MANGALORE INSTITUTE OF TECHNOLOGY AND ENGINEERING

(An ISO 9001:2015 Certified Institution)

BADAGA MIJAR, MOODBIDRI, DK DIST - 574225

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(NBA Accredited)

MITE

Invent Solutions

A

PROJECT LITERATURE REVIEW REPORT ON

"GENERATION OF PHOTOREALISTIC IMAGE USING GAN AND SPADE"

SUBMITTED BY

ADARSH REVANKAR	**4MT16CS003**
AKSHAYA M	**4MT16CS007**
SHUBHAM DOGRA	**4MT16CS025**

<u>Project Guide</u>

DR. VENKATRAMANA BHAT P

Professor & Head of Department

Dept of CSE

INEDX

INTRODUCTION

A tool for the synthesis of photorealistic images using conditional image synthesis is proposed. Conditional image synthesis refers to the task of generating photorealistic images conditioning on certain input data. Seminal work computes the output image by stitching pieces from a single image or using an image collection.

There is a requirement of a specific form of conditional image synthesis, which is converting a semantic segmentation mask to a photorealistic image. This form has a wide range of applications such as content generation and image editing. This form is referred as semantic image synthesis. Here it is shown that the conventional network architecture, which is built by stacking convolutional, normalization, and nonlinearity layers, is at best suboptimal because their normalization layers tend to "wash away" information contained in the input semantic masks. To address the issue, spatially-adaptive normalization is proposed, which is a conditional normalization layer that modulates the activations using input semantic layouts through a spatially adaptive, learned transformation and can effectively propagate the semantic information throughout the network.

LITERATURE SURVEY

I Dataset

A new large-scale dataset is introduced which addresses three core research problems in scene understanding: detecting non-iconic views (or non-canonical perspectives) of objects, contextual reasoning between objects and the precise 2D localization of objects. The current recognition systems perform fairly well on iconic views, but struggle to recognize objects otherwise – in the background, partially occluded, amid clutter – reflecting the composition of actual everyday scenes.

To create a large-scale dataset that accomplishes these three goals, a novel pipeline is employed for gathering data with extensive use of Amazon Mechanical Turk. First and most importantly, a large set of images containing contextual relationships and non-iconic object views is harvested. This is accomplished by using the technique that queries for pairs of objects in conjunction with images retrieved via scene-based queries. Next, each image was labeled as containing particular object categories using a hierarchical labeling approach. For each category found, the individual instances were labeled, verified, and finally segmented.

The Microsoft Common Objects in Context (MS COCO) dataset contains 91 common object categories (Figure 1.1) with 82 of them having more than 5,000 labeled instances. In total the dataset has 2,500,000 labeled instances in 328,000 images. In contrast to the popular ImageNet dataset, COCO has fewer categories but more instances per category. This can aid in learning detailed object models capable of precise 2D localization. The dataset is also significantly larger in number of instances per category than the PASCAL VOC and SUN datasets. Additionally, a critical distinction between our dataset and others is the number of labeled instances per image which may aid in learning contextual information. In contrast, the SUN dataset, which contains significant contextual information, has over 17 objects and “stuff” per image but considerably fewer object instances overall.

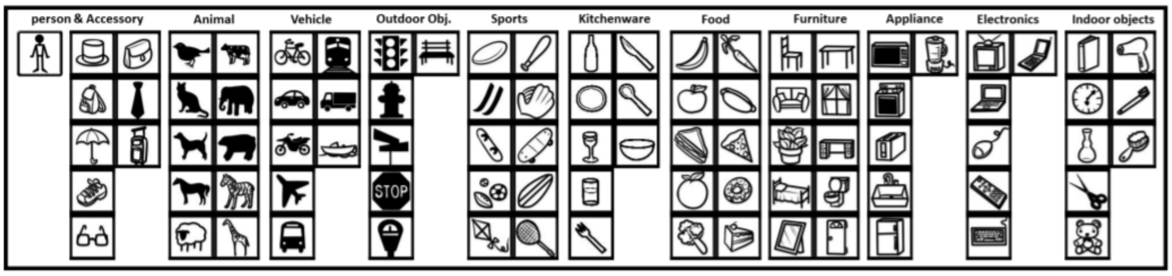

Figure 1.1: Icons of 91 categories in the MS COCO dataset grouped by 11 super-categories.

