DEEP LEARNING PROJECT: GAN ARTWORK

A PREPRINT

Kevin Steele
Department of Electrical and Computer Engineering
University of Iowa
kevin-steele@uiowa.edu

Brain Fiegel
Department of Industrial and Systems Engineering
University of Iowa
brian-fiegel@uiowa.edu

Stjepan Fiolic
Department of Electrical and Computer Engineering
University of Iowa
stjepan-fiolic@uiowa.edu

May 10, 2021

ABSTRACT

Generation of landscape images from an input of deoxyribonucleic acid (DNA) is an intriguing and challenging topic for deep learning. The idea that an individual's DNA sequence would be able to produce an environmental landscape sounds impractical, but in this paper, we show working proof that it is possible. To achieve this feat, a generative adversarial network (GAN) and a GAN variation stack GAN was used to produce images of landscapes with just the input of DNA after the model was built. StackGAN has a stage-I and a stage-II. The success we recorded in the paper came from the stage-I with stage-II falling short of our desired target.

Keywords Deep Learning · GAN

1 Introduction

Generative Adversarial Networks (GANs) are an approach to generative modeling using deep learning methods. Generative models are an unsupervised machine learning task. This task involves learning and discovering input data patterns in a way that the model can generate new output data that has a close relationship to the original input data. For example, if we were to feed the model with images of the human face, the model would generate a new human face that is not in the original dataset. This may be an example most people are familiar with, but models can also convert natural language text descriptions into images. This type of model is called a Text-to-Image synthesis issue. We aim to achieve a similar goal by feeding a valid DNA sequence into our model and output a landscape image.

To understand GANs, the reader should first note that there are two sub-models of a GAN. The first is the generator model. The generator model takes a fixed-length random vector as input and generates a sample in that domain. A gaussian distribution is used to create a randomized vector. That vector then becomes the seed of the generative process. The training process will then produce latent (hidden) variables to input into the generator, the collection of which is referred to as a latent space. The second model of a GAN is the discriminator model. The discriminator model takes an example from the domain as input and predicts a binary class label. This label is either authentic or generated. The authentic examples come from the training dataset, in our case images of landscapes. The generated examples come from the output of the generator model.

The best way to distinguish the two model types is to think about them as a tug of war. One model is trying to build a landscape image (generator) great enough to fool a human (discriminator) into thinking that it is a real landscape. The generator must make the landscapes as close to real landscapes while the discriminator will point out flaws in the generated image and send it back. At convergence, the generator will have samples that are indistinguishable from the real data, or training set.

Given our particular object of Text-to-Image synthesis, we will require a specialized GAN architecture. A few architectures have been proposed in this realm, particularly StackGAN [1] for text-to-image and CycleGAN [2] for unpaired image mapping. This process can be used for text-to-image as well [3], which involves using two generators, one mapping text to the generated image, then another doing the reverse with an image mapping to a generated text. A more recent attempt at this is MirrorGAN [4], which uses a stacked generator architecture to generate an image and an auto-encoder to regenerate semantically similar text. In examining the architectures, we decided to work with StackGAN. This architecture has two stages: stage-I GAN and stage-II GAN. Stage-I GAN does a rough estimation sketch of the text input provided. For a mountain landscape this would be the equivalent of having mountains distinguishable from sky but not being able to distinguish individual trees or different mountain faces. Stage-II uses the same text input that stage-I has and the low-resolution output image of stage-I as an input. This stage is where the details are included, like being able to distinguish mountain faces and trees.

Real world applications that we discussed include a provided digital image of a landscape generated based on the participant's DNA. Although this might be a neat souvenir for the participant, some privacy concerns arise with participant DNA structure being transmitted over an unsecure connection. Also, the image would be a random conglomeration of DNA, but a clever enough person might be able to reconstruct participant data with the correct tools.

2 Problem Definition

Our goal is to convert a valid DNA input sequence into a landscape image using a Generative Adversarial Network (GAN). Examples of these can be seen in the Data section 3. This is a Text-to-Image synthesis issue, with the generator taking valid DNA codons (TCAG) as input and creating landscape 224 by 224 by 3 (RGB) images as output. Image values range within [0,255] across the 3 color channels. Similar to a more general Text-to-Image GAN, which would take text descriptions to generate images, the DNA codon sequence will be a translated language by which we generate landscapes. Valid DNA codons follow the standard generic code table with valid inputs of T, C, A, and G in groups of 3. Each sequence must also start with a start codon and end with an end codon. Noted in the original table, there are 25 different amino acids that these codons correspond to. While it is not currently our goal, this fact could be used similarly to an alphabet, and thusly a sentence or description of the image.

Looking more comprehensively at the GAN, the generator takes as input valid codon, considered the latent space of the system, and outputs the image we desire. The discriminator would take as input an image, either directly from the dataset or from the generator, and return the probability that the image is real (direct input) or fake (generated input). It is obvious that, generally, DNA does not map to images in the real world. For our purposes, we currently create input to the generator, the latent variables, by converting the DNA sequence to a normalized average value between [0, 1]. Specifically, each character is represented by a number between 0 to 3 (ACGT respectively). These numbers are averaged by taking the sum of them and dividing by the length of the sequence and the length of a codon (3). This average is what we record, and thus we convert the variable length string into a single floating-point feature. This is done to create 128 latent variables, thus creating a 128-dimension latent space. Finally, the values are normalized to [0,1], finishing our noise generation. We played around with using text embeddings using k-mer counting [1] and Sequence Graph Transformation [2], but ended up leaving this out as it would not conform to the latent space defined. It is an area with room for exploration, however. As for the landscape images, all images vary in resolution, many of which are far too large to train on, and thus must be down-sampled into their 224 by 224 format. This is handled by keras preprocessing when loading the dataset into training and testing sets.

3 Data

The datasets used within this project are a Landscape dataset obtained from Kaggle and a DNA dataset obtained from Joint Research Center. They can be found at their respective pages:

`https://www.kaggle.com/arnaud58/landscape-pictures`
`https://data.jrc.ec.europa.eu/dataset/2abb5c2b-3ab6-4ce4-b103-cb1c5fc7349e`

3.1 Landscape Dataset

The landscape dataset contains a variety of non-descriptive landscape images. There are no classifications or metadata for any image. All images start as JPEG images of varying sizes, but are pre-processed to be 224 by 224, as can be seen

[1] Method of counting substrings in DNA codons

[2] Sequence embedding function. Extracts the short- and long-term sequence features and embeds them in a finite-dimensional feature space. Reference:

Table 1: DNA Formatting

ID	Name	Description	Number of Features	Per letter annotation for:	Sequence

in Figure 1. The images used were scraped from Flickr images, which appears to present some issues of dirtied data. For instance, we found the data is skewed toward ocean and mountain images, and there also contains a few images of selfies and posters (images of images). This within itself would cause issues with training the data, as the dirty data could cause the model to learn unnecessary things. However, the dataset is viable in its current iteration and does create a valuable model.

