From Zero to Hero
Training a GAN from Scratch For Image Inpainting

Ella Shalom

1 Introduction

Image inpainting is the task of filling missing pixels in an image in a way that looks realistic. We aspire to not be able to set apart the part we filled from the original picture. It should seem to us as realistic as if it was a real picture. There are numerous applications to image inpainting. It can be used for removal of unwanted objects from pictures. For example, text, subtitles, stamps and publicity that we may want to remove. The filling of missing parts of an image is a technique that exist in literature and can be divided into two main categories: image inpainting and copy-paste methods. The core idea of copy-paste is to first search for the most similar image patches from the remaining pixels of the image itself or a large dataset with millions of images, then directly paste the patches on the missing parts. The main advantage of this method is its simplicity, but it is not suitable for some cases like face images or complicated scene. In addition to that, the search algorithm could be time-consuming, and it involves hand-crafted distance measure metrics. The other category is deep learning-based image inpainting. The success of this approach derives from the era of Big Data. We are trying to learn the distribution of the input and to generate the missing pixels in an image with good global consistency and local fine textures. Therefore, image inpainting may be more suitable in object removal problems that need to be efficient and better generalized. Image inpainting is also crucial for restoration of photographs, films and paintings. In the past, it was done manually. Today, because of the recent development in image processing techniques it has gained even more popularity. By using recently developed algorithms, image inpainting can restore coherently both texture and structure components of the image.

2 Related Work

This is a review of a couple of the main articles I've red and used their ideas in my solution. I described each article shortly and focused mainly on the

ideas I used. The article this report is based on, is Context Encoders[10]. It is the first Generative Adversarial Networks based inpainting algorithm. This paper used some basic terms in image inpainting. For example, the term "Context" relates to the understanding of the entire image itself. One of the main ideas in this paper is the use of Channel-wise Fully Connected Layer between the encoder and the decoder. It is done to allow the network to get a better semantic feature understanding. The network can learn the relationship between all the feature locations, all the feature locations at the previous layer (encoder) would contribute to each feature location at the current layer (decoder). Patch-based Image Inpainting with GANs [5] introduced two advanced concepts namely residual learning and PatchGAN. The authors of this paper combined residual connection and dilated convolution to form a dilated residual block. The traditional GAN discriminator was also replaced by the PatchGAN discriminator to encourage better local texture details and global structure consistency. PatchGAN output is a matrix of predicted labels which indicate the realness of each local region of the input. The Pix2Pix GAN [6] is a general approach for image-to-image translation. It is based on the conditional generative adversarial network, where a target image is generated, conditioned on a given input image. In their article, the Pix2Pix GAN changes the loss function so that the generated image is both plausible in the content of the target domain and is a plausible translation of the input image. They also use PatchGAN discriminator and U-Net architecture. The U-Net model architecture is very similar to the regular encoder-decoder autoencoder architecture, it involves down sampling to a bottleneck and up sampling again to an output image. But it adds skip-connections between layers of the same size in the encoder and the decoder, allowing the bottleneck to be skipped. Deep Convolutional GAN [3] article uses a couple of guidelines. They replaced any pooling layers with stride convolutions in the discriminator and fractional-strided convolutions in the generator. Used ReLU activation in generator (except for the output, which uses tanh), and LeakyReLU activation in the discriminator. They used batchnorm in both the generator and the discriminator.

3 Solution

3.1 General approach

I relied on the articles described above in planning my solution to the problem. I maintained the overall architecture as described in the Context Encoders[10] paper. My model is composed of encoder-decoder generator architecture. Its purpose is taking an image as input and down sampling it over a few layers until a bottleneck layer, where the representation is then up sampled again over a few layers before outputting the final image with the desired size. The encoder is capturing the context of an image into a compact latent feature representation, and of Decoder architecture which uses that representation to produce the missing image content. Between the encoder and the decoder, there is a channel-

wise fully connected layer, as described in the paper. I took inspiration from the pix2pix [5] article described above and added a U-Net architecture between the encoder and decoder. It was built using simple blocks, that was combined together in a recursive manner. It will be further explained in the next section. I implemented a global discriminator and PatchGAN discriminator. The global discriminator looks at the whole image, while a local discriminator (PatchGAN discriminator) looks at different regions of the filled image, which may give the possibility to concentrate more on the masked regions. I wanted to learn as much as I can from this project, and to experiment with my model while implementing various ideas and techniques. I decided to try and train a GAN from scratch. It allowed me to understand how every decision I make influence the results, some for better and some for worse.

