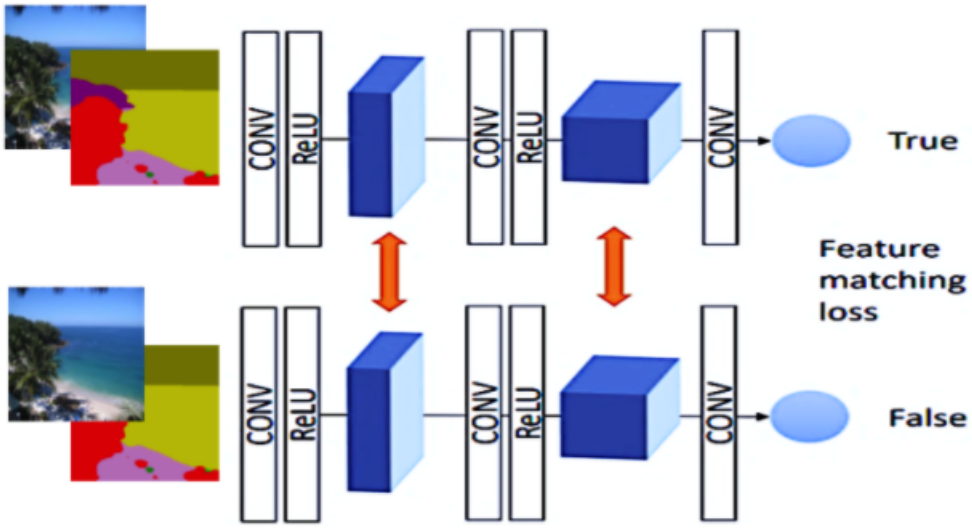


Figure 12: Feature Matching Loss Computation

5.3 Multimodal Synthesis

It is an image encoder which is in correspondence to getting diverse results out of a single semantic map. At the time of testing it serves as a style guide to capture the style of the targeted images. It takes the original image as the input, encodes it into 2 vectors which are mean and standard deviation for a normal Gaussian distribution. A random vector is then sampled out of this distribution to serve as input to the generator. Each random vector generated from sampling different values from the distribution will produce an image with the same semantic layout but different modal features like brightness, color etc.

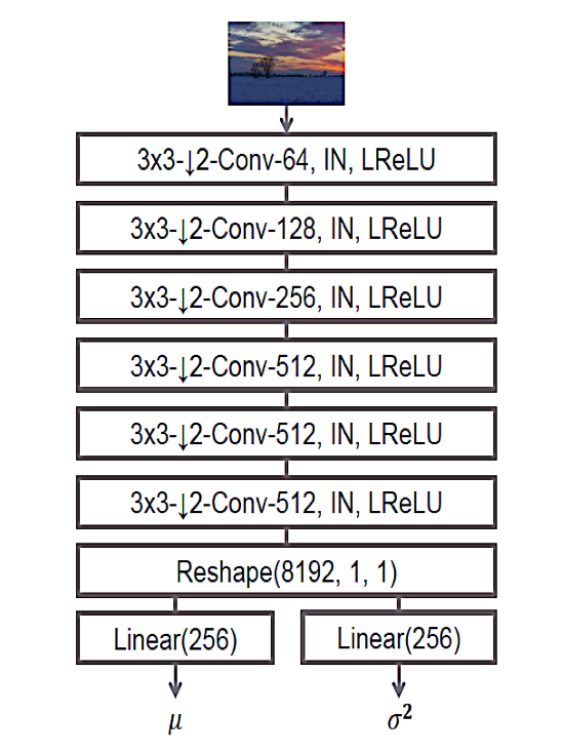


Figure 13: Encoder architecture

6 Objective and Loss Functions

6.1 Objective Function

The loss for discriminator should go up during training process and down for generator. It comprises of two terms: How accurately the discriminator classifies real images as real. How accurately the discriminator detects the fake images.

1. How accurately the discriminator classifies real images as real.
2. How accurately the discriminator detects the fake images.

$$\mathcal{L}_{\text{GAN}}(D,G) = \underbrace{E_{\boldsymbol{x}\sim p_{\text{data}}}[\log D(\boldsymbol{x})]}_{\text{accuracy on real images}} + \underbrace{E_{\boldsymbol{z}\sim\mathcal{N}}[\log(1-D(G(\boldsymbol{z}))]}_{\text{accuracy on fakes}}$$

When Discriminator (D) predicts the image as real the log D(x) should go up. Eventually as D gets better in detecting fake images G(z) then D(G(z)) will reach towards zero therefore 1-D(G(z)) will increase.

We will calculate the accuracy on batches. We will take input real images x and random vectors(z) and do the average. Then we will back propagate the error and simultaneously update the D's parameters to increase LGAN and G's parameters to decrease it.

Problem when training with the above mentioned loss is when the generator effectively fools the discriminator which implies D(G(z)) is large, LGAN loss gives small gradients so it is possible that G(z) may not get closer to the true data distribution to produce more realistic images. To solve this issue, we will use a least square based function (VLSGAN) which will result in more stable training and higher quality images and solves the problem of vanishing gradients.

One more problem we encounter while training is that higher resolution images to discrimination gives more information to it to detect fake images. State of the art GANs trains the network at lower resolution images by adding more layers until the desired resolution is reached.

One more term is introduced to the loss function LFM (Feature matching loss) that will cause the generator to make the distribution of features (layer activations) in the discriminators similar between the real data and the generator outputs the images by minimizing L1 distance between them. SPADE uses the hinge loss instead of squaring the values in VLSGAN

Overall, the optimization equation is:

$$min_G \left(\lambda \sum_{k=1,2,3} V_{LSGAN}\ (G, D_k) + (max_{D_1,D_2,D_3}) \sum_{k=1,2,3} L_{FM}\ (G, D_k)) \right)$$

GAN's Loss Function consists of the following loss terms as described in subsequent sections

6.2 Multiscale Adversarial Loss

GauGAN incorporates a hinge loss, that was also seen in papers like the SAGAN and Geometric GAN. The following is the loss function. Given an image generated by the generator, we create an image pyramid, resizing the generated image to multiple scales. Then, we compute the realness score using the discriminator for each of these scales and backpropagate the loss. The generator

and discriminator loss is given by,

$$L_G = -E_z \sim p_z, y \sim p_{data}\ D(G(z), y))$$

$$L_D = -E_{(x,y)} \sim p_{data},\ [min(0, -1 + D(x, y))] - E_z \sim p_z, y \sim p_{data}\ [min(0, -1 + D(G(z), y))]$$

6.3 Feature Matching Loss

This loss encourages the GAN to produce the images which are not merely able to fool the generator, but the generated images should also have the same statistical properties as that of real images. In order to accomplish this, we penalize the L1 distance between the discriminator feature maps of the real images and the discriminator feature maps of the fake images.

