CSCI 1430 Final Project Report: Semantic Image Synthesis with Spatially-Adaptive Normalization

James TompGAN: Jeremy Chen, Steven Cheung, Jason Ho, Charlie Gagnon
Brown University
9th May 2020

Abstract

Our final project is an implementation of NVIDIA's GauGAN paper in Tensorflow (rather than PyTorch), using weaker computing resources. The paper proposes a novel spatially adaptive normalization for semantic image synthesis models. Since typical normalization loses some input semantic information, the proposed spatially adaptive normalization is designed to preserve semantic information by conditioning the normalization on the structure of each input segmentation map. Due to our limited computing resources, we were forced to omit certain aspects of the paper's architecture and reduce the resolutions of the generated images. Despite those constraints, we successfully trained a model that generates somewhat realistic images based on semantic maps of landscape scenes. Our code is available here and our dataset is available here.

1. Introduction

Generating photo-realistic images is useful both for directed tasks, such as image editing, and explorative tasks, such as automated content generation, and image generation has attracted recent research attention as a test of the limits of deep learning. Image generation is a very difficult task, because an image's geometry, shadows, texture, and light all must be reasonably realistic in order for the entire image to be "believable." In recent years, Generative Adversarial Networks (GANs) [1] have come to dominate image synthesis, producing very convincing images when trained on sufficiently large datasets. Additional improvements to the GAN paradigm for image generation, such as NVIDIA's GauGAN, are focused on classifying and preserving semantic information [3] to minimize the distortion of fine details.

We are particularly interested in photo-realistic generation from semantic input, modeled after NVIDIA's GauGAN paper [3]. The paper's use of Spatially Adaptive Normalization preserves semantic information that is lost with other normalization techniques (e.g. conditional Batch Normalization as in Pix2PixHD [6]). We implement NVIDIA's paper [3] using less computation and training time.

Our implementation is trained on select natural landscape images from the *ADE20K* dataset [7] which contains pairs of images and their labeled segmentation maps, with object labels sourced from WordNet. Our final project follows the architecture of the original NVIDIA paper, save for changes that were necessary reduce computation cost.

2. Related Work

Generative Models for Images GANs learn to produce new images using a generator and discriminator [1]. A generator tries to create photorealistic images in order to "fool" the discriminator, and the discriminator learns to distinguish real images (photographs) from generated images.

SPADE Normalization Spatially-Adaptive Normalization (SPADE) [3] is a conditional normalization that takes in an input and segmentation map and performs a learned transformation that makes the image more interpretable for training while preserving relevant semantic information. This learned normalization allows for much more realistic generated images from segmentation maps that have large sections composed of one particular object.

Semantic Image Synthesis Pix2PixHD [6] tries to generate photorealistic images from segmentation maps. The model is an improvement on the original Pix2Pix (another GAN), and uses a coarse-to-fine generator and multi-scale discriminator. NVIDIA's GauGAN [3] is another model that is able to produce realistic details on large areas from segmentation maps. Compared to other conditional GANs, GauGAN usually performs the best (in terms of FID score, which measures the similarity between generated images and their real counterparts [2]). Our project is based on NVIDIA's implementation of GauGAN, but is written with TensorFlow, rather than PyTorch.

3. Method

Data Preparation: We trained and tested models on a set of natural landscape images selected from the *ADE20K* dataset.

The original paper reported superior generated images from models trained on the entire *ADE20K* dataset, *ADE20K-outdoor* (a 5000-image subset of *ADE20K* that includes only outdoor scenes), and a set of landscape images collected from Flickr and segmented with a trained image segmentation network [3]. To isolate a smaller subset of *ADE20K* than *ADE20K-outdoor*, and to emulate the content of the Flickr-collected landscapes without producing segmentation maps ourselves, we qualitatively selected images from *ADE20K* by scene type (as labeled in the dataset). We then eliminated images that did not include at least three distinct objects from a list of landscape-like objects that we curated based on the content of the Flickr images.