Figure 1: Pre-processed images

3.2 DNA Dataset

The DNA dataset contains various sequences of DNA codons. Collected by the Joint Research Center by sequencing genetically modified strains of bacillus subtilis used for production of vitamin B2 in feed additive.
Data is formatted in a *.fastq* file with the structure found in Table 1.

4 Method

Here we will discuss the methodology of employed on the project. All code was developed to be run on Google Colab using Jupyter Notebook formatting for Python. Code can be found at `https://github.com/KaiwenS7/DeepLearningProject`.

4.1 Relevant Equations

GANs optimize their networks around a minmax problem. The generator G is optimized to produce images that are difficult for the discriminator D to differentiate from real images. D is then optimized to know the difference between generated images and real images. This optimization comes by way of minmaxing the loss function. For our loss function, we use the binary cross entropy between the predicted labels and the true labels for the discriminator D and between predicted labels and fake labels for the generator G. Cross Entropy can be defined as:

$$H(p,q) = -\Sigma_i p_i log q_i = -y log\hat{y}\breve{}(1-y)log(1-\hat{y}) \tag{1}$$

where $p \in y, 1-y$ or the true examples and $q \in \hat{y}, 1-\hat{y}$ or the generated examples.

We can convert the loss function into a minmax problem for G and D, where we maximize the loss of D while minimizing the loss on G. This dual-optimization can be summarized with the objective function:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_z}[log(1-D(G(z))] \tag{2}$$

Where x an image from the real data p_{data}, and z is the noise from a Gaussian distribution p_z.

To evaluate images produced by our model, we use the Fréchet inception Distance (FID). This, like inception score, is used to evaluate the quality and diversity of a GAN's image production. FID in particular compares the distribution of generated images to the distribution of training or real images. This assumes that the two distributions are multidimensional Gaussians: $\mathcal{N}(\mu, \Sigma)$, the generated distribution and $\mathcal{N}(\mu_\omega, \Sigma_\omega)$ the real distribution. Using these, we can describe FID as:

$$FID = |\mu - \mu_\omega|^2 + tr(\Sigma + \Sigma_\omega - 2 * (\Sigma\Sigma_\omega)^{1/2}) \tag{3}$$

4.2 Implementation

We originally followed the StackGAN model as closely as was capable, implementing the neural networks of the stage-I GAN and the stage-II GAN without the conditional augmentation. In the original paper, conditional augmentation was used for word segmentation and sentence feature extraction, which is then fed into both stages. We do no such thing with DNA sequences. Both stages use an Adam optimizer with a Learning Rate of 0.0002 and a Beta of 0.5 for both the discriminator and the generator. We have also implemented a learning rate decay of 0.5 every 50 epochs.

In preparation for discussion of each stage, we will first define the blocks used to simplify explanation of the models. There are 2 repeating blocks used in building the GANs, the Up Sample block and Down Sample block. The Up Sample block involves an up sampling layer followed by a convolutional layer with a ReLU activation. The Down Sample block is a convolutional layer of at least stride 2, followed by a ReLU activation and a Dropout layer. Also as a note, all LeakyReLU activations use an alpha of 0.2, and all Dropout layers use a 0.2 dropout rate.

The stage-I generator takes an input with the latent space size of 128. This is fed into a dense layer and reshaped with the values (4, 4, 1024). This is fed through a series of 4 Up Sample blocks, each reducing the filter size by half. Finally we use a convolution layer with 3 filters and a tanh activation to obtain a 64x64x3 image. The discriminator follows a convolutional layer of 64 filters and a kernel of 4 using a LeakyReLU activation. Following this is 4 Down Sample blocks, each doubling the number of filters, and finally the discriminator ends with a dense layer of output size 1, the probability that the input sample is real.

The stage-II generator follows the output of the stage-I generator, taking the 64x64x3 image as input. Since we desire a 224x224x3 output, the input image must be resized to 56x56x3 using lanczos resampling [5]. This is fed through 2 Up Sample block of kernel 4 with filter sizes of 112 and 224 respectively, and the generator finishes with a convolutional layer of 3 filters and a tanh activation output. This provides a 224x224x3 image. The discriminator opens with convolutional layer of filter size 224, kernel 4, and stride 2, followed by a LeakyReLU layer. This is then fed through 3 Down Sample blocks of kernel 4 and filters 56, 112, and 224 respectively. Finally, a dense layer of output size 1 generates the discriminator's prediction.

4.3 Training

Training was done using a batch size of 32 and a loss of Binary Cross Entropy. We train stage-I using 100 epochs and stage-II with 300 epochs. We use a normally distributed noise of shape 128 as input to the stage-I GAN. The stage-II GAN training uses the normally distributed noise to obtain an image from the stage-I GAN (64x64x3 image vector), then inputs the image to the stage-II GAN. stage-II does not take any separate noise input as the image created is assumed to maintain the randomness of the noise. For both GANs, true labels are given a random uniform noise scaled by a factor of 0.2 in order to reduce the strength of the discriminator's learning. This is done in the hopes of stabilizing the GAN's loss convergence [6].

5 Results and Discussion

Here we will discuss the results of our testing and potential changes that could be done to further improve the model in future.

5.1 Evaluation

With our model, we saw similar trends with the loss of both stage-I and stage-II. As epochs increased, the loss for the discriminator exponentially decreased (see 2), and the loss for the generator linearly increased (see 3). Since these losses never level off, we do not appear to reach equalibrium with the models. If only the losses were considered, then the generator is getting much worse at creating new images, and the disciminator is out-pacing the generator's learning. We can see this trend by viewing the images themselves in Figures 4 and 5.