Feature matching loss is computed for all the scales of the generated image.

$$L_{FM}\ (G, D_k) = E_{s,x}\ \sum_{i=1}^{T} \frac{1}{N_i} ||\ D_k^i(s, x) -\ D_k^i(s, G(s))||_1$$

Here k represents which image scale we are using. We use T feature maps from the discriminator and N_i is the normalizing constant for each feature map so that the $L1$ difference between every pair of feature maps has the same scale despite different number of filters in each feature map

6.4 VGG Loss

This loss is similar to the above loss, the only difference being instead of using the discriminator to compute the feature maps, we use a VGG-19 pre-trained on Imagenet to compute the feature maps for real and the fake images. We then penalise the the L1 distance between these maps.

$$L_{VGG}\ (G, D_k) = E_{s,x}\ \sum_{i=1} \frac{1}{2^i} ||\ VGG(x, M_i)) - VGG(G(s), M_i)||_1$$

where $VGG(x, m)$is the feature map of VGG19 when x is the input and
M = 'relu$1'_1$,'$relu2'_1$,'$relu3'_1$,'$relu4'_1$,'$relu5'_1$

6.5 Encoder Loss

Authors use a KL divergence Loss for the encoder. Note that, while we can use a style image during inference, the ground truth of the segmentation map serves as our style image during training. This makes sense as the style of ground truth as well as the image we are trying to synthesize is the same. You may recognise the above loss function as the the regularization loss term from Variational Auto-Encoder Loss. With the Encoder, GauGAN behaves as a sort of Variational AutoEncoder, with the GauGAN playing the part of the decoder. For this note familiar with Variational AutoEncoders, the KL divergence loss term acts as a regularizer for the Encoder. This loss penalizes the KL divergence between the distribution predicted by our encoder and a zero mean Gaussian. If not for this loss, the encoder could cheat by assigning a different random vector for each training example in our dataset, rather than actually learning a distribution that captures the modalities of our data.

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

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

7 Training

7.1 Parameters

Parameters	**Values**
Optimizer	Adam
Learning Rate Generator	0.0003
Learning Rate Discriminator	0.0001
Convolutional filters at start in G	88
Convolutional filters at start in D	88
Convolutional filters at start in image encoder	32
Batch Size	16
Betas	0.1 and 0.9
GPU	4
D steps per G	1
Aspect Ratio	1.0
Gan Loss	Hinge
Image Encoder Loss	KL Divergence
Loss Initialization Rate	Xavier
Normalization	Spectral Norm
Kernel Size	3x3
Total Labels	150
Lambda Feat	10
Lambda KLD	0.05
Lambda VGG	10
Layers in D	4
Output Channels	3
z dim	256

7.2 Evaluation

Loss	Ideal Range	Training Values
Feature Matching Loss	7-14	11.443
Encoder Loss (KLD Loss)	0-3	1.463
Adversarial Loss	0-2	0.777
VGG Loss (Perceptual Loss)	6-8	6.8

The training was intitally preceded with multiple test runs (5 runs) on P3.2x large (1GPU - Tesla V100 GPU) for getting the right set of hyperparameters for this dataset. We had couple of challenges installing SPADE and its dependencies as it required packages of previous versions. The training was finally run on P3.8x large for 50 epochs. The cost is described in the Statistics section. P3.8x large provides us 4 Tesla V100 GPUs which had optimal compute power required to train with our model architecture and huge dataset.

Loss Curves: The loss curves monitored continuously and seemed to overfit intitally, however by the quality of images generated for different epochs made it easy to understand the model

behaviour. The loss values were in the expected range and model predicted results as expected. So we had a good fit model trained for this dataset and results justify how well the model is trained.

7.3 Statistics

Parameters	P3.2x Large	P3.8x Large	P3.16x Large
GPU Count	1	4	8
Cost per hour	$ 0.918 per hour	$ 3.672 per hour	$ 7.344 per hour
Time per epoch	2hrs 20mins	1hr 10mins	38mins
Cost for 50 epochs	$ 106.9	$ 207.46	$ 232.43
Total time to train	116.5 hrs	56.5 hrs	31.6 hrs

The training was executed for 50 epochs on P3.8x large for 50 epochs. The results provided in this report are based on the model trained on this hardware. Currently, we want to train the network on P3.16x large which has 8 Tesla V100 GPUs. This would lead to better training results as per the published paper.