II Generative Adversarial Network

The promise of deep learning is to discover rich, hierarchical models that represent probability distributions over the kinds of data encountered in artificial intelligence applications. Deep generative models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context.

Generative adversarial networks (GANs) are an emerging technique for both semi-supervised and unsupervised learning. They achieve this through implicitly modelling high-dimensional distributions of data. They can be characterized by training a pair of networks in competition with each other. A common analogy, apt for visual data, is to think of one network as an art forger, and the other as an art expert. The forger, known in the GAN literature as the generator, G, creates forgeries, with the aim of making realistic images. The expert, known as the discriminator, D, receives both forgeries and real images, and aims to tell them apart. Both are trained simultaneously, and in competition with each other.

Crucially, the generator has no direct access to real images - the only way it learns is through its interaction with the discriminator. The discriminator has access to both the synthetic samples and samples drawn from the stack of real images. The error signal to the discriminator is provided through the simple ground truth of knowing whether the image came from the real stack or from the generator. The same error signal, via the discriminator, can be used to train the generator, leading it towards being able to produce forgeries of better quality.

Generative Adversarial Network is a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. Generative Adversarial Networks are most straightforward to apply when the models are both multilayer perceptrons. To learn the generator's distribution Pg over data x, define a prior on input noise variables $Pz(z)$, then represent a mapping to data space as $G(x;\ \theta_d)$, where G is a differentiable function represented by a multilayer perceptron with parameters θg. Second multilayer perceptron $D(x;\ \theta_d)$ that outputs a single scalar. D(x) represents the probability that x came from the data rather than pg. Train D to maximize the probability of assigning the correct label to both training examples and samples from G. Simultaneously train G to minimize $log(1\ -\ D(G(z)))$:

In other words, D and G play the following two-player minimax game with value function $V\ (G, D)$:

$$min_G\ max_D V(D, G) = \mathrm{E}_{x \sim P_{data}(x)}[logD(x)] + \mathrm{E}_{z \sim P_z(x)}[\log\ (1 - D(G(z)))] \quad (1)$$

Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting. Instead, alternate between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough.

Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $log(1\ -\ D(G(z)))$ saturates. Rather than training G to minimize $log(1\ -\ D(G(z)))$ can train G to maximize $log(D(G(z)))$.