Segmentation maps are fed to the model as one-hot encoded, three-dimensional image-vectors, with each image-dimensional slice encoding the presence or absence of a particular object from our selected list.

In order to decrease the complexity of the task that we presented to the model, we relabeled the segmentation maps in our selected subset of *ADE20K* so that image regions with objects that were not on our list of landscape-like objects were marked as regions with unknown semantic content.

In order to operate on the dataset, we wrote code to adapt the index of semantic information provided with *ADE20K* for Python. The dataset's provided semantic data is formatted in MATLAB, so we converted the semantic indices to CSV files, and made queries to those CSV files in order to isolate information about particular images and objects. The code to convert the MATLAB index files to CSV format is included in the projects GitHub repository.

SpadeLayer: As stated above, one of the largest changes GauGAN makes from previous architectures is the implementation of a spatially adaptive normalization. We implemented this by creating a custom Keras layer called SpadeLayer. Just like in the original paper, it first performs batch normalization on the input. Simultaneously, it performs two convolutions on the original segmentation map input creating a β and γ. The batch normalized input is then element-wise multiplied by γ and added to β (Figure 1).

SpadeBlock: We created another custom Keras layer called SpadeBlock, which is the main layer used in the generator. The layer mimics a ResNet block but uses SpadeLayer to normalize. (Figure 2)

Generator: To reduce computation cost, we did not use an image encoder. Instead, the first input to our generator is the image segmentation map and randomly encoded noise, which is passed through a convolution layer. This result is then passed through seven SpadeBlocks and upsampled five times in between (as opposed to seven in the original implementation [3]. Finally, this result is passed through a convolution layer which returns the generated image. Our generator uses two types of losses: first is a hinge loss on the discriminator logits produced from the generator's images.

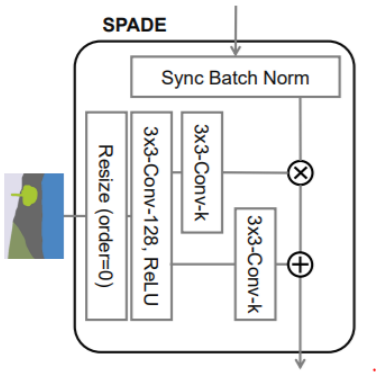


Figure 1. Architecture of a SPADE layer (from [3])

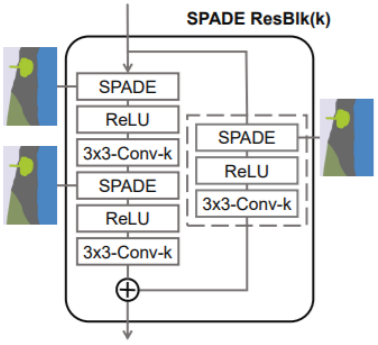


Figure 2. Architecture of a SPADE block (from [3])

The generator loss is a VGG loss which measures content and style differences between generated and fake images.

Discriminator: Our discriminator is fairly standard. First, it concatenates either the fake or real image with its segmentation map. Next, the concatenated object is passed through four convolution layers and one convolutional layer with one filter, representing the head of the discriminator. This produces a real or fake logit on each part of the image. The discriminator uses hinge loss on real and fake images.

Other Implementation Details: In convolutional layers for both the discriminator and generator, we applied spectral normalization, as implemented in [4]. In addition to the original NVIDIA paper's PyTorch implementation, we periodically compared our implementation structure to that of [5]'s.

4. Results

Our most accurate model was trained over 200 epochs with a batch size of 8, image size of 96 x 128, generator learning rate of 0.0001, and discriminator learning rate of 0.0004. The training set included approximately 1200 images and their corresponding segmentation maps.