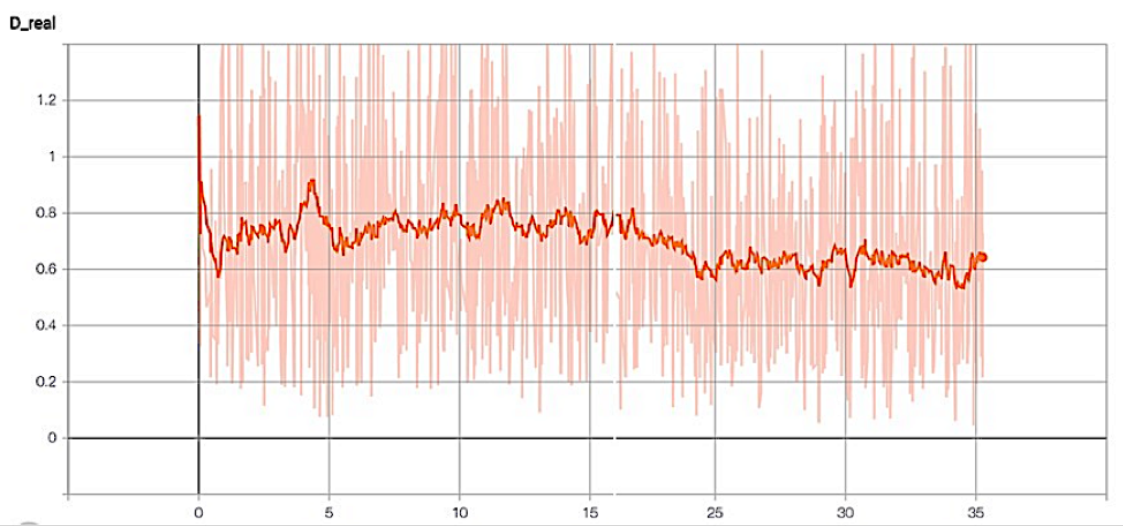


Figure 14: D Real

Figure 15: D Fake

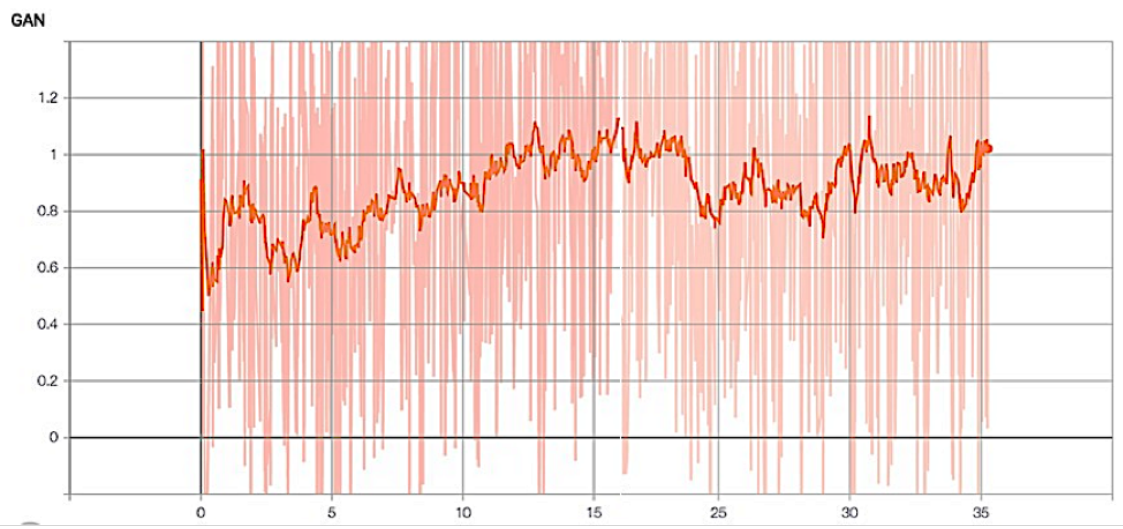


Figure 16: GAN

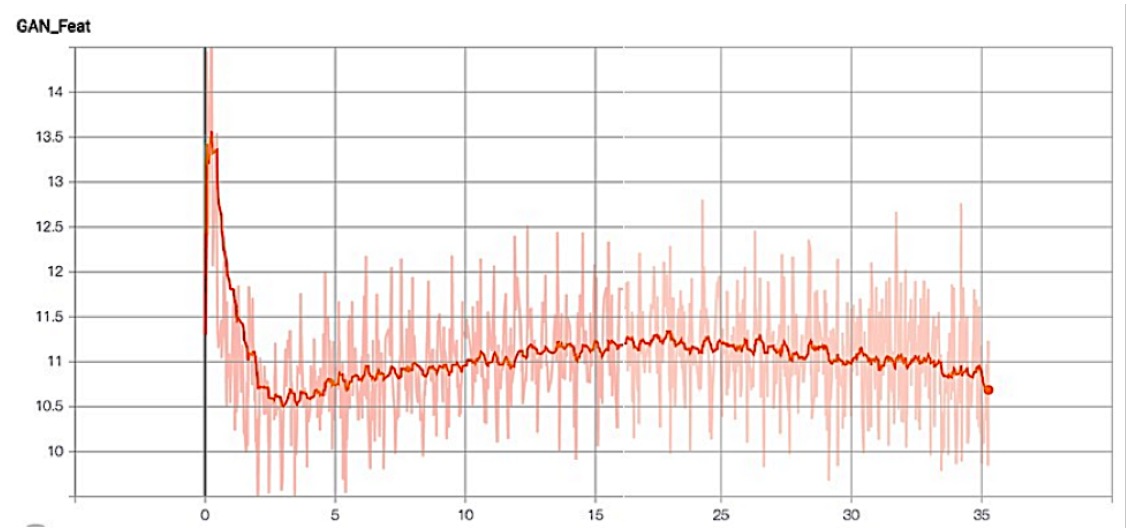


Figure 17: FM

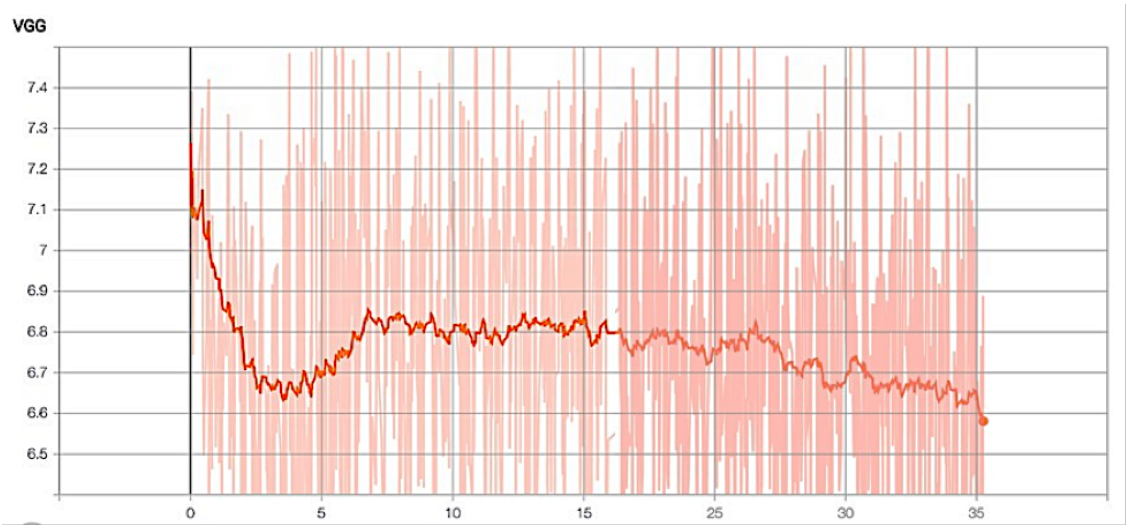


Figure 18: VGG

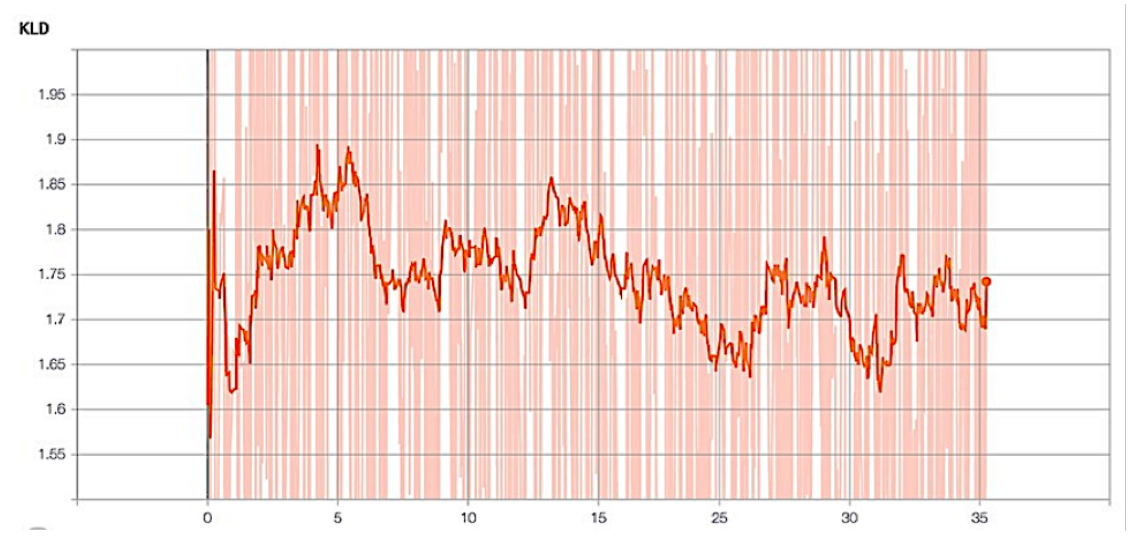


Figure 19: KLD

8 Inference

8.1 Model Evaluation

Model Evaluation is amongst the toughest jobs while working with GANs. The reason being there are no fixed results coming out of GAN like the ones we get in object detection, image classification. In object detection, we have the ground truth bounding box and then we compare it with the predicted bounding box. It's not the case here. Let's assume we do somewhat similar in GAN. We have the semantic map as an input which generates the photorealistic image. Suppose we somehow manage to create the ground truth image from a semantic map so as to compare. Suppose we used distance metric to compare between these 2 images, which is ground truth image and the GAN generated image. For example: if we have an ground truth image of a beach gathered during the day and GAN generated the image of night duration. These 2 images are ideally different renditions of the same scene at different times of the day. We know visually that both these images are equally good. When we use distance metric, the later image will have more dark colors, per pixel squared distance would penalize it. So, the distance metric wouldn't work in our case.