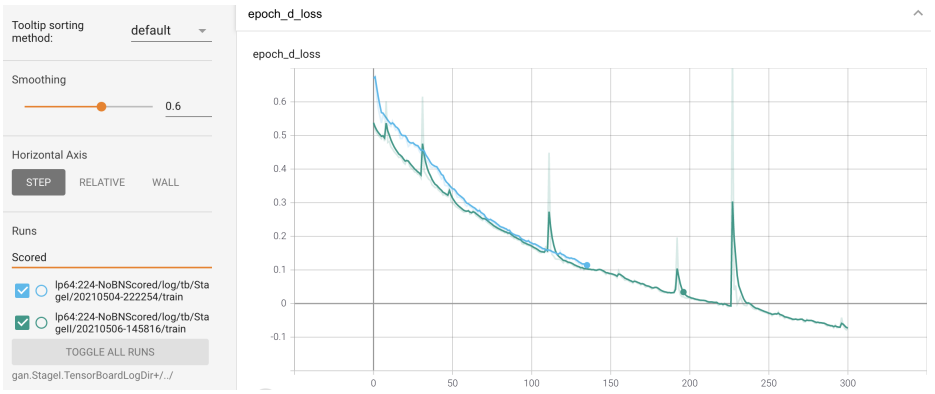

Figure 2: Stage-I and stage-II discriminator loss as provided by TensorBoard. The sky-blue line is stage-I and the dark green line is stage-II

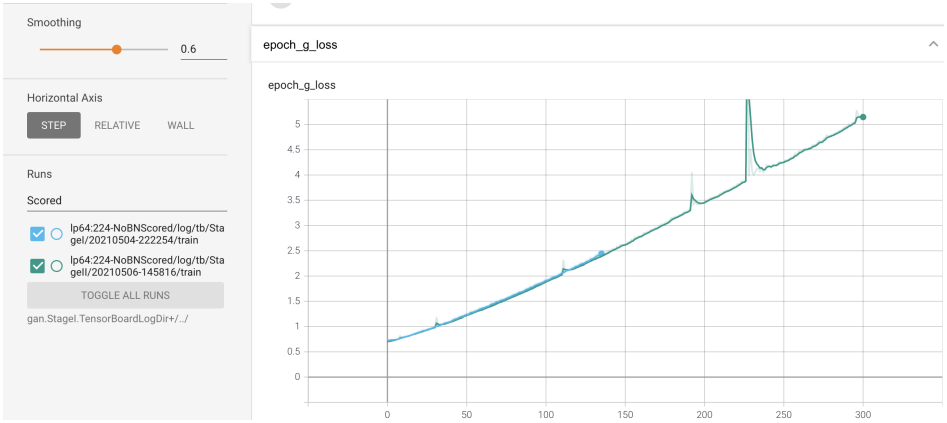

Figure 3: Stage-I and stage-II generator loss as provided by TensorBoard. The sky-blue line is stage-I and the dark green line is stage-II

As described in section 4.1, we use the FID as a qualitative measure for our GAN network. Generated images are compared to the images used to train the network for similarly, quality, and diversity. We used this measurement on multiple stages of epochs to obtain a more comprehensive look at the model's changes over time. These can be found in Table 2 for stage-II, and their corresponding image output can be seen in Figure 5. As can be seen from the table, the FID barely changes over time, meaning the images have been found to be the same quality and diversity from the source materials over all epochs abover 120. This means that there is little correlation with the loss, and that the generator is failing to change over more epochs. Stage-I was found to have it's best network at 100 epochs with an FID of 420, and it's images can be seen in Figure 4.

Stage-I tended to provide rather realistic images after about 100 epochs. However, we found that there appears to be less diversity and realism to the images as the model evolves. This tendency can be better observed in stage-II, though stage-II produces less realistic images overall. However, stage-II does show promise in that it is a realitively strong translation of the input image from stage-I. The issue lies with the colors creating less realistic images in general. It's difficult to say why the this is the case, but we can say that the generators are learning. Perhaps more model exploration might be required, but otherwise it is difficult to discern why the images cannot improve past around 300 epochs.

5.2 Discussion

To explain the issues further, there are too many different landscapes ranging from mountains, deserts, oceans, fields and many other filler landscapes. As you can see in Figure 5 and Figure 4. The images have many different landscapes mixed in with the output images. Some landscape types are favored more than others, for example, there are more images of ocean/coast and fewer images of desert. Along with too many differing landscapes, our data set also consists of images with people, animals, non-environmental objects, and modern infrastructure. There was discussion about

Table 2: stage-II FID for different Epochs

Epochs	FID
120	433
140	423
170	399
180	438
190	430
200	403
210	429
220	467
230	430
240	400
250	414
260	431
270	413
280	445
290	406
300	420

Figure 4: stage-I epoch 100 images. Corresponding FID is 420.

filtering the different types of images ourselves because the landscape data set that was used for training had no filter by type of images or classification of the images. There is a belief that if we classify only images of a particular feature set, like mountains, then the resulting images would have turned out environmentally natural. It would be a suggested course of action to change this dataset in future projects of the like.

We also do not have the fullest grasp of the loss. As mentioned in 5.1, the losses tend in opposite directions and never reach an equilibrium point. Given this, we would expect that the generated images get worse over time, which is an assessment we agree with. As time goes on, particularly for stage-II, the images become noticably discolored and less varied. To the human eye test, stage-II fails to learn and capture the features of a landscape image, even though the FID suggests the similarity of the images.