Training Results: We tracked average Frechet Inception Distance (FID), average generator loss, and average discriminator loss per epoch. Our average FID started at around

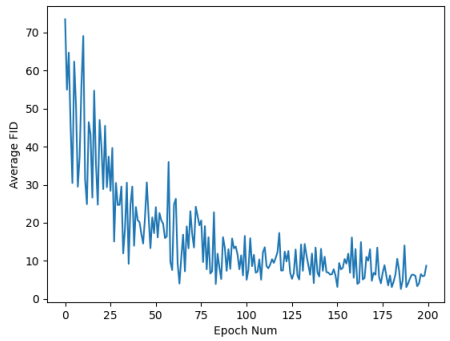


Figure 3. Average FID per Epoch

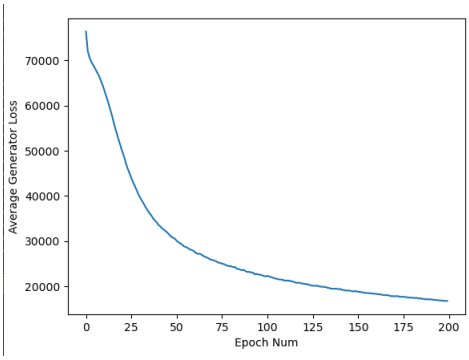


Figure 4. Average Generator Loss per Epoch

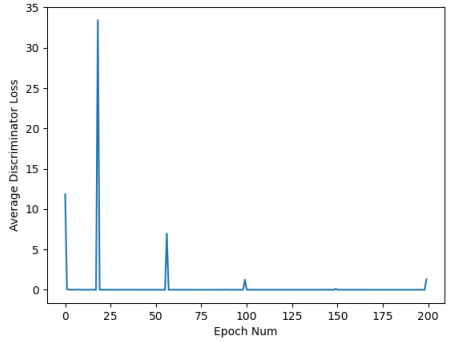


Figure 5. Average Discriminator Loss per Epoch

70 and stabilized around 5 (Figure 3). Our generator loss started very high at around 80k and stabilized around 12k. This high generator loss is due to weighting our VGG loss by 10 in comparison to hinge loss. Our hinge loss after 200 epochs is around 1. (Figure 4). Our discriminator loss started at about 12 but stabilized around 0.0001. (Figure 5). Figure 4 shows the generator output sampled every 40 epochs. By the end of the 200 training epochs, the images generated during training closely resemble the ground truth images.

Test Results: Generated images from our test set did noticeably worse, although most still closely resembled natural landscapes. Some generated examples preserved semantic information but failed on larger details. Figure 8 shows how the generator preserved a fence but failed in creating mountains or the snowy landscape. Figure 4 shows good test examples where most semantic information was preserved and the generated/ground truth images look similar. Figure 7 displays the avergae FID score that our model achieved on the test set, in comparison to the original NVIDIA implementation and other semantic image synthesis models. We note that, in terms of average FID score, our model performs better (on the small subset of *ADE20K* that we selected) than both the original NVIDIA GauGAN implementation and Pix2PixHD perform on *ADE20K-outdoor*. Of course, numerical comparisons between those models and our implementation are not fully appropriate, since we operate on much smaller images.

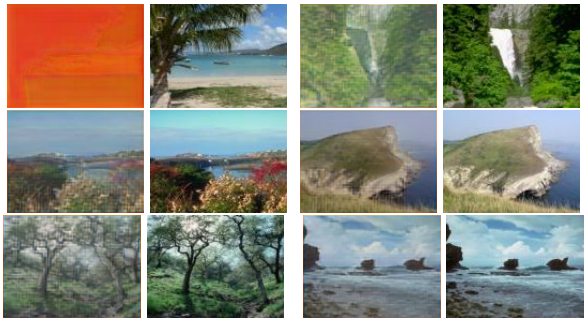


Figure 6. Training Output Every 40 Epochs. Generated Image on Left; Real Image on Right

Architecture	Our Dataset	ADE20K	ADE20K-outdoor
James TompGAN	60.77		
NVIDIA GauGAN [3]		33.9	63.3
pix2pixHD [6]		81.8	97.8

Figure 7. Comparison of average FID score during testing for various architectures and datasets

Figure 8. Example of Poor Test Performance (Generated on Left)