8.2 FID Score

Frechet Inception Distance (FID) has been becoming popular for the GAN evaluation. We use the Inception V3 network which is taken as a statistical computation function. The idea is to capture the statistics of an image rather than the pixels of the two images.

The procedure of calculating FID score is as follows:

1. Real Image R and Fake image F put into Inception V3 network.
2. 2048D pool layer to calculate the feature vector for both images.
3. Compute distance between these two feature vectors.

$$\textbf{FID} = ||\mu_r - \mu_g||^2 + Tr(\sum_r + \sum_g - 2(\sum_r \sum_g)^{\frac{1}{2}})$$

where,
μ - mean of the feature vector,
$\sum$ - variance matrix and
Tr - Trace operator.

Inception V3 network is a color invariant network and hence a better choice than the distance metrics. So, here the time of day, color, orientation doesn't affect the diverse image. The FID score calculated for our test data: 58.22 When compared to SPADE implementation on ADE20k outdoor dataset, the FID score was 63.3 The lower the FID score, the better it is.

8.3 Human Preference Statistics

As already mentioned, the evaluation is difficult when it comes to GAN. Human level evaluation seems to be a better choice too. We conducted a survey where we asked participants to rate the images generated from our model between 1 to 10, where 1 being the least photorealistic and 10 being the most photorealistic. For the 8 images that were provided in the survey (link is provided below), the ratings for different images were captured and below are the results averaged for each category.

Survey Link

1. Most Photorealistic Images: 50% of the images
2. Nearly Photorealistic Images: 0% of the images
3. Just Photorealistic Images: 25% of the images
4. Not Photorealistic Images: 25%

1. Not Photorealistic: 1 - 2 2. Just Photorealistic: 3 - 5 3. Nearly Photorealistic: 6 - 8 4. Mostly Photorealistic: 9 - 10

9 Results

9.1 Good Performance

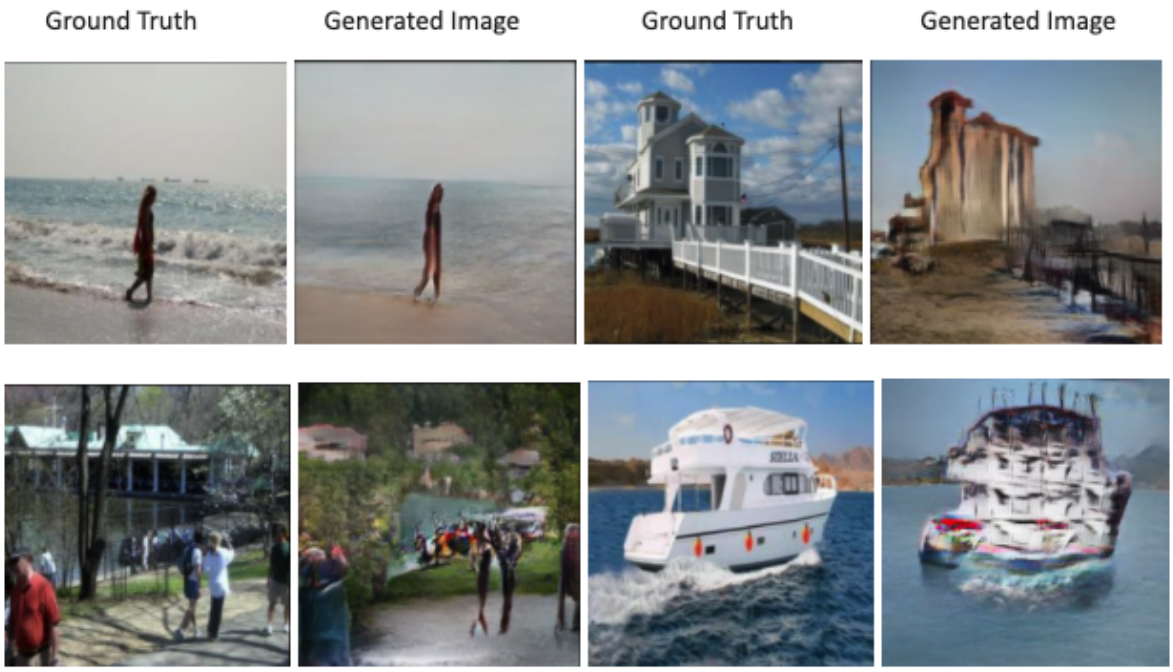
Despite relatively small training set and short training time compared to the technical paper, our model could still achieve very impressive performance and very similar to ground truth images quite well on some straight forward landscape and other scenes. Our model can clearly distinguish sky, vegetation, sand, stone, cliffs, ocean, wood etc... Class imbalance due to insufficient data can be managed by more extensive and stratified training. With the intital plan to train over cityscaped images, such as skyscrapers, buildings and urban images, it became challenging to train small objects as data was not sufficient to train all objects and resulted in bad results. However, to demonstrate the positives achieved by SPADE implementation, we used landscape images which had plenty of object information that we intended to train for and achieved exemplary results with the time and resources available to us. The generated images are provided in next page.

9.2 Failed cases

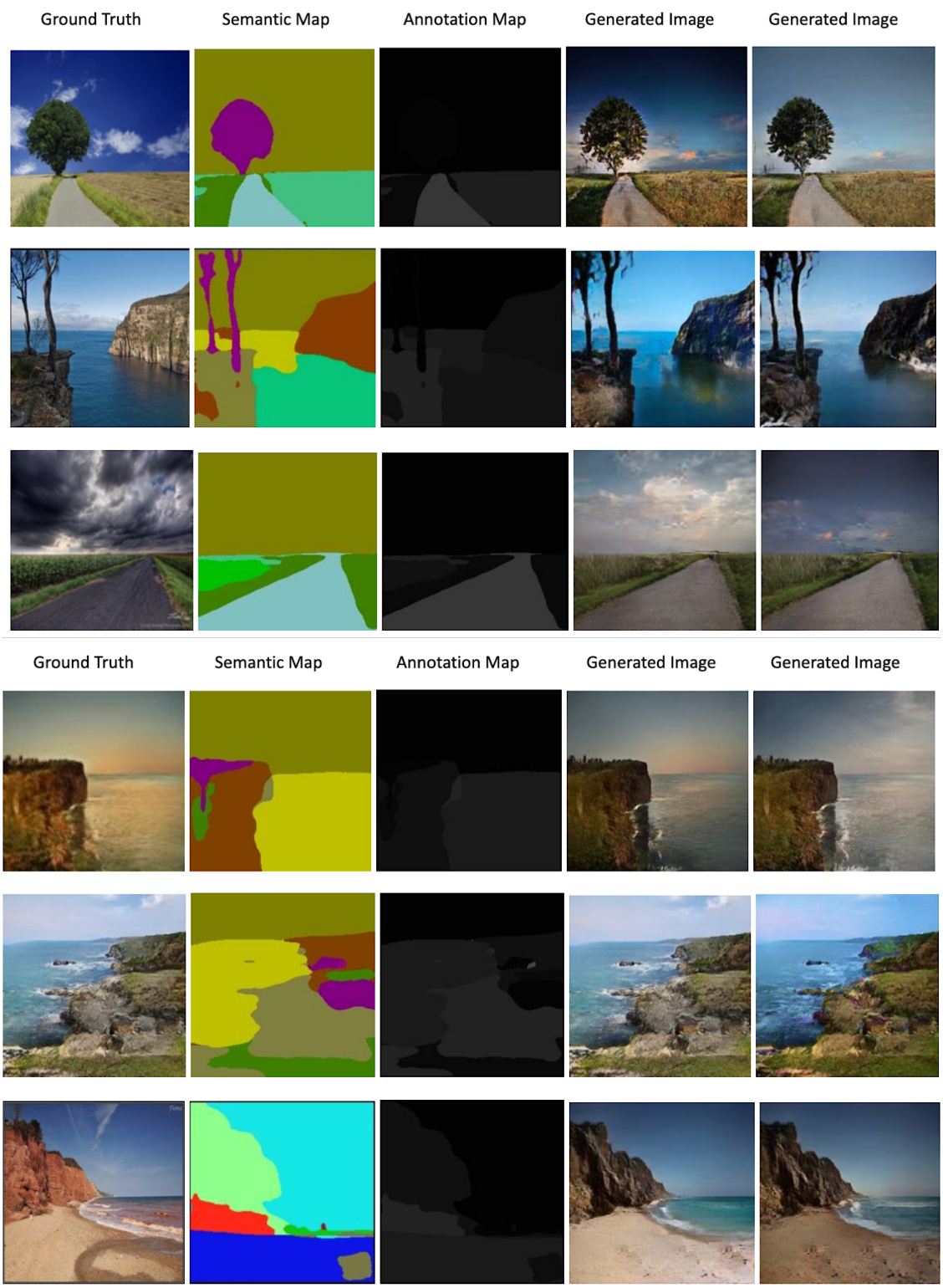
However, when the scene becomes more complicated in colors and semantics, the limitations of our model expose. Sometimes it has the ability to understand the contextual meaning of image but fails to colorize some semantic areas accurately and properly. If the input image contains various objects or components, its main part is highly artificial or it is distorted artificially on human purpose, the predicted results will be awkward and gain little. That is because the training image set is not large enough to cover enough training samples for each situation.