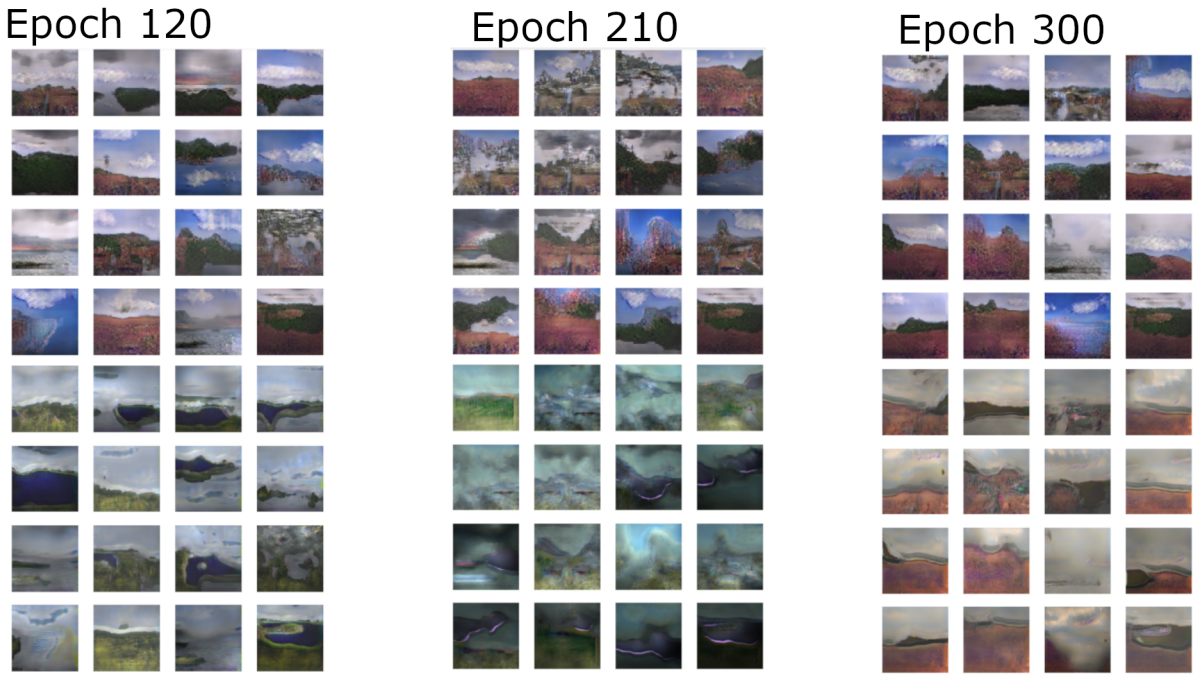

Figure 5: Stage-I and stage-II images organized by epoch. Top half of each image is the generated stage-I image (all done at epoch 100 for stage-I). The bottom half is the result of the top half image as input.

6 Conclusion

To summarize, we set out to generate landscape images from an input of DNA sequences. While we did not provide a perfect solution, we believe to have shown working proof that it is possible. To achieve this feat, a generative adversarial network (GAN) and a GAN variation stack GAN was used to produce images of landscapes with just the input of DNA after the model was built. StackGAN has a stage-I and a stage-II. Our results are well enough given the difficulty of the task, but a simpler model might suit better for this process. In the case that the StackGAN architecture were to be followed, there is room to grow with information extraction from the input sequence, such as k-mer counting and Sequence Graph Transformations.

References

[1] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. *arXiv preprint arXiv:1612.03242*, 2017.

[2] Zhu, Jun-Yan and Park, Taesung and Isola, Phillip and Efros, Alexei A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In IEEE International Conference on Computer Vision (ICCV), 2017.

[3] Trevor Tsue and Samir Sen and Jason Li Cycle Text-To-Image GAN with BERT *arXiv preprint arXiv:2003.12137*, 2020

[4] Tingting Qiao and Jing Zhang and Duanqing Xu and Dacheng Tao. MirrorGAN: Learning Text-to-image Generation by Redescription *arXiv preprint arXiv:1903.05854*, 2019.

[5] Lanczos Resampling. In *Wikipedia.* `https://en.wikipedia.org/wiki/Lanczos_resampling`

[6] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford and Xi Chen Improved Techniques for Training GANs *arXiv preprint arXiv:1606.03498*, 2016.