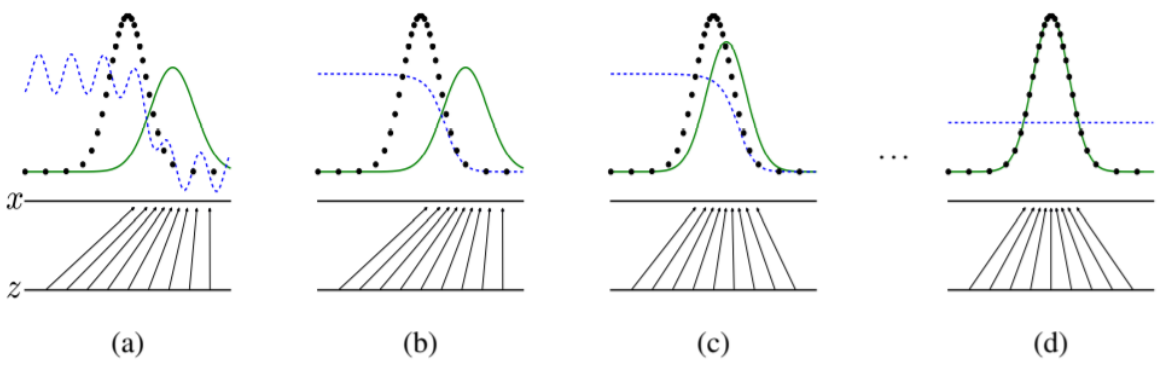

Figure 2.1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) px from those of the generative distribution Pg (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution P_g on transformed samples. G contracts in regions of high density and expands in regions of low density of pg. (a) Consider an adversarial pair near convergence: Pg is similar to $Pdata$ and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D*(x) = \frac{P_{data}}{P_{data}+P_g(x)}$. (c) After an update to G, gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $Pd = Pdata$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
              $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by descending its stochastic gradient:
          $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator. This means that components of the input are not copied directly into the generator's parameters. Another advantage of adversarial networks is that they can represent very sharp, even degenerate distributions, while methods based on Markov chains require that the distribution be somewhat blurry in order for the chains to be able to mix between modes.

GANs have attracted considerable attention due to their ability to leverage vast amounts of unlabelled data. While much progress has been made to alleviate some of the challenges related to training and evaluating GANs, there still remain several open challenges.

i) Mode Collapse: Common problem of GANs involves the generator collapsing to produce a small family of similar samples (partial collapse), and in the worst case producing simply a single sample. Diversity in the generator can be increased by practical hacks to balance the distribution of samples produced by the discriminator for real and fake batches, or by employing multiple GANs to cover the different modes of the probability distribution

ii) Training instability: In a GAN, the Hessian of the loss function becomes indefinite. The optimal solution, therefore, lies in finding a saddle point rather than a local minimum. In deep learning, a large number of optimizers depend only on the first derivative of the loss function; converging to a saddle point for GANs requires good initialization

iii) Evaluating Generative Models: How can one gauge the fidelity of samples synthesized by a generative model? Should we use a likelihood estimation? Can a GAN trained using one methodology be compared to another?

III Convolutional Neural Network

A. Introduction

The term Deep Learning or Deep Neural Network refers to Artificial Neural Networks (ANN) with multi layers. Over the last few decades, it has been considered to be one of the most powerful tools, and has become very popular in the literature as it is able to handle a huge amount of data. The interest in having deeper hidden layers has recently begun to surpass classical methods performance in different fields; especially in pattern recognition. One of the most popular deep neural networks is the Convolutional Neural Network (CNN). It take this name from mathematical linear operation between matrixes called convolution. CNN have multiple layers; including convolutional layer, non-linearity layer, pooling layer and fully connected layer. The convolutional and fully- connected layers have parameters but pooling and non-linearity layers don't have parameters. The CNN has an excellent performance in machine learning problems. Especially the applications that deal with image data, such as largest image classification data set (Image Net), computer vision, and in natural language processing (NLP) and the results achieved were very amazing.

The most important assumption about problems that are solved by CNN should not have features which are spatially dependent. In other words, for example, in a face detection application, we do not need to pay attention to where the faces are located in the images. The only concern is to detect them regardless of their position in the given images. Another important aspect of CNN, is to obtain abstract features when input propagates toward the deeper layers. For example, in image classification, the edge might be detected in the first layers, and then the simpler shapes in the second layers, and then the higher-level features such as faces in the next layers.

B. Components of CNN

a. Convolution

Let's assume that the network receives raw pixels as input. Therefore, to connect the input layer to only one neuron (e.g. in the hidden layer in the Multi- Layer perceptron). If we add one more neuron into the hidden layer, then we will need another $32\times32\times3$ weight connection, which will become in total, $32\times32\times3\times2$ parameters. To make it clearer, more than 6000 weight parameters are used to connect the input to just only two nodes. It may be thought that two neurons might not be enough for any useful processing for an image classification application.

