Generating Architecture Designs through Image Blending

Lan, Ting-Yu*
Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332
tingyu.lan@gatech.edu

Mou, Yingjun*
Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332
ymou32@gatech.edu

Abstract

The goal of this project is to adapt image-blending techniques into generating architectures. We aim to generate image content between two given architect images. The generated middle part should be realistic and seamlessly connect to two distinctive photos. We construct a novel dataset for the task, show reasonable results through our four approaches, and provide analysis.

1. Introduction

Wide-range image blending is a novel task proposed by [2]. It aims to generate a middle context between two very different images to create a smooth panorama image. It is closely related to but more challenging than image inpainting and outpainting. It requires a generated wide-range middle image that is reasonable, realistic, and smoothly transit between two very different random paired input images.

We aim to generate architecture design through wide-range image blending techniques. Previously, such tasks used landscape photos with relatively few rigid boundaries. Generating architecture design becomes a more challenging task because it needs to smoothly transfer context with many sharp edge lines and keep pertinent architecture details.

Our work applies image generation and composition techniques to areas that require more details and boundary processing. This will help to apply this technology to more scenarios. As mentioned in [2], an application would be generating interesting panoramic images from photos simultaneously taken by the front and rear camera of a cellphone.

2. Motivations

We expect the context prediction to be an upstream task that can be applied to multiple applications.

The problem of context generation can be formulated as given image I_{left} on the left, image I_{right} on the right, generate image $\widetilde{I}_{mid}$ in the middle, which can best blend the image I_{left} and I_{right}, forming an image which does look not only natural but also have meaningful semantics. However, depending on the contents of inputs image I_{left} and I_{right}, the results can be used for different purposes, such as skyline generation, architecture image reconstruction, and building transition generation.

(1) If the input is a pair of images of two different architectures in their completed shapes, then the task of predicting the middle ground can be considered as "skyline generation," which can be helpful in the movie production and the design of the urban environment in video games.

(2) If the input is a pair of images of the same architecture, i.e., the left image I_{left} is the 1/3 left portion of architecture, the right image I_{right} is the 1/3 right portion, then the context prediction task, in this case, can be considered as "architecture image reconstruction". When there are some incompleted historical images of architecture, context prediction can be helpful for historic preservation.

(3) If the input is a pair of images of a portion of two different architecture, then the context prediction task, in this case, can be considered as a "generative design of architectural transition."

While the motivation (2) can be done in a fully supervised way by keeping the cropped middle part of the original image (I_{mid}) as the ground truth, (1) and (3) are more generative and can only be done in an unsupervised way.

3. Related Works

To the best of our knowledge, [2], which was published in CVPR 2021, is the only publication dealing with wide-range image blending. Since it is highly related to image inpainting and outpainting, we will discuss these two fields first.

Image inpainting restores missing parts inside of an image. Image outpainting generates appropriate content based on a given image to extend its boundary to create a larger image. Both of them do not consider creating a context

*These authors contributed equally to this work

Figure 1. The formulation of problem and model structure, which combine both supervised and unsupervised learning

that can seamlessly transfer an input image to a very different one. The wide-range image blending task combines the techniques in these two fields. [2] uses a U-Net to encode and decode images, bidirectional LSTMs to generate images, and applies attention mechanisms to provide more details to the generated context.

4. Data

Our project constructs a novel dataset from extracting images from AIDA dataset [1]. AIDA dataset consists of 25 architectural categories with 14,659 architect images. We only collect and use the outdoor scene of the Apartment category. There are 2,474 architect images with size 256×256 in our dataset, and they are split into 1978/396/1000 for training/validation/testing separately.

5. Approach

Our work uses state-of-the-art image blending architecture [2] as our basis network. The model requires two-stage training, which is the self-reconstruction and fine-tuning stage. The self-reconstruction stage utilizes supervised learning first to learn high-quality images. It splits wide input training images into three parts $I_{left}, I_{mid}, I_{right}$, and learn to reconstruct the middle part I_{mid} of the input images. We stretch our training data into wider images to provide model smooth images with ground truth middle part. Input images are reshaped into 256×768 for training. For the Fine-tuning stage, the model uses paired data capture from distinct scenes to strengthen the model's ability to blend different images. We randomly paired our data without resizing them for this stage. No additional data preprocessing or post-processing is required.

It is worth mentioning that it is a large model that requires a considerable amount of computational resources. Using only 1/4 training epoch of the original paper takes more than 20 hours to go through the entire training process using Colab Pro. Since we only have two weeks, this is a huge problem we encounter. We carefully arranged our experiment schedule to face this problem.