3.2 Design

3.2.1 Data

There was Small number of training examples in the data that was given. I decided to expand the dataset by adding datasets from similar domains. Because the data is not tagged, I could easily find similar datasets. For the first dataset, with size 7038, I found a landscape dataset [2]. For the second dataset, with size 300, I found a paintings dataset and chose manually the painters with similar style [1]. Overall, I trained on a dataset of 11357 pictures in the first model. After that, I took this model and tuned it to fit the Monet task. I trained on a dataset of 2343 pictures in the Monet task. Tunning the first model and not training one from scratch, also helped with the small number of input images at the Monet task. The input for the model is a masked image, and as it will be explained later, the mask creation process is random. This means that even when the model gets the same picture in a different epoch, it will probably not get the same input. Which is a kind of data augmentation, that amplifies the input size for the model.

3.2.2 Masks

As described on the Context Encoders paper, I created three types of masks. central square patch, referred as center region. Random number of smaller possibly overlapping masks which covers up to quarter of the image, referred as random region. And random region, a random mask obtained from the PASCAL VOC 2012 [8] dataset and paste in a random location. The type of the masks is chosen randomly, with a higher probability for the last two. As in the article it says using these types will lead to better results. After running my model a few times, I saw that the generator may need reinforcement. First, I tried concatenating the mask to the input as a fourth channel. I was hoping it will allow the generator to distinguish between the parts it should restore and the parts it should generate. After that, I tried to initialize the masked areas with noise, and not set it to zero. We learned in class it is done in GANs, and I hoped

it will help the generator learn also in my model. Those two improvements led to better results.

3.2.3 Prepossessing

I resized the input pictures to 256 x 256 as I knew I had limited time and resources for training my model, and zero centered it

3.2.4 Generator

First, I implemented the encoder decoder architecture as described in [10]. I saw that the generated image does not contain details as the original one. Even after training it for many hours, the output was still too blurry. U-Net architecture was implemented as suggested in Pix2Pix [5]. The U-Net architecture is a fully convolutional network, without any other layers such as max-pooling or flatten. The U-Net architecture is composed of blocks. Each U-Net block contains (encoder block)-(recursive U-Net block)-(decoder block). In the encoder, we downsample the Image, using convolution operation that reduces both the Height and Width of the Image by half and doubles the channel number. In the decoder, we apply an upsampling operation, ConvTranspose2d which doubles the height and width of the image and reduces the number of channels by half. Also, in each decoding block, the outputs from the corresponding encoding block are concatenated to the input from the previous decoding block. Each encoder block has a convolution operation, followed by LeakyReluand and BatchNorm at the end. The convolution operations are 4x4 convolutions, with 2 stride and 1 padding. Another attempt to preserve the image details was using PatchGAN. I wanted that the output of each matrix cell will have a receptive field of 40×40 square from the input image. the output of the global discriminator is a single value ranges from 0 to 1, 1 stands for real and 0 for fake. Its receptive field is the entire image. Because of this receptive field, local texture and details of the image may not be expressed. On the other hand, the output of PatchGAN discriminator is a matrix and each element in this matrix ranges from 0 to 1. Each element represents a local region in the input image as described above. The discriminator decides for each patch whether it is real or not. By doing this, the local texture details of the generated images can be enhanced. I experimented with different patch sizes. When I tried larger ones, I saw that it damages the generator ability to fill the missing part with texture and details. When I tried smaller ones, I saw that it has too much impact on the loss, and therefor hurt the effect of the global discriminator. The small receptive field is causing the PatchGAN output matrix to be too big. After many experiments, I decided on 40×40 receptive field. I wrote receptive field calculator, which computes the depth needed in the model that will output the receptive field I want. This calculator helped me easily observe the effect of different size of receptive field on my model.

3.2.5 Preserving Style

After running the model for a few more hours, I saw that the generator succeeds in understanding the overall colors of the missing part in the input, but it has difficulty with details. I thought that style loss may help generate more detailed output. I tried to search for research references of that idea and found that the idea of preserving the texture is common. Texture loss is related to perceptual loss and style loss[4]. I read in [6] that PatchGAN can be understood as a form of style loss, and my decision to implement it was also part of my effort to preserve the image style. Even though it was important to me training from scratch, I also wanted to experiment with transfer learning. I took the trained classification network of VGG-13 with batch normalization that was pre-trained on ImageNet. I assumed this network will have good feature extraction that will help me to preserve the style of the generated image. The generated images were given as an input to this network. And each layer activation was saved in order to compute the style loss. For each layer, a Gram matrix was computed, which represent the style. for the input image and the generated image. Then the style loss is defined as the root mean square difference between the two style matrices. The Gram matrix essentially captures the "distribution of features" of a set of feature maps in each layer. By trying to minimize the style loss between two images, you are essentially matching the distribution of features between the two images, as can be seen in [7].

3.2.6 Discriminator

The architecture of the discriminator is similar to the one of the encoder from the generator. I decided to give the generator more computational power than the discriminator, meaning smaller number of learning parameters. This was done after the discriminator was much better than the generator, and managed to correctly classify the image almost every time.