There was a fair share of successfully generated photorealistic images and some failed cases as well. This is mainly due to the reason described above. The successfully generated photorealistic images and the failed ones are provided below.

***** FAILED PHOTOREALISTIC IMAGES *****



*** SUCCESSFULLY GENERATED PHOTOREALISTIC IMAGES ***



More Results have been provided in Appendix A

10 Learning with Challenges

1. Worked with state-of-the-art architecture (SPADE) among photorealistic image generation methods.
2. Researched and learnt various image segmentation techniques.
3. Selected and trained with challenging dataset (Places365) with diverse and difficult to train categories.
4. Re-training with right hyperparameters especially Generator and Discriminator Convolutional Layers

11 Future Work

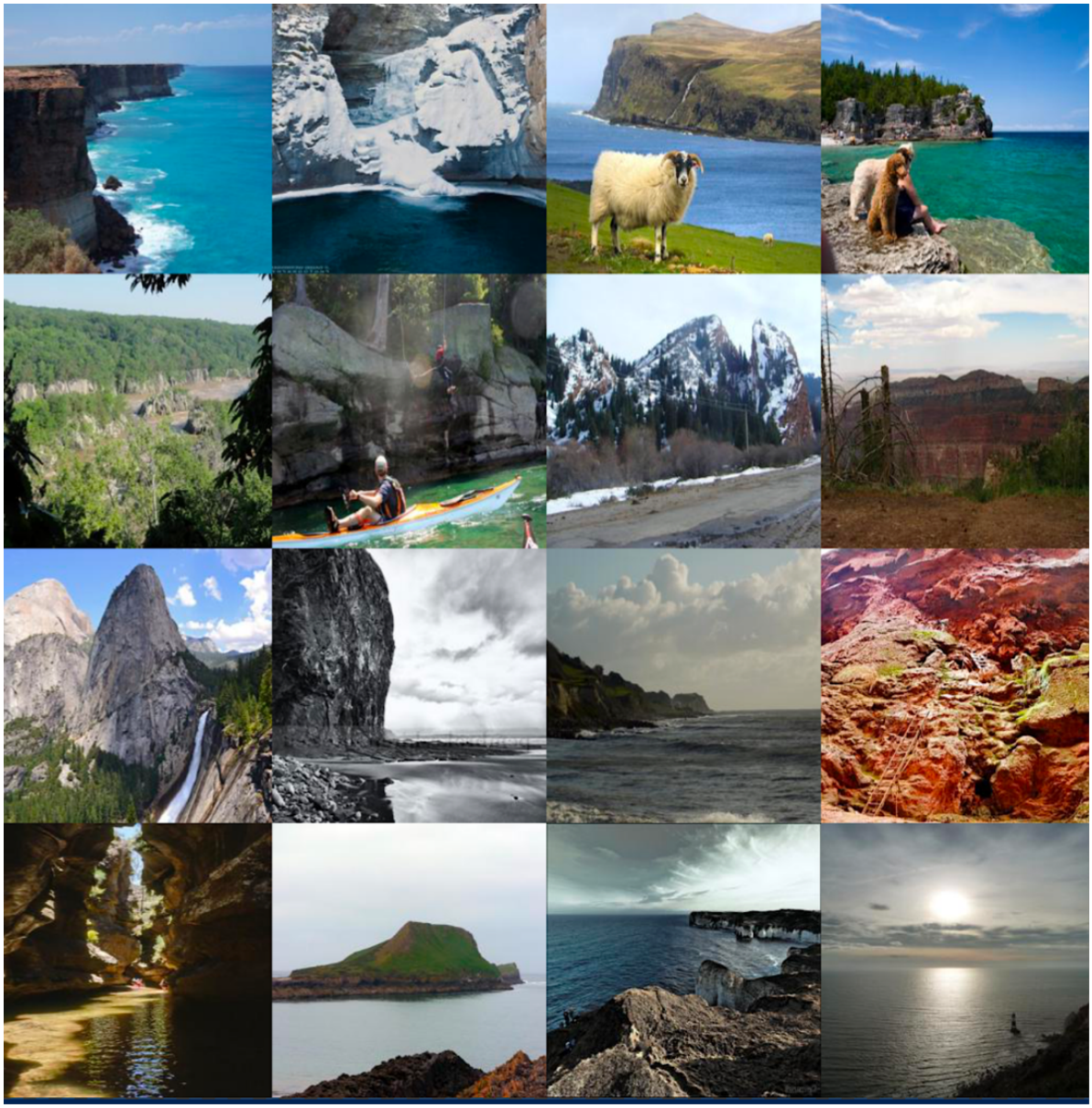
1. Training with higher resolution images (1024 x 712) and more categories.
2. Hyperparameter Tuning:
   (a) Vary loss weights
   (b) Vary number of convolutional layers of Generator and Discriminator networks
3. Improving segmentation accuracy by training a robust segmentation model or selecting a right pre-trained model for highest accuracy
4. Data preprocessing: Image filters could be used to reduce the noise of our images and to preserve the image details as much as possible. The processing effect will directly affect the effectiveness and reliability of subsequent performance of our model.