The following are four approaches we have tried:

5.1. Adapt pre-trained model weights

We directly use pre-trained weight provided by [2] as our baseline. Since the weights were trained with scene images, we expect generated buildings to be distorted because there is no restriction on the structure of the building in the natural scenery training data. However, we guess it is still able to perform well in the remaining parts. For example, it should be able to blend the sky nicely.

5.2. Fine-tune with our own dataset

We tried to load pre-trained weight and only fine-tune the fine-tuning stage for the following two reasons. (1) Our dataset is small compared to the original work. We likely do not have enough data to train a good context generator (2) We do not have wide images. The data is 256×256. Our stretched wide images have lower resolution. To avoid the above two potential problems, we tried to fine-tune only the fine-tuning stage and compare it with other results.

5.3. Train the entire model with pre-trained weights

We also tried to train both self-reconstruction and fine-tuning stages, starting with loading pre-trained weight provided by [2]. The reason for loading pre-trained weights is that we expect it would help solve the problem of having a limited dataset. We expected our model to fine-tune well on architecture images with training through a smaller epoch number.

5.4. Train the entire model from scratch

We trained the entire model from scratch. The potential problem of training the whole process from scratch is mentioned in the above subsections. However, since the model contains adversarial loss, we guess training from scratch can avoid having discriminator being too strong at the beginning of the training to cause the entire training process to collapse.

5.5. Loss functions

The loss functions follow [2]. Therefore, we will just briefly introduce it here. Please refer to the original paper for more details.

The training objective contains four generator losses: pixel reconstruction loss, feature reconstruction loss, texture consistency loss, and feature consistency loss. It also includes a discriminator loss, which is adversarial loss. Reconstructions losses compute the similarity for each pixel/feature with ground truth; Consistency loss maintains smoothness for neighboring image pixels; Adversarial loss strengthens the generator ability through learning to fool the discriminator.

Not everything we tried at the beginning worked as expected, but we managed to improve the results by incre-

lr	ratio of update intervals of discriminator and generator	batch size	num of epochs	alpha	random pair inputs
2e-3	2:1	28	200	0.1	False

Table 1. Default hyper-parameters

Changed Hyper-parameter	Discriminator	Generator				
	Adversarial Loss	Adversarial Loss	Pixel Reconstruction Loss	Texture Consistency Loss	Feature Reconstruction Loss	Feature Consistency Loss
lr=1e-2↑	4048	2.631	0.856	0.210	1243	2409
interval ratio=10:1↑	0.603	3.5e-3	0.027	0.084	7.65e-7	1.16e-4
rand_pair=False	0.144	3.56e-3	0.024	0.081	3.08e-7	4.67e-6

Table 2. The (training) loss of the discriminator and generator with different hyper parameters.

mentally overcoming the several challenges (see section 6 for more details).

6. Experiments and Results

During our experiments, we identified and overcame the following challenges:

6.1. Balance of Discriminator vs Generator

During the training, we found that while the discriminator loss decreased rapidly, the generator loss didn't have a significant decrease. Our first intuition is that the discriminator is trained too actively and thus becomes dominant. When the discriminator is too strong, no matter how the generator generates images, it fails to "fool" the discriminator. Thus, failing to be trained effectively.

We tried to tweak the interval between two different gradients backward to address this problem. The original model updated the weights of discriminator every 3 iterations iter-idx % 3 == 0) and updated the weights of generator when iter-idx % 3 != 0, having an update interval ratio of 1:2. We first decreased the ratio to 1:5, then to 1:10. We found that although this modification results in the discriminator loss decreased at a slower speed, the generator loss still failed to decrease significantly.

To be more specific, this happened in the second stage, fine-tuning stage. Therefore, we have also tried to initiate the training with different weights, including our own model weights by training the first stage from scratch, our own weights by training the first stage with pre-trained weight, and directly loading pre-trained weight provided by [2]. Unfortunately, we are still not able to reduce the adversarial loss.

6.2. Mode Collapse

Since tweaking the update interval of generator and discriminator didn't result in the expected decrease of generator loss, we suspect this may be caused by mode collapse. Mode collapse happens when the generator can only produce a single type of output or a small set of outputs. This may occur because the generator finds a type of data that can easily fool the discriminator and thus keeps generating that one type. Because the generator has no incentive to switch things up, the entire system will over-optimize on that one output.

To address this problem, we tried to add a Gaussian blur on the images generated by the generator before they were passed into the discriminator. We found that by doing this and tuning the hyperparameters, especially the learning rate, the generator loss started to decrease as expected.