To make it more efficient, we can connect the input image to the neurons in the next layer with exactly the same values for the height and width. It can be assumed this network is applied for the type of processing such as the edge in the image. However, the mentioned network needs 32×32×3 by 32×32 weight connections, which are (3,145,728).

Therefore, looking for a more efficient method, it emerged that instead of a full connection, it is a good idea to look for local regions in the picture instead of in the whole image. In other words, the hidden neurons in the next layer only get inputs from the corresponding part of the previous layer. For example, it can only be connected to 5×5 neurons. Thus, if we want to have 32×32 neurons in the next layer, then we will have 5×5×3 by 32x32 connections which is 76,800 connections (compared to 3,145,728 for full connectivity).

b. Stride

In fact, CNN has more options which provide a lot of opportunities to even decrease the parameters more and more, and at the same time reduce some of the side effects. One of these options is stride. In the above-mentioned example, it is simply assumed that the next layer's node has lots of overlaps with their neighbors by looking at the regions. We can manipulate the overlap by controlling the stride. Fig 3.2.2, shows a given 7×7 image. If we move the filter one node every time, we can have a 5x5 output only. Note that the output of the three left matrices in Fig. 6, have an overlap (and three middle ones together and three right ones also). However, if we move and make every stride 2, then the output will be 3x3. Put simply, not only overlap, but also the size of the output will be reduced.

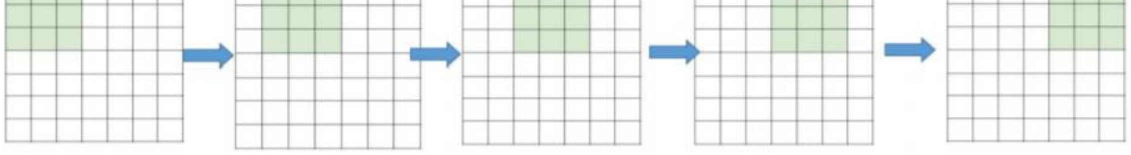

Figure 3.2.2: Stride 1, the filter window moves only one time for each connection

C. Non-Linearity

The next layer after the convolution is non-linearity. The non-linearity can be used to adjust or cut-off the generated output. This layer is applied in order to saturate the output or limiting the generated output. For many years, sigmoid and tanh were the most popular non-linearity. Figure 3.3 shows the common types of nonlinearity. However, recently, the Rectified Linear Unit (ReLU) has been used more often for the following reasons.

1. ReLU has simpler definitions in both function and gradient.

2. The saturated function such as sigmoid and tanh cause problems in the back propagation. As the neural network design is deeper, the gradient signal begins to vanish, which is called the “vanishing gradient”. This happens since the gradient of those functions is very close to zero almost everywhere but the center. However, the ReLU has a constant gradient for the positive input. Although the function is not differentiable, it can be ignored in the actual implementation.

3. The ReLU creates a sparser representation However, sigmoid and tanh always have non-zero results from the gradient, which might not be in favor for training.