12 Conclusion

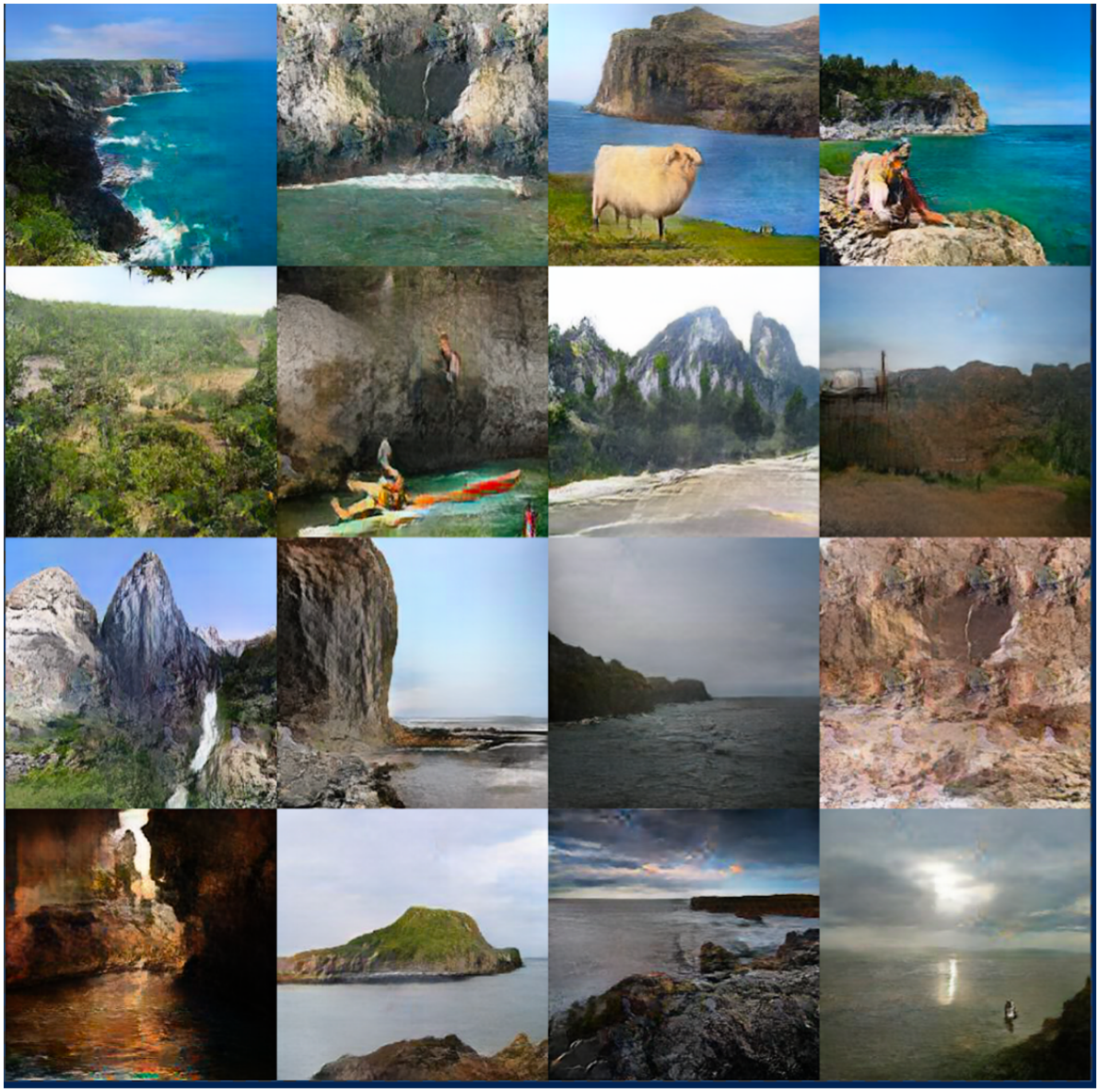
In our project we have proposed the new normalization technique known as Spatially Adaptive Normalization (SPADE) which performs the affine transformation in the normalization layers to utilize the input semantic maps. SPADE leads to the formation of photorealistic images for diverse scenes such as sky, ocean, cliff, nature, road. Also, a technique known as multi-modal synthesis is also used to generate the pictures of different styles.

A More Training Results

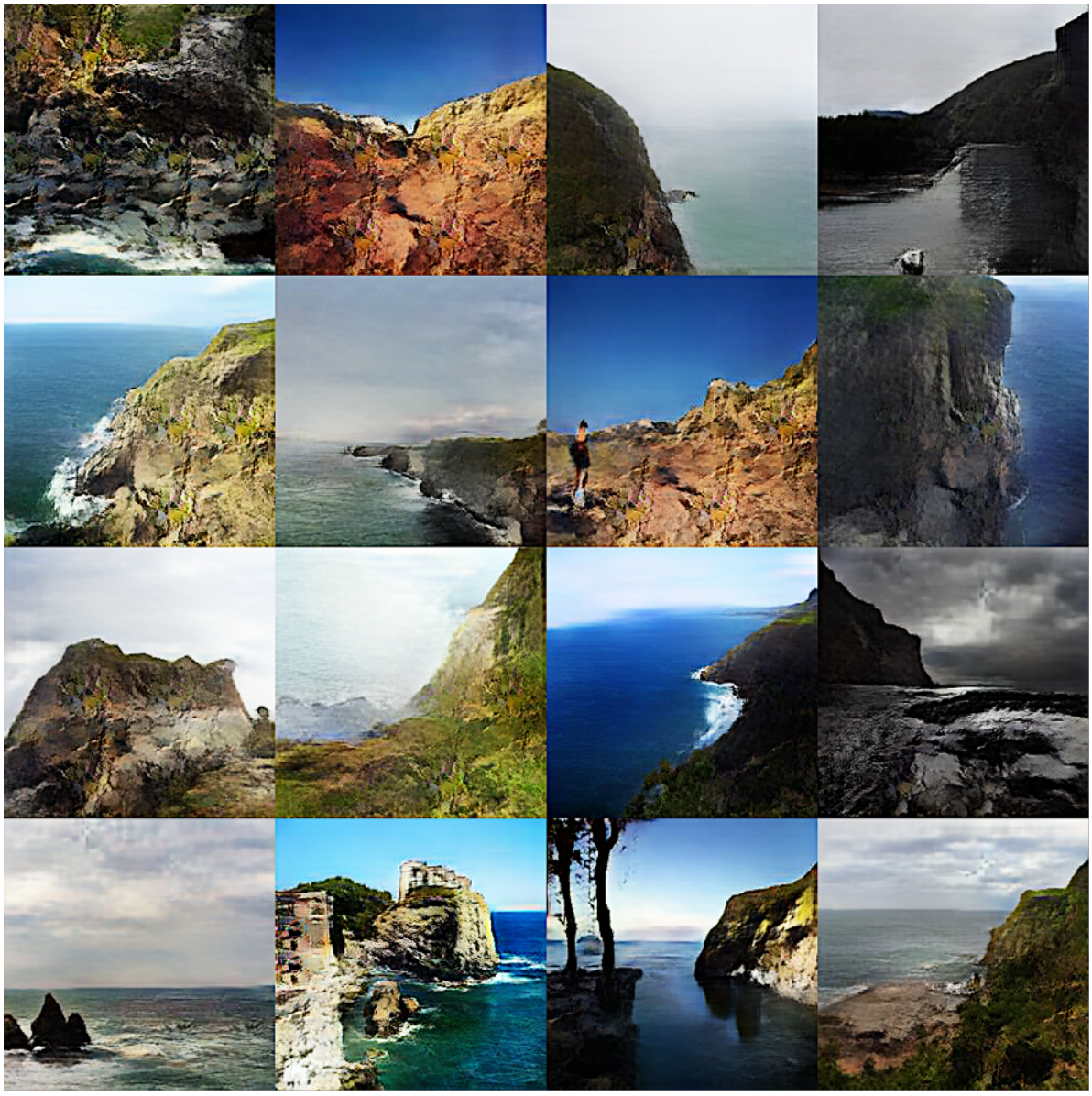
ORIGINAL GROUND TRUTH IMAGES The images provided below are set of original images that were provided as input to GAN.