7. Experiments and Results

To minimize the loss of both discriminator and generator, we have tried different sets of hyperparameters starting from the default values given by the original paper (Table 1). The results are shown in Table 2.

7.1. Settings

All of our experiments were done on Colab. We upgraded to Google Colab Pro to access better GPUs and longer usage time. The entire code was written in PyTorch. We started with code provided by [2], which can reproduce the result they claimed in their paper. Then we modified it to deal with the challenges mentioned above results from having different goals with the original paper, fitting our dataset, and adjusting to the Colab environment.

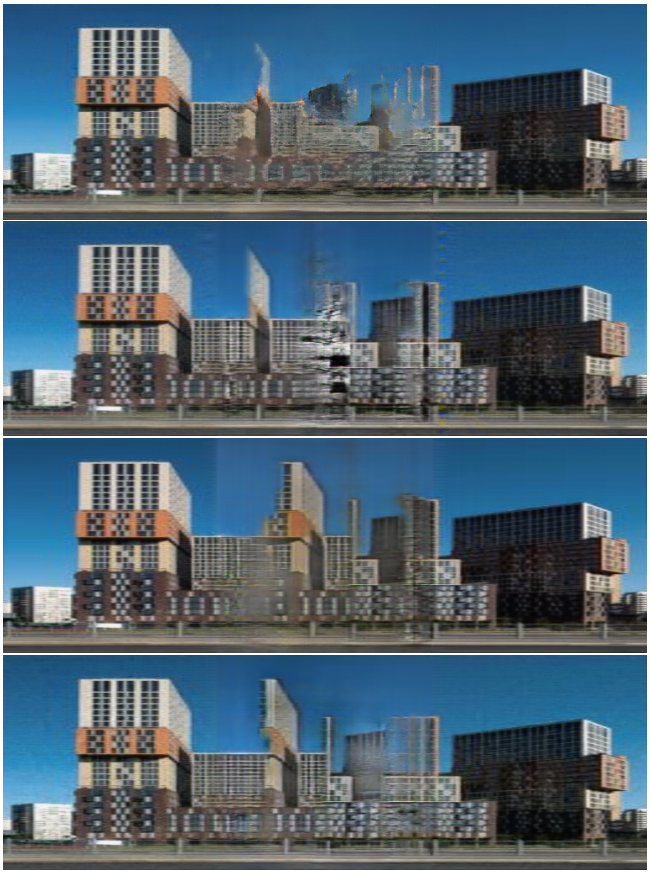

Figure 2. Sample results of 5.1, 5.2, 5.3 , and 5.4 in order

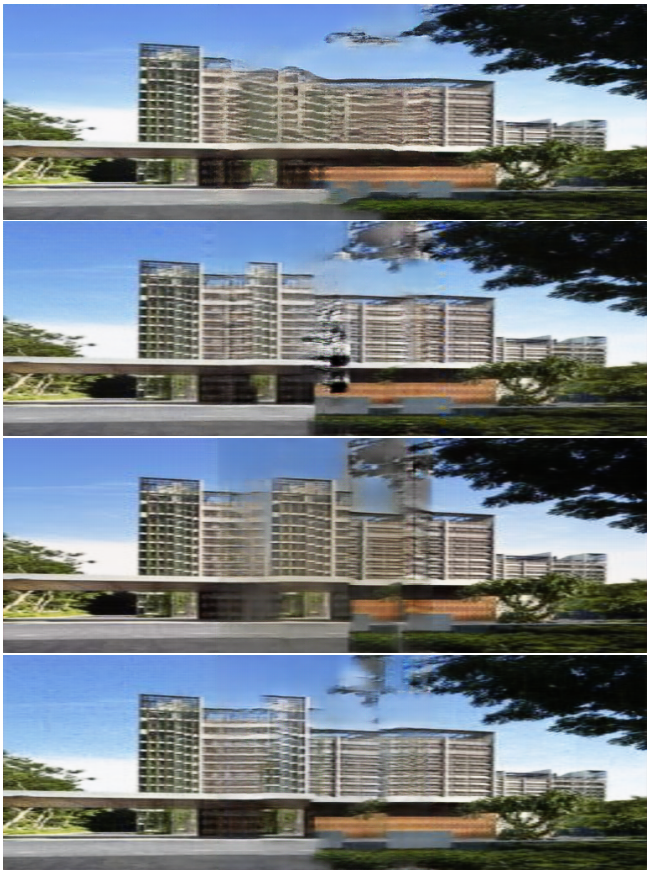

Figure 3. Sample results of 5.1, 5.2, 5.3 , and 5.4 in order

7.2. Results

This is an image-generating task. Therefore, directly looking at generated images is a critical way to measure the work's success. Figure 2 and Figure 3 show our results for four approaches with same input pairs. The results show that training the entire process, whether the pre-trained weights are loaded, performs better. It transits from one input image to another more smoothly, having more architecture details and having boundary lines of building more reasonable.