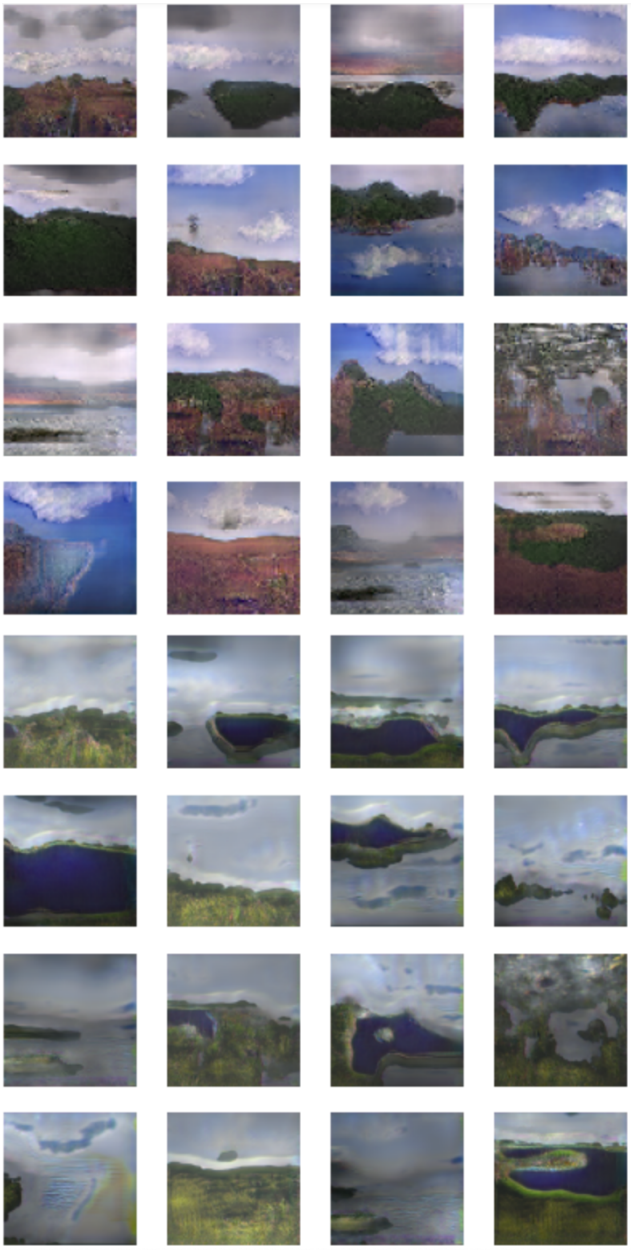

Figure 6: Stage-I and stage-II images for stage-II epoch 120

Figure 7: Stage-I and stage-II images for stage-II epoch 130

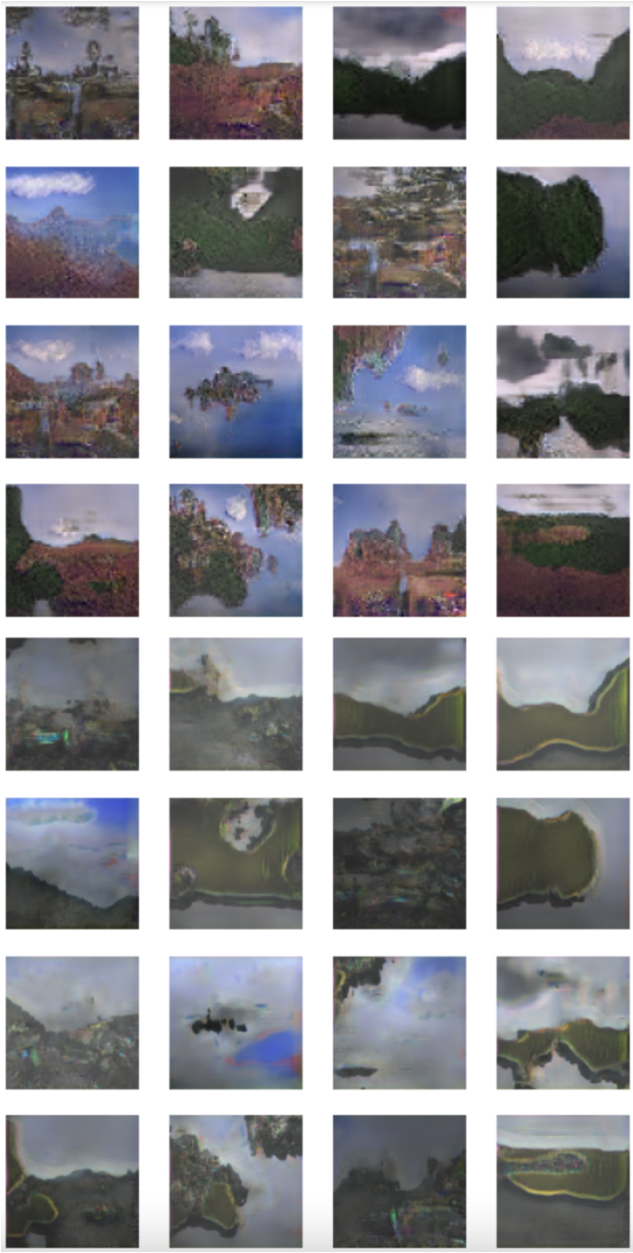

Figure 8: Stage-I and stage-II images for stage-II epoch 140

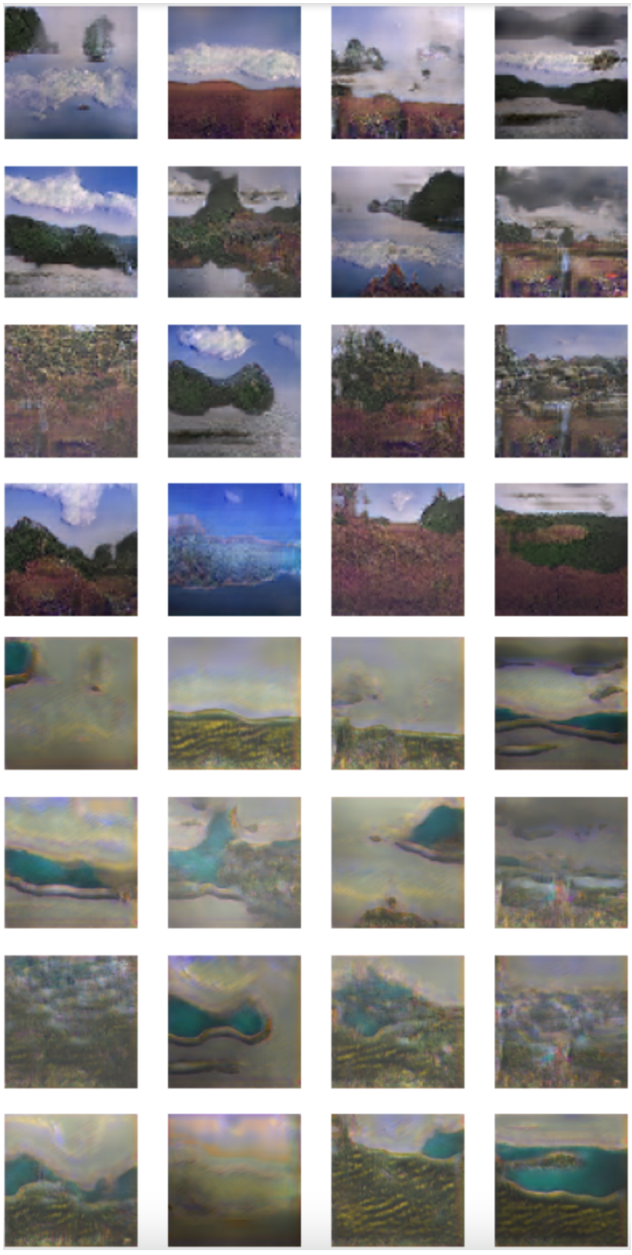

Figure 9: Stage-I and stage-II images for stage-II epoch 170