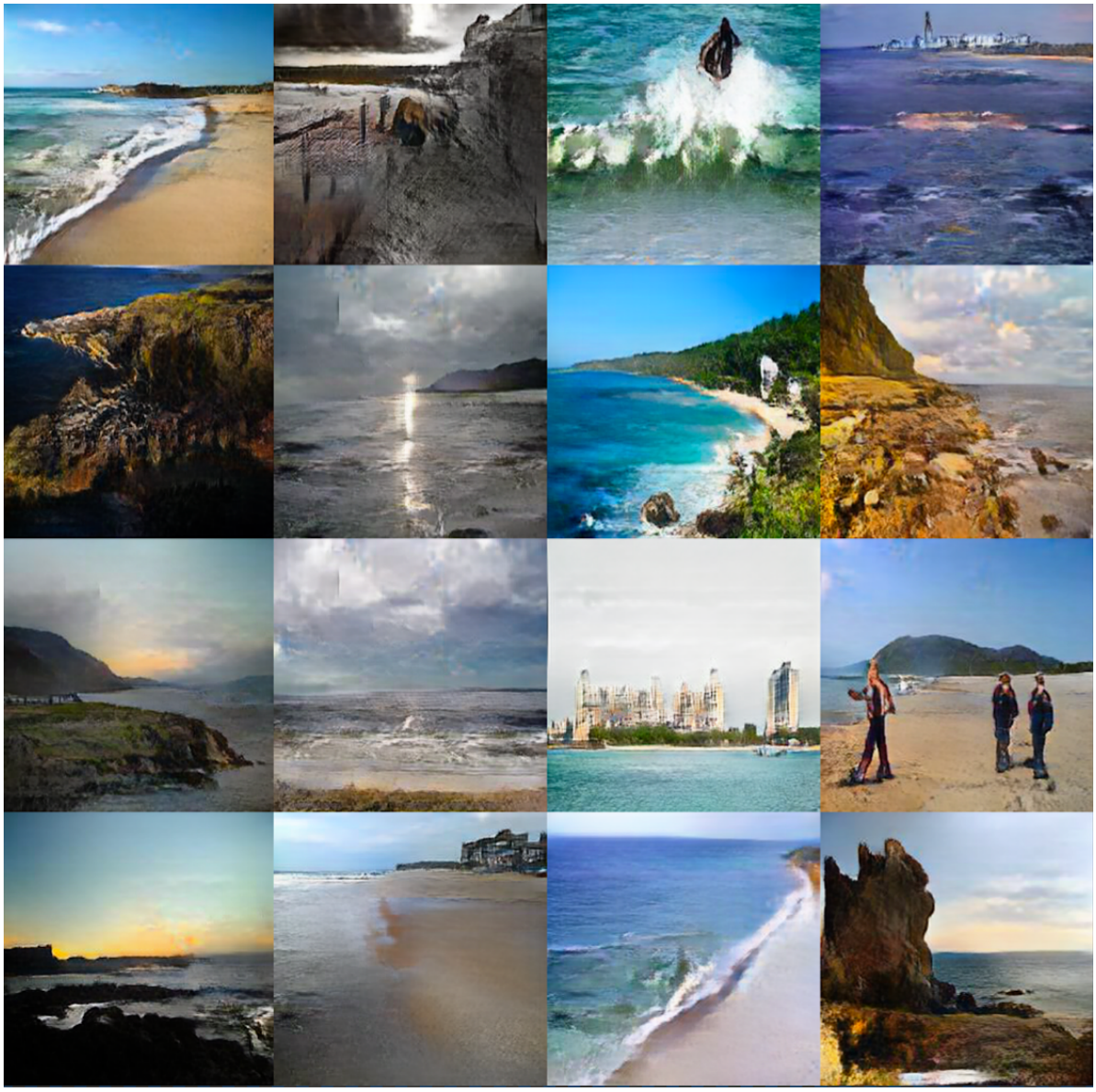
GOOD PHOTO-REALISTIC IMAGES - 1 Successfully generated photorealistic images for the original images provided in previous page



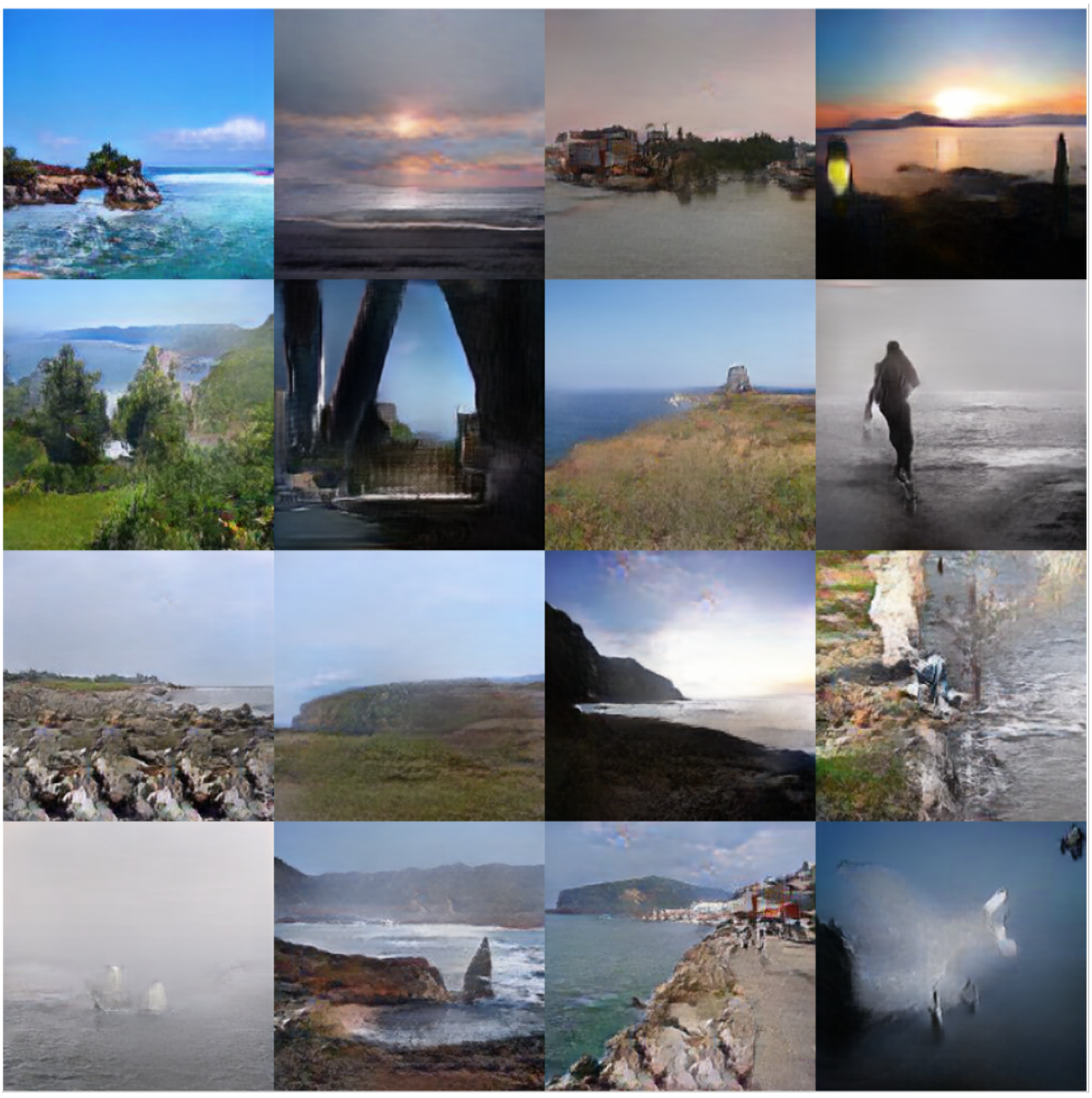
GOOD PHOTO-REALISTIC IMAGES - 2 These are few more images generated by GAN