4.1. Discussion

4.1.1 Changes to Original Architecture

One of the major changes our team made was to reduce the image and batch size in order to decrease computation cost. Their implementation uses an image size of 2048×1024 and a batch size of 32, while ours uses 128×96 and a batch size of 8. The reduced image size meant there was less detail in the input images and the reduced batch size may have decreased the effectiveness of batch normalization. We believe this is the primary reason why our generated

Figure 9. Good Test Examples (Fake on Left)

pictures aren't entirely photo-realistic like in the original paper. We had to do reduce the scale of the task because we used a single NVIDIA GTX 1080Ti to train models, while the original paper used an NVIDIA DGX1 server with 8 NVIDIA Tesla V100 GPUs, which has vastly more computing power and memory than what we could access.

Another change was the exclusion of the image encoder which produced a the parameters for the sample noise for the generator. This would have allowed us to generated images with different styles while preserving high level features. This was mostly for, again, reducing the computation cost given the resources we had.

Overall, the changes we made to the original paper's model architecture were necessary tradeoffs between accuracy and computational resources, and we feel that we successfully implemented the techniques of the paper, but on a smaller scale.

4.1.2 Behavior on Excluded Objects

As we trained early models with a working architecture, we adjusted the list of landscape-like objects to include in the generated images. At first, we did not include some prominent objects, such as "sun." When generating images from segmentation maps that were missing prominent objects, the model was generally able to "remove" those objects from the scene, by filling regions of unknown semantic content with content similar to neighboring regions. This behavior (shown in Figure 10) suggests that the trained model learns to prefer adjacent image regions with similar content, and does not simply fill regions of unknown semantic content with noise. As a result, this model architecture is not likely to perform as well when input segmentation maps with unknown content have sharp variation in the appearance of the unknown content and adjacent known content. This effect makes the model severely limited by the precision of the semantic information it receives, since the model may not properly generate an object it "knows" if those objects occur next to regions with unknown semantic content.

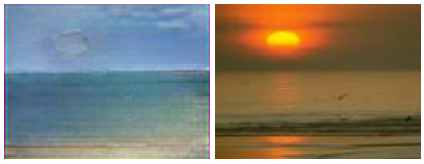


Figure 10. Generated beach image (at left) with the sun labeled as unknown content. We note the distortion of the sky near the location of the sun in the real image (shown at right).

5. Conclusion

Team Sentiments

We were able to partially recreate the results of NVIDIA's GauGAN paper. Our generated images had a clear resemblance of natural landscapes but did not have the same quality or resolution as the original paper's. Given that the original paper [3] uses 8 32 GB V100 GPUs while we used 1 11 GB 1080 TI, we believe that our results are very successful. Of course, we generate much smaller image sizes and our images have less photo-realism, but on many of the test images, the generated images look very good. By the qualified success of our small-scale emulation of the original NVIDIA paper, we believe that we demonstrate that contemporary techniques in deep learning are practically accessible without much research experience, and that computing resources (physical hardware and time) are the primary limitation on access to contemporary research findings. Working in this reduced architecture and still producing relatively good results shows that it is still feasible to reduce computation cost greatly without needing to sacrifice too much accuracy.

As a team, we found that this project underscored the importance of appropriate data selection in deep learning and machine learning generally. We spent significant time writing code to access the semantic information in the *ADE20K* dataset, because the scripts and informational indices provided with the dataset were written in MATLAB, while our project was implemented in Python. The choice of a more Python-friendly dataset may have offered us more time to adjust or extend the model architecture. Additionally, we found that hand-selecting a natural-landscape subset of *ADE20K* made the quality of our results quite sensitive to the objects we chose to include in the generated images. It would have been preferable to select a dataset with less scene diverity, or to arrange for more computational resources so that we did not need to limit the size of our training set so dramatically.

Project Implications

While the original NVIDIA paper reported more realistic generated images than other semantic image synthesis models, it is not clear that our implementation produced significantly more realistic images than other architectures

(such as Pix2PixHD [6]). This result may suggest that the benefits of SPADE normalization are marginal, and most significant only for models trained with sufficiently large computing resources. If so, our findings reinforce the idea that contemporary outcomes in deep learning (particularly for image generation) are limited more by hardware constraints than by the complexity of the techniques.