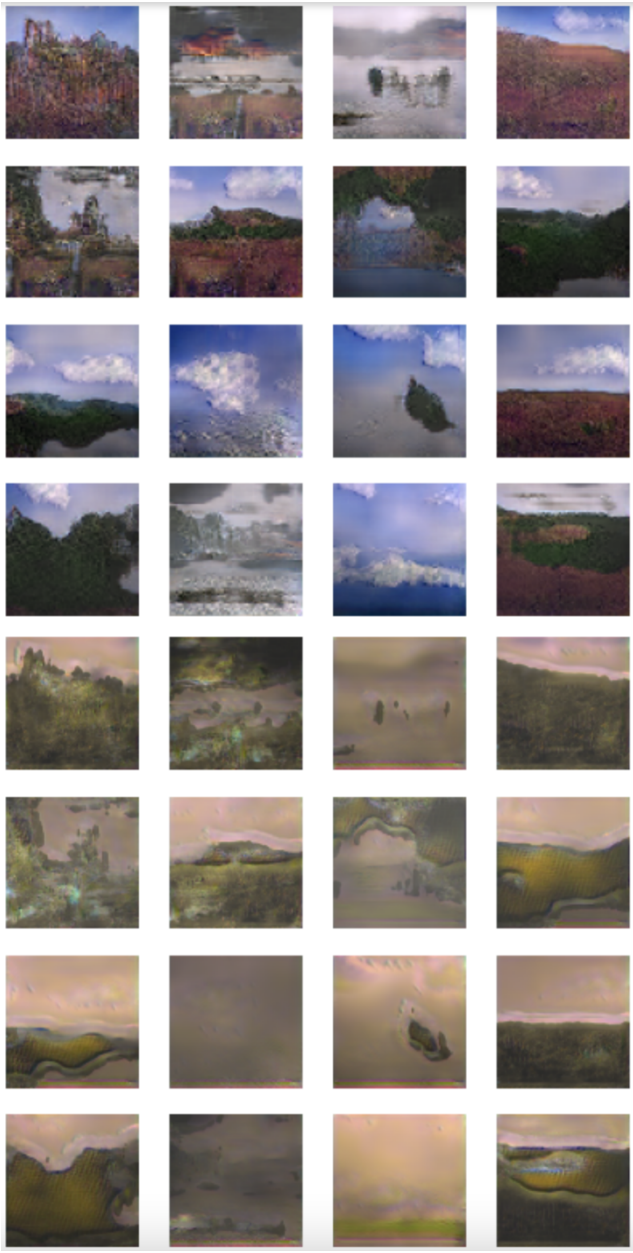

Figure 10: Stage-I and stage-II images for stage-II epoch 180

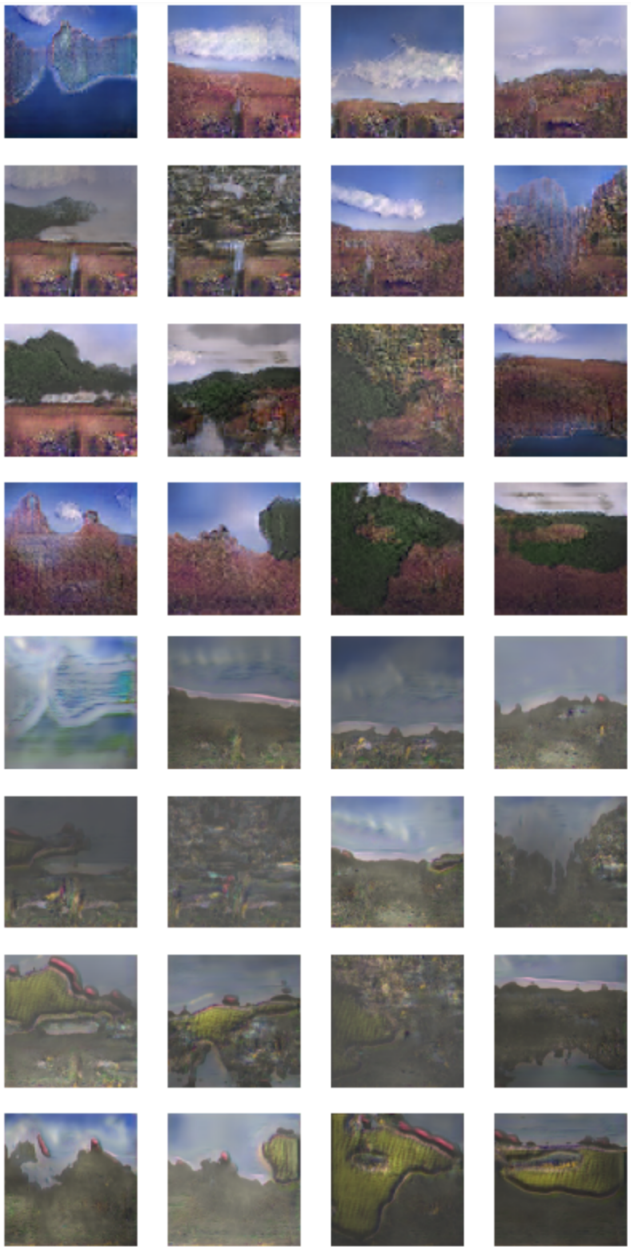

Figure 11: Stage-I and stage-II images for stage-II epoch 190

Figure 12: Stage-I and stage-II images for stage-II epoch 200

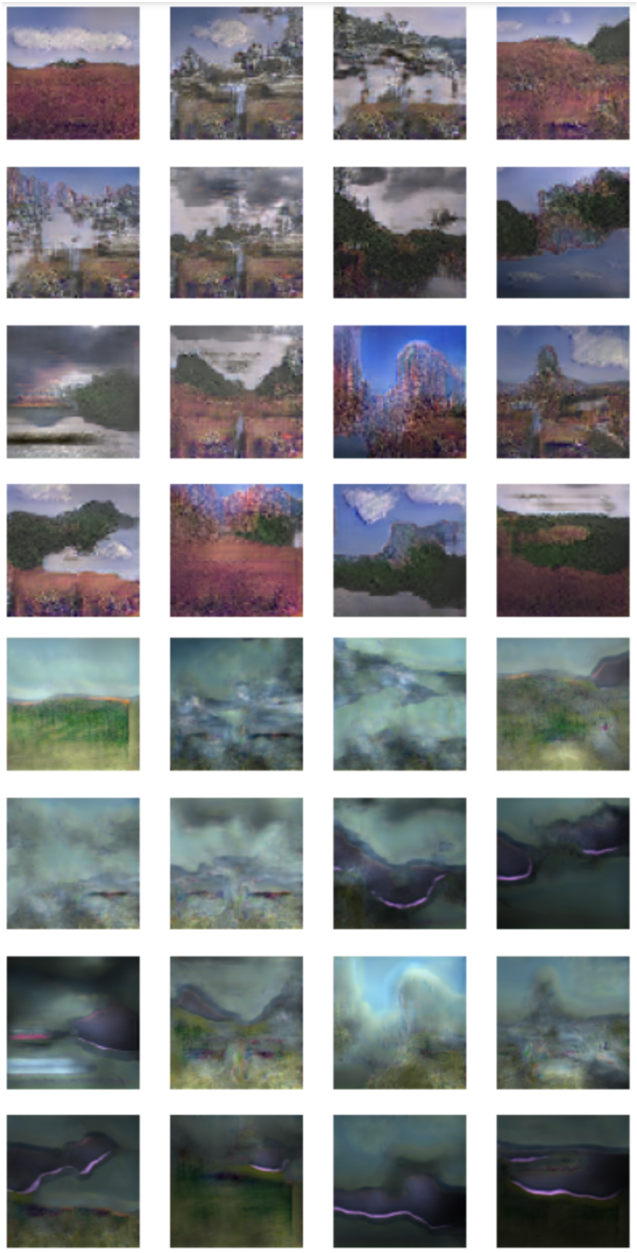

Figure 13: Stage-I and stage-II images for stage-II epoch 210