3.2.7 Loss Functions

At first, I implemented Gan loss, and added L1 loss as described in [10]. After running the model with this loss function, the results were not satisfying. I saw that the discriminator loss was substantially smaller than the generator loss. I thought that maybe this it is due to saturating gradients. The generator loss takes into account the gradient of the discriminator. If the gradient of the discriminator is saturating to zero, the generator won't be able to learn good feature representation. I thought it will help to punish the model according to its proximity to the decision boundary. Even if the discriminator can easily distinguish the generated inputs from the original ones, I hoped that considering how far the generated input was from a real looking one, will help guide the generator in the right direction. This is what brought me to implement Least Squares GAN [9]. Style loss was computed and added to the generator loss as described above. After implementing style loss, I read about total variation loss at [3]. I realized that optimization to reduce only the style and content losses

may cause the output to be pixelated and noisy. I read that total variation loss could help ensuring spatial continuity and smoothness in the generated image. After implementing total variation loss and style loss, at first it seemed like adding those losses damaged the quality of my results. It took some playing with the losses weights to see the improvement. PatchGAN was implemented as described, so I also implemented Patch loss for it. The output of the global discriminator is a single value, while the output of the Patch discriminator is matrix. Therefore, the values are concatenated and then adversarial loss is computed . In conclusion, I wanted to experiment with different loss functions, each one helps to get better result in a different manner. And to find the best loss weights for the problem.

3.2.8 Performance evaluation

During the time I trained the model It was often difficult to understand why things were not working properly. I looked on the pictures manually and tried to help the generator get good results. I looked on the generator loss and the discriminator loss on the real input vs on the generated input. I trained the model for the photo's task about 22 hours, 500 epochs. And the model for the Monet task for 7 hours about 90 epochs.

3.2.9 Optimizers and activation functions

As I saw in different articles, I chose to use Adam optimizer. I played with different schedulers in my solution, and at the end decided on scheduler that reduce the learning rate on Plato. This scheduler gets good results because it allows me to keep high learning rate as long as the model converges, and to lower its value when it starts to get stack.

4 Results

I present four image result for each picture: the original picture (up left), the original picture with mask (up right), the generated image (bottom left), the original image with the generated masked parts (bottom right). The third one presents the ability of the model to autoencode the image, even when passed through a narrow spatial bottleneck. The last one allows us to see how the missing part fits in the generated image. The inpainting results are mostly realistic. The model succeeds to capture the concept of the image as well as the details of it. The model struggles more with the first kind of mask, or with other large continuous masks. Each time I looked on those inputs during training I saw that the model is getting better and better. it was able to complete growing regions in the edges of the mask. I believe that it could improved further with more training time. The results of the landscape test set are presented in Figure 1. The results of the Monet test set are presented in Figure 2. I believe that the Monet inpainting could be significantly improved given additional training

time. Increasing the data set size, by using more artists could also be helpful. Additional results can be seen in Figure 3.

5 Discussion

It was a learning process, in which I kept adding improvements according to problems I detected as I described on the different sections above. I knew training a GAN from scratch may be difficult, but it helped me better understand all the related concepts. Training a GAN from scratch is a difficult process, which requires a lot of attention to details, and time to play with the different parameters. I had a lot of difficulty in creating a generator which is able to generate specific details, and I tried to implement many ideas to solve that problem. During my work, I saw massive improvement of it, and still, the generator struggles with some inputs. In order to solve this project problems, I read articles and mainly tried to use the intuition I have to help find solutions to problems in the learning process.

References

[1] Best artworks of all time, collection of paintings of the 50 most influential artists of all time. *https://www.kaggle.com/ikarus777/best-artworks-of-alltime?select=images*.

[2] Landscape pictures, datasets of pictures of natural landscapes. *https://www.kaggle.com/arnaud58/landscape-pictures*.

[3] Soumith Chintala Alec Radford, Luke Metz. Unsupervised representation learning with deep convolutional generative adversarial networks.

[4] Zhe Lin Eli Shechtman Oliver Wang Hao Li Chao Yang, Xin Lu. High-resolution image inpainting using multi-scale neural patch synthesis.

[5] Ugur Demir and Gozde Unal. Patch-based image inpainting with generative adversarial networks.

[6] Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *Berkeley AI Research (BAIR) Laboratory, UC Berkeley.*

[7] Matthias Bethge Leon A. Gatys, Alexander S. Ecker. Image style transfer using convolutional neural networks.

[8] L. Van Gool C. K. Williams J. Winn M. Everingham, S. A. Eslami and A. Zissermans. The pascal visual object classes challenge: A retrospective. *IJCV*, 2014.

[9] Haoran Xie Raymond Y.K. Lau Zhen Wang Stephen Paul Smolley Mao, Qing Li. Least squares generative adversarial networks.

[10] Deepak Pathak Philipp Krahenb ¨ uhl Jeff Donahue Trevor Darrell Alexei A. Efros. Context encoders: Feature learning by inpainting. *Asimov, I. Runaround*, 2016.