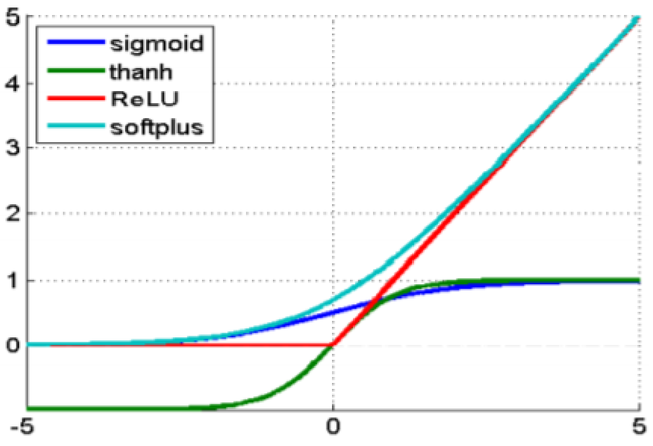

Figure 3.3: Common types of nonlinearity

D. Pooling

The main idea of pooling is down-sampling in order to reduce the complexity for further layers. In the image processing domain, it can be considered as similar to reducing the resolution. Pooling does not affect the number of filters. Max-pooling is one of the most common types of pooling methods. It partitions the image to sub-region rectangles, and it only returns the maximum value of the inside of that sub-region. One of the most common sizes used in max-pooling is 2×2. As we can see in Fig 2.4, when pooling is performed in the top-left 2×2 blocks (pink area), it moves 2 and focus on top-right part. This means that stride 2 is used in pooling. To avoid down-sampling, stride 1 can be used, which is not common. It should be considered that down-sampling does not preserve the position of the information. Therefore, it should be applied only when the presence of information is important (rather than spatial information). Moreover, pooling can be used with non-equal filters and strides to improve the efficiency.

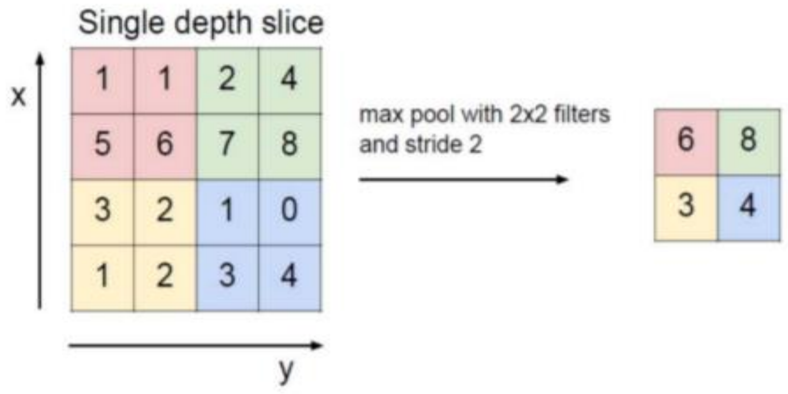

Figure 3.4: Max-pooling is demonstrated. The max-pooling with 2x2 filter and stride 2 lead to down-sampling of each 2x2 blocks is mapped to 1 block (pixel).

E. Fully Connected Layer

The fully-connected layer is a similar to the way that neurons are arranged in a traditional neural network. Therefore, each node in a fully-connected layer is directly connected to every node in both the previous and in the next layer. Each of the nodes in the last frames in the pooling layer are connected as a vector to the first layer from the fully-connected layer. These are the most parameters used with the CNN within these layers, and take a long time in training.

IV Spatially-Adaptive Normalization

Spatially Adaptive Normalization, a simple but effective layer for synthesizing photorealistic images given an input semantic layout. Previous methods directly feed the semantic layout as input to the deep network, which is then processed through stacks of convolution, normalization, and on linearity layers. This is suboptimal as the normalization layers tend to “wash away” semantic information. To address the issue, use the input layout for modulating the activations in normalization layers through a spatially-adaptive, learned transformation.

Related Works to reach to this normalization layer

1. Unconditional normalization layers, important component in modern deep networks and can be found in various classifiers, including the Local Response Normalization and the Batch Normalization (BatchNorm). Other popular normalization layers include the Instance Normalization (InstanceNorm), the Layer Normalization, the Group Normalization, and the Weight Normalization. Labeling these normalization layers as unconditional as they do not depend on external data in contrast to the conditional normalization layers.
2. Conditional normalization layers include the Conditional Batch Normalization (Conditional BatchNorm) and Adaptive Instance Normalization (AdaIN). Different from the earlier normalization techniques, conditional normalization layers require external data and generally operate as follows.
   i. layer activations are normalized to zero mean and unit deviation.
   ii. The normalized activations are denormalized by modulating the activation using a learned affine transformation whose parameters are inferred from external data.

SPADE applies a spatially-varying affine transformation, making it suitable for image synthesis from semantic masks. Both methods are built on spatially adaptive modulation layers that condition on semantic inputs.