Figure 14: Stage-I and stage-II images for stage-II epoch 220

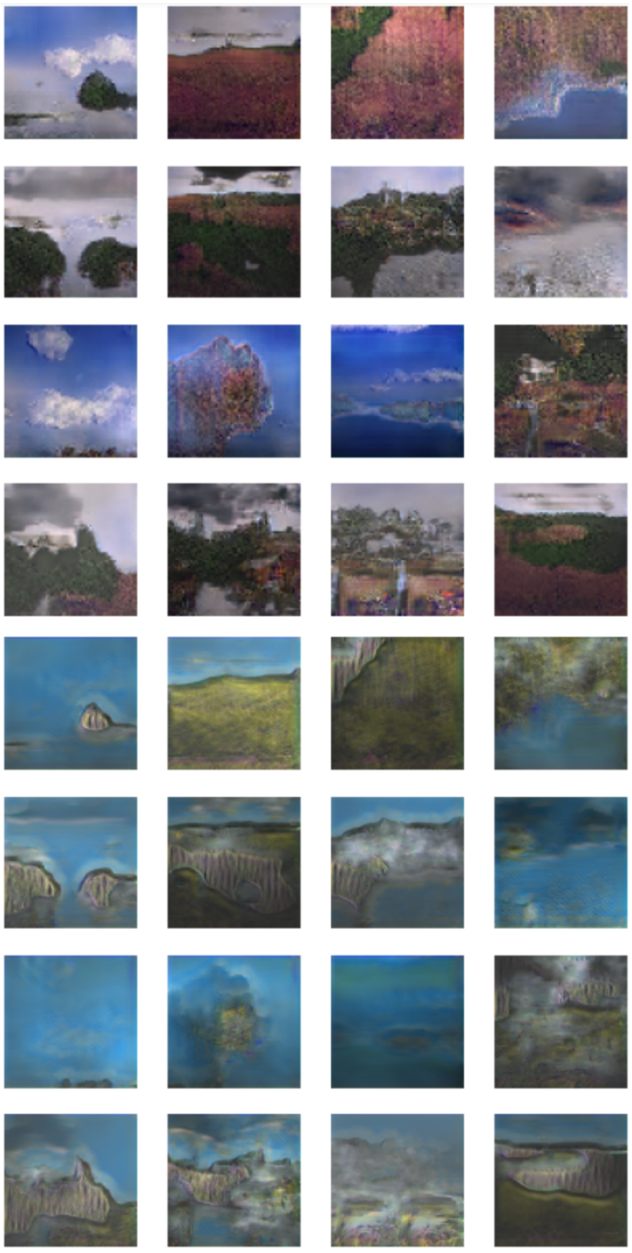

Figure 15: Stage-I and stage-II images for stage-II epoch 230

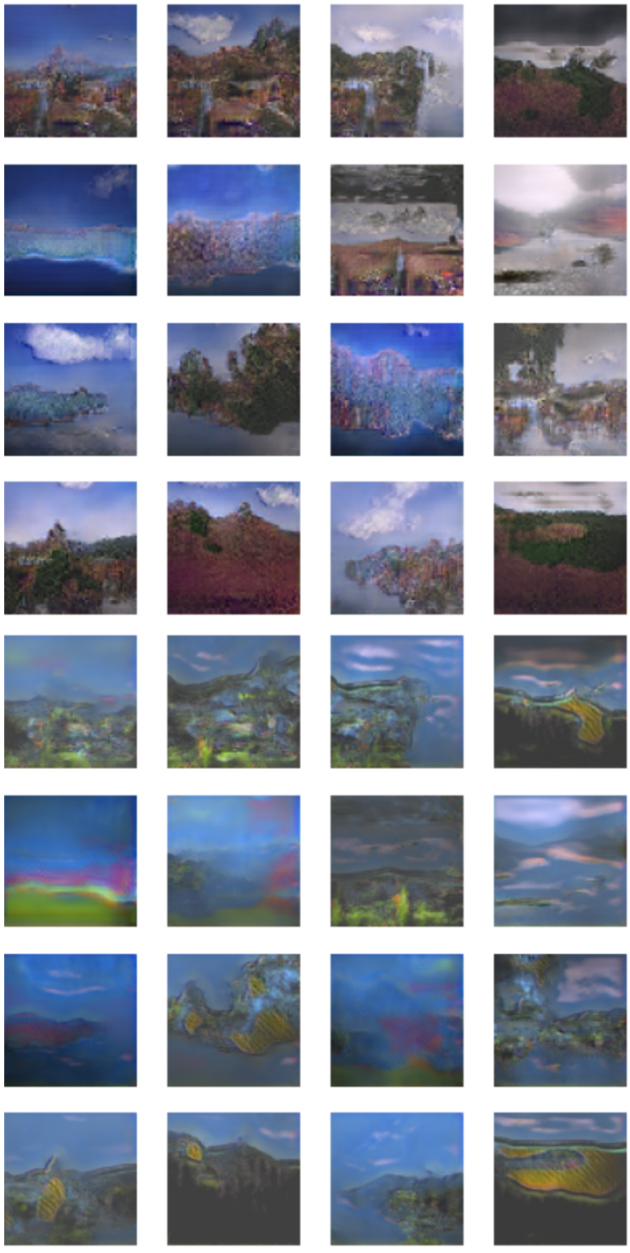

Figure 16: Stage-I and stage-II images for stage-II epoch 240

Figure 17: Stage-I and stage-II images for stage-II epoch 250

Figure 18: Stage-I and stage-II images for stage-II epoch 260