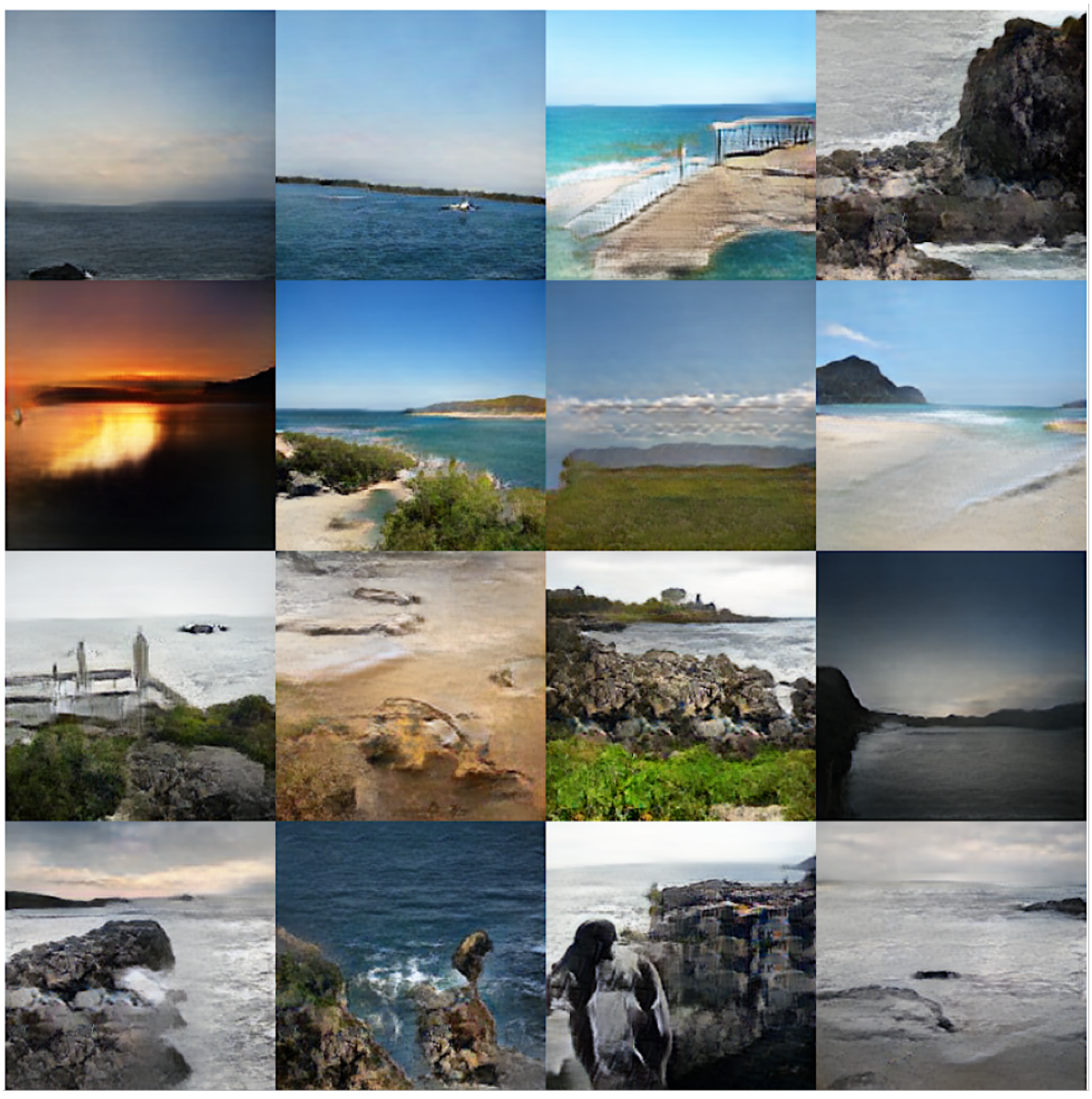
GOOD PHOTO-REALISTIC IMAGES - 3



FAILED PHOTO-REALISTIC IMAGES - 1



FAILED PHOTO-REALISTIC IMAGES - 2



B Other Resources

Github: The complete source code files developed as part of this project is uploaded to GitHub. Please refer to this link to find all the steps, procedures to successfully execute the project.

1. Source code, setup and installation files are provided in the below GitHub repository Click here to view the repository.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large scale gan training for high fidelity natural image synthesis". In: *arXiv preprint arXiv:1809.11096* (2018).

[3] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. "A learned representation for artistic style". In: *arXiv preprint arXiv:1610.07629* (2016).

[4] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[5] Aaron Hertzmann et al. "Image analogies". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001, pp. 327–340.

[6] Xun Huang and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1501–1510.

[7] Xun Huang et al. "Multimodal unsupervised image-to-image translation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 172–189.

[8] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[9] Diederik P Kingma and Max Welling. "Stochastic gradient VB and the variational auto-encoder". In: *Second International Conference on Learning Representations, ICLR*. Vol. 19. 2014.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[11] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. "Which training methods for GANs do actually converge?" In: *arXiv preprint arXiv:1801.04406* (2018).

[12] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

[13] Takeru Miyato and Masanori Koyama. "cGANs with projection discriminator". In: *arXiv preprint arXiv:1802.05637* (2018).

[14] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional image synthesis with auxiliary classifier gans". In: *International conference on machine learning*. PMLR. 2017, pp. 2642–2651.

[15] Taesung Park et al. *Semantic Image Synthesis with Spatially-Adaptive Normalization*. 2019. arXiv: 1903.07291 [cs.CV].

[16] Tim Salimans and Durk P Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep neural networks". In: *Advances in neural information processing systems* 29 (2016), pp. 901–909.

[17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022* (2016).

[18] Ting-Chun Wang et al. "High-resolution image synthesis and semantic manipulation with conditional gans". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8798–8807.

[19] Xintao Wang et al. "Recovering realistic texture in image super-resolution by deep spatial feature transform". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 606–615.

[20] Yuxin Wu and Kaiming He. "Group normalization". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[21] Han Zhang et al. "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5907–5915.

[22] Hengshuang Zhao et al. *Pyramid Scene Parsing Network*. 2017. arXiv: 1612.01105 [cs.CV].