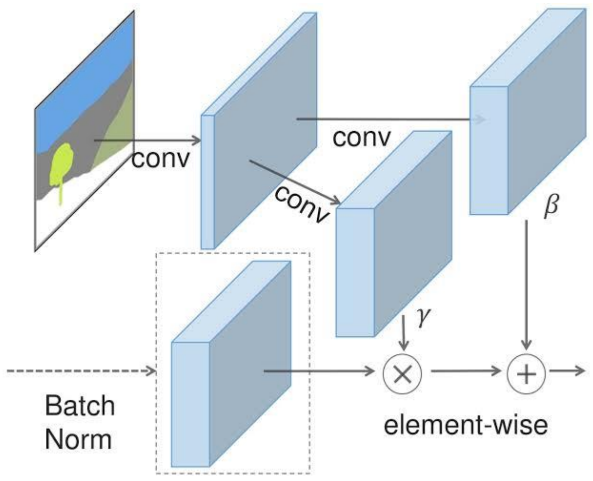

Figure 4.1: In the SPADE, the mask is first projected onto an embedding space and then convolved to produce the modulation parameters γ and β. Unlike prior conditional normalization methods, γ and β are not vectors, but tensors with spatial dimensions. The produced γ and β are multiplied and added to the normalized activation element-wise.

Let h^i denote the activations of the i^{th} layer of a deep convolutional network for a batch of N samples. Let C_i be the number of channels in the layer. Let H_i and W_i be the height and width of the activation map in the layer. Conditional normalization method called the SPatially-Adaptive (DE)normalization1 (SPADE). Similar to the Batch Normalization, the activation is normalized in the channel wise manner and then modulated with learned scale and bias.

Figure 4.1 illustrates the SPADE design. The activation value at site ($n\epsilon N, c\epsilon C^i, y\epsilon H^i, x\epsilon W^i$) is

$$\Upsilon^i_{c,y,x}(m)\frac{h^i_{n,c,y,x}-\mu^i_c}{\sigma^i_c}+\beta^i_{c,y,x}(m)$$

where $h^i_{n,c,y,x}$ is the activation at the site before normalization and μ^i_c and σ^i_c are the mean and standard deviation of the activations in channel c

$$\mu^i_c=\frac{1}{NH^iW^i}\sum_{n,y,x}h^i_{n,c,y,x}$$

$$\sigma^i_c=\sqrt{\frac{1}{NH^iW^i}\sum_{n,y,x}\left(\left(h^i_{n,c,y,x}\right)^2-(\mu^i_c)^2\right)}$$

The variables $\Upsilon^i_{c,y,x}(m)$ and in $\beta^i_{c,y,x}(m)$ (1) are the learned modulation parameters of the normalization layer. In contrast to the BatchNorm, they depend on the input segmentation mask and vary with respect to the location (y,x). Symbol $\Upsilon^i_{c,y,x}(m)$ and $\beta^i_{c,y,x}(m)$ to denote the functions that convert m to the scaling and bias values at the site (c,y,x) in the i^{th} activation map. Implement the functions $\Upsilon^i_{c,y,x}(m)$ and $\beta^i_{c,y,x}(m)$ using a simple two-layer convolutional network.

SPADE is related to, and is a generalization of several existing normalization layers. First, replacing the segmentation mask m with the image class label and making the modulation parameters spatially-invariant, arrive at the form of the Conditional BatchNorm. Indeed, for any spatially-invariant conditional data, our method reduces to the Conditional BatchNorm. As the modulation parameters are adaptive to the input segmentation mask, the proposed SPADE is better suited for semantic image synthesis.

SPADE Generator: With the SPADE, there is no need to feed the segmentation map to the first layer of the generator, since the learned modulation parameters have encoded enough information about the label layout. Discarding encoder results in a more lightweight network. Furthermore, similarly to existing class-conditional generators, the new generator can take a random vector as input, enabling a simple and natural way for multi-modal synthesis.