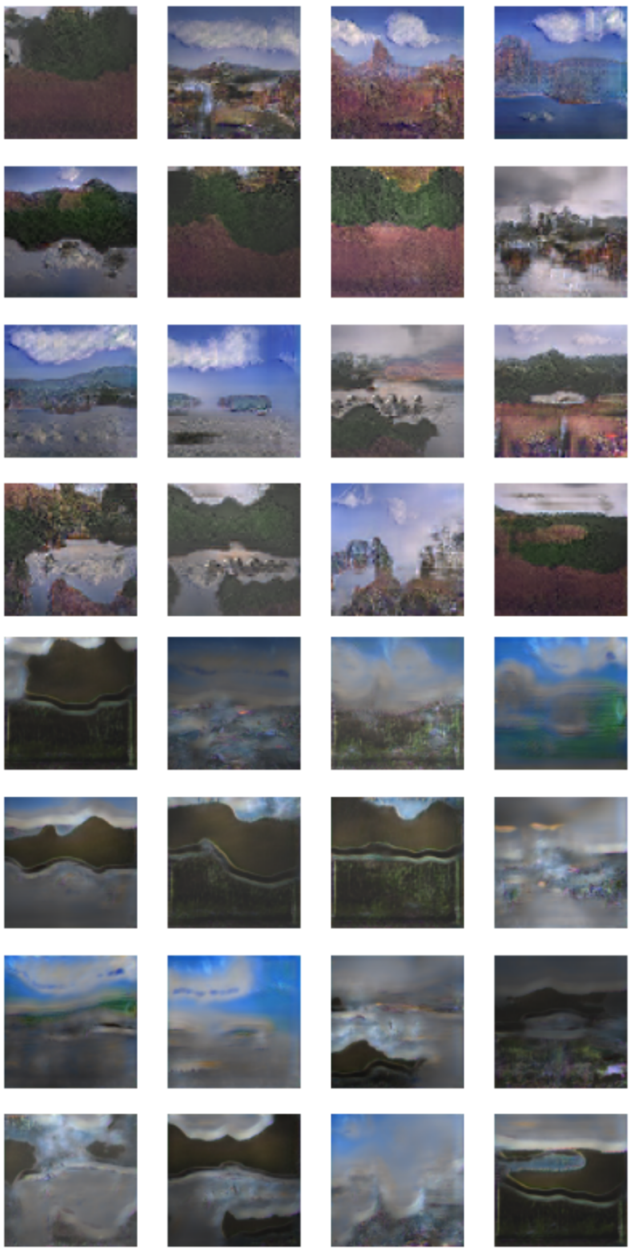

Figure 19: Stage-I and stage-II images for stage-II epoch 270

Figure 20: Stage-I and stage-II images for stage-II epoch 280

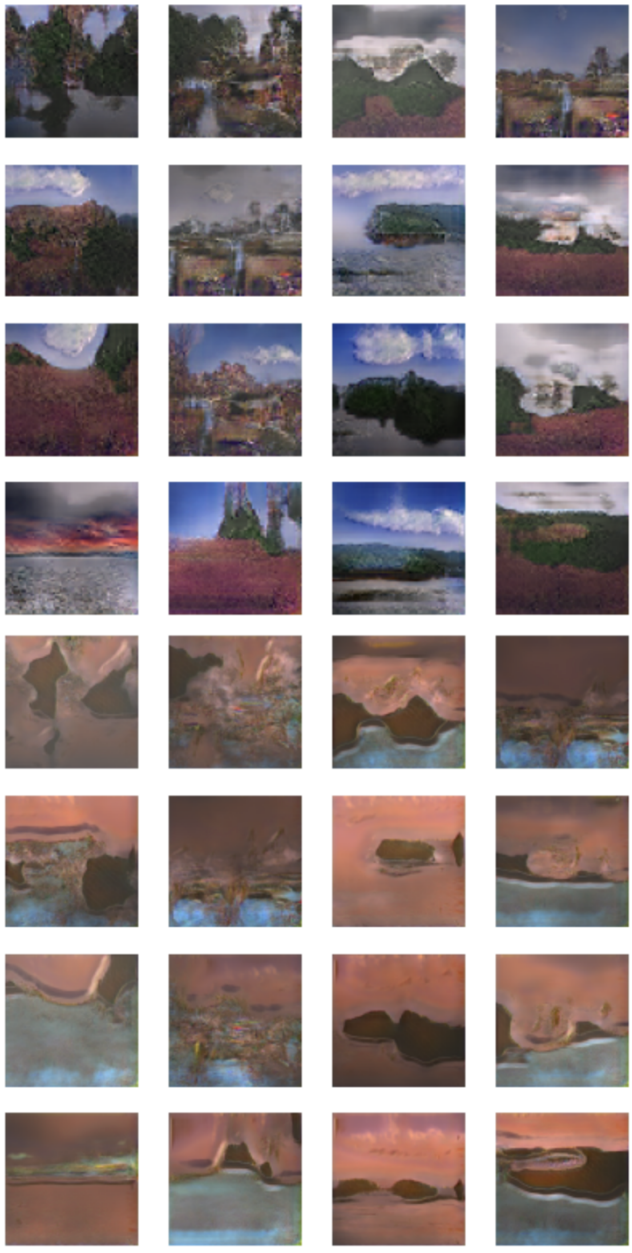

Figure 21: Stage-I and stage-II images for stage-II epoch 290

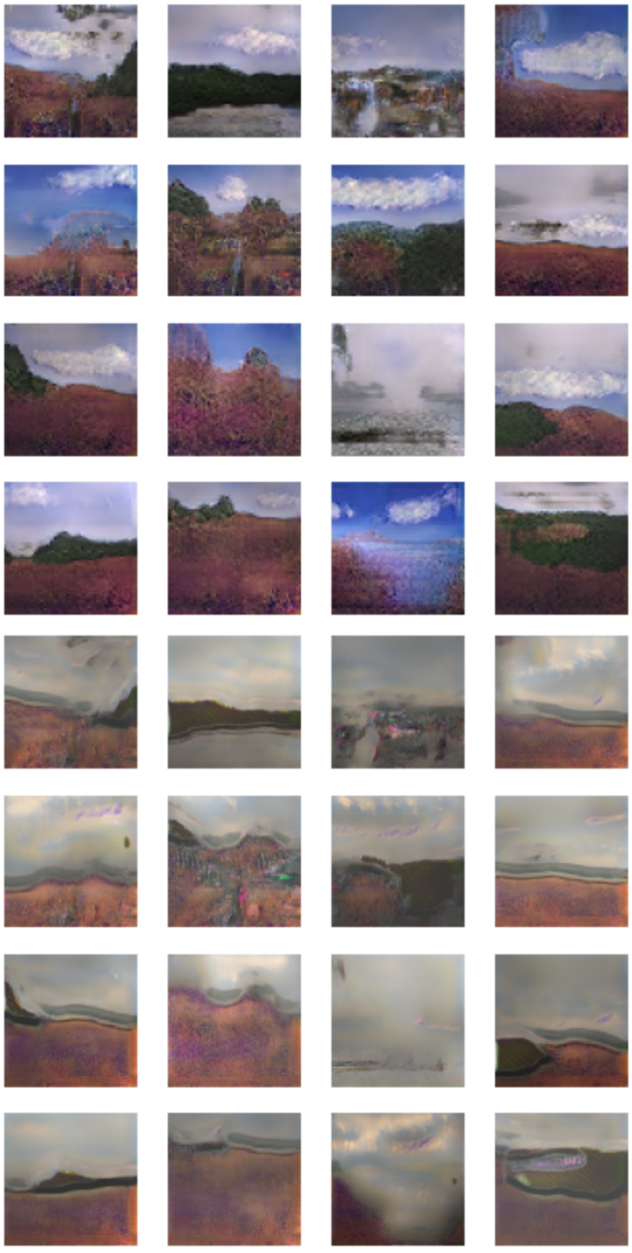

Figure 22: Stage-I and stage-II images for stage-II epoch 300