Moreover, we have shown interesting results generated from directly using pre-trained weight 5.1 in Figure 4. Though we input a pair of architecture with no mountains, the model tries to generate a mountain. The phenomenon is due to the reason that the training data contains lots of mountains.

Another way to measure success is through loss values. The lower the loss value, the better our model achieves the objective of reconstructing, generating smooth content and fooling the discriminator. Please refer to table 2 for quantitative results.

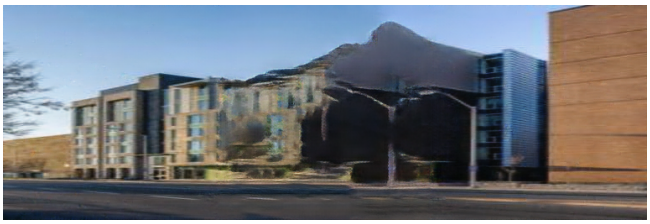

Figure 4. A sample result of 5.1. The model generates a mountain while there are no mountains in the input pairs

7.3. Result Analysis

7.3.1 Perspective distortion

The natural scene images, especially those wide and open scenes such as mountains, usually don't have a strong vanishing point and perspective distortion. Ont the other hand, architectural images have smaller scales in which the vanishing points can easily distort the lines and proportions.

Thus, while our model performs well on the original landscape dataset, the qualities of its results on architectural images vary. We found that while our model performed well on those street views (flat elevation side views without obvious vanishing point), it performed relatively worse

on those images with a vanishing point and perspective distortion, see Figure 5. This aligns with our intuition that recognizing the "vanishing point" and reconstructing an image with "depth" is harder than filling in the street views by doing copy-and-paste. And we think this may be because the current convolutional layers in our GAN model could not extract the semantic information about vanishing point and perspective distortion due to its limited perceptive field.

7.3.2 Self-copying

Another pattern we found is that the model tends to copy a patch of the input images and use it to do the inpainting, as Figure 6. Similar to the problem of mode collapse, the model was trained to think that "blending by copying" can fool the discriminator for the majority of data samples.

While it is true that doing self-copying works well for side views where there is a consistent texture, colors, and degree of distortion, it cannot be used to deal with views with depth and vanishing points. To address this issue, we propose to modify the loss function in future works. By doing so, an invalid vanishing point generated by the model will be penalized.

7.3.3 Hue transition

While the model can achieve descent results of shape and line blending, it did not do quite well in color blending, especially when there is a significant hue difference on the two sides. The generated result has an abrupt change of hue generated in between.

This usually happens when the two input images are taken during different seasons or different times of the day. As the below examples show Figure 7, the bright color of the sky on the left and the darker color on the right collided with each other without having a smooth transition.

We think this issue is that while we have texture consistency loss to guarantee the smooth transition of texture at the pixel level, we have not defined a "hue consistency loss" to penalize the abrupt change of hues.

7.3.4 Foreground vs background

One of the differences between the natural scene images and architectural images is that while orthogonal straight lines are rare in natural scenes, they are common in those images of man-made architectures. And while there are contrast differences between foreground and background in the architectural images, they are subtle in those natural scene images.

We found that using the model pre-trained on natural scene images tends to blend the images by blurring the two sides with dissolved water-color texture and organic lines. On the other hand, using our model trained on architectural images from scratch tends to blur the photos by copying a patch from input and rescaling and shifting. See Figure 8.

This demonstrated that our re-trained model has successfully learned the distribution of architectural input images and become "aware" that self-copying and extending the

Figure 5. Comparison between side views and views with vanishing points

Figure 6. An example of "self-copying"

Figure 7. example failure of hue transition

lines are better solutions to the pictures with depths. As shown in the example below, the original pre-trained model did not learn to distinguish depth of field, thus mistakenly blurring the building in the foreground with the mountain in the background.

Figure 8. results from pretrained model vs. results from re-trained model

References

[1] Jielin Chen. Annotated Image Database of Architecture (AIDA), 2020. 2

[2] Chia-Ni Lu, Ya-Chu Chang, and Wei-Chen Chiu. Bridging the visual gap: Wide-range image blending. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 843–851, 2021. 1, 2, 3