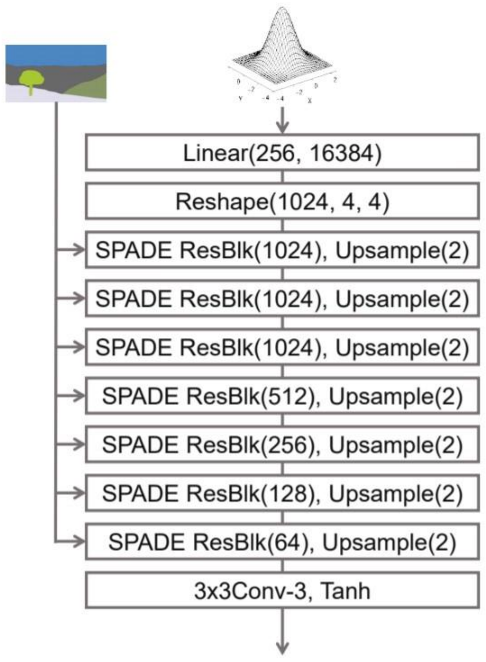
Linear(256, 16384)
Reshape(1024, 4, 4)
SPADE ResBlk(1024), Upsample(2)
SPADE ResBlk(1024), Upsample(2)
SPADE ResBlk(1024), Upsample(2)
SPADE ResBlk(512), Upsample(2)
SPADE ResBlk(256), Upsample(2)
SPADE ResBlk(128), Upsample(2)
SPADE ResBlk(64), Upsample(2)
3x3Conv-3, Tanh

Figure 4.2: SPADE Generator

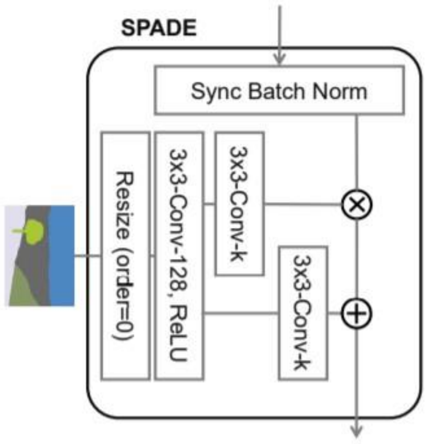

Figure 4.3: SPADE Design. The term 3x3-Conv-k denotes a 3-by-3 convolutional layer with k convolutional filters. The segmentation map is resized to match the resolution of the corresponding feature map using nearest-neighbour down sampling

CONCLUSION

We have proposed the spatially-adaptive normalization, which utilizes the input semantic layout while performing the affine transformation in the normalization layers. The proposed normalization leads to the first semantic image synthesis model that can produce photorealistic outputs for diverse scenes including indoor, outdoor, landscape, and street scenes. We further demonstrate its application for multi-modal synthesis and guided image synthesis.

REFERENCES

[1]. Taesung Park, Ming-Yu Liu, Ting-Chun Wang, Jun-Yan Zhu, "Semantic Image Synthesis with Spatially-Adaptive Normalization", 2019. URL: https://arxiv.org/pdf/1903.07291

[2]. Mehdi Mirza, Simon Osindero, "Conditional Generative Adversarial Nets", 2014. URL: https://arxiv.org/pdf/1411.1784.pdf

[3]. Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta and Anil Bharath, "Generative Adversarial Networks: An Overview", 2017. URL: https://arxiv.org/pdf/1710.07035.pdf

[4]. Saad AL-ZAWI, Saad ALBAWI, Tareq Abed MOHAMMED, "Understanding of a Convolutional Neural Network", 2017. URL: https://ieeexplore.ieee.org/document/8308186.

[5]. Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan C. Lawrence Zitnick, Piotr Dollar, "Microsoft COCO: Common Objects in Context", 2015. URL: https://arxiv.org/pdf/1405.0312.pdf