Appendix

Team Contributions

Jeremy Chen I focused mainly on writing the generator architecture. I also worked on the preprocessing of images and segmentation maps, such as converting the segmentation maps to a universal grayscale image that could be used for comparison and implementing one-hot encoding for better training, along with some debugging.

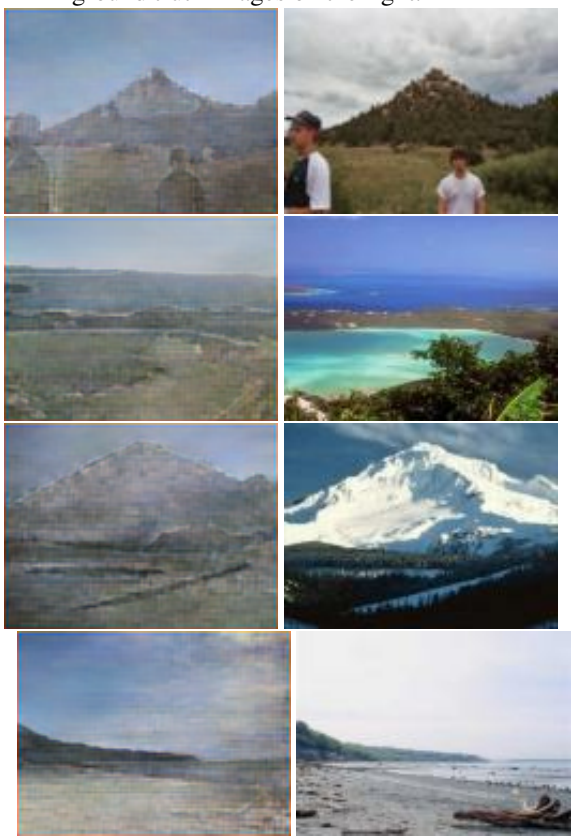
Steven Cheung : Focused mainly on writing the SpadeLayer and SpadeBlock Keras layers. Also worked on train() and test() in the main function and created a script to visualize FID and losses over time. Created a class to implement VGG loss and wrote a custom convolution function that was compatible with spectral normalization. Finally, worked on general debugging which included most portions of the code.

Jason Ho : Originally I worked on preprocessing the *ADE20K* dataset [7] to explicitly grab natural imagery scenes and scale down the images and segmentation maps to the correct size. In addition, I wrote the preprocessing script that loads the dataset with Charlie and Jeremy into the trainer.I then worked on writing the discriminator Keras model for the GANs. I spent most of my time debugging all of the code as we put the parts together and writing scripts to debug those parts. Of the time spent training, I tuned the hyper parameters to produce good results.

Charlie Gagnon naïvely agreed to work on processing the *ADE20K* semantic data in order to access each image's object content. This was very frustrating to do, but it got done. After converting the MATLAB data into CSV format, Charlie worked on selecting images from *ADE20K* by object content. The approach was ultimately combined with Jason's approach (selecting images by scene type) to produce the final selection process: selecting all images from handpicked scene types, and filtering out images that did not contain at least three distinct objects from a hand-designed list of landscape-like objects. Charlie then worked on formatting the images and encoding the segmentation maps as they were fed into the model. Charlie assisted with debugging all parts of the model architecture.

Additional Images Generated on the Test Set

Generated images are on the left, with corresponding ground truth images on the right.



References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014. 1

[2] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2017. 1

[3] T. Park, M. Liu, T. Wang, and J. Zhu. Semantic image synthesis with spatially-adaptive normalization. *CoRR*, abs/1903.07291, 2019. 1, 2, 3, 4

[4] taki0112. Spectral normalization-tensorflow, Aug 2018. 2

[5] taki0112. Spade-tensorflow, Aug 2019. 2

[6] T. Wang, M. Liu, J. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. *CoRR*, abs/1711.11585, 2017. 1, 3, 5

[7] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5122–5130, 2017. 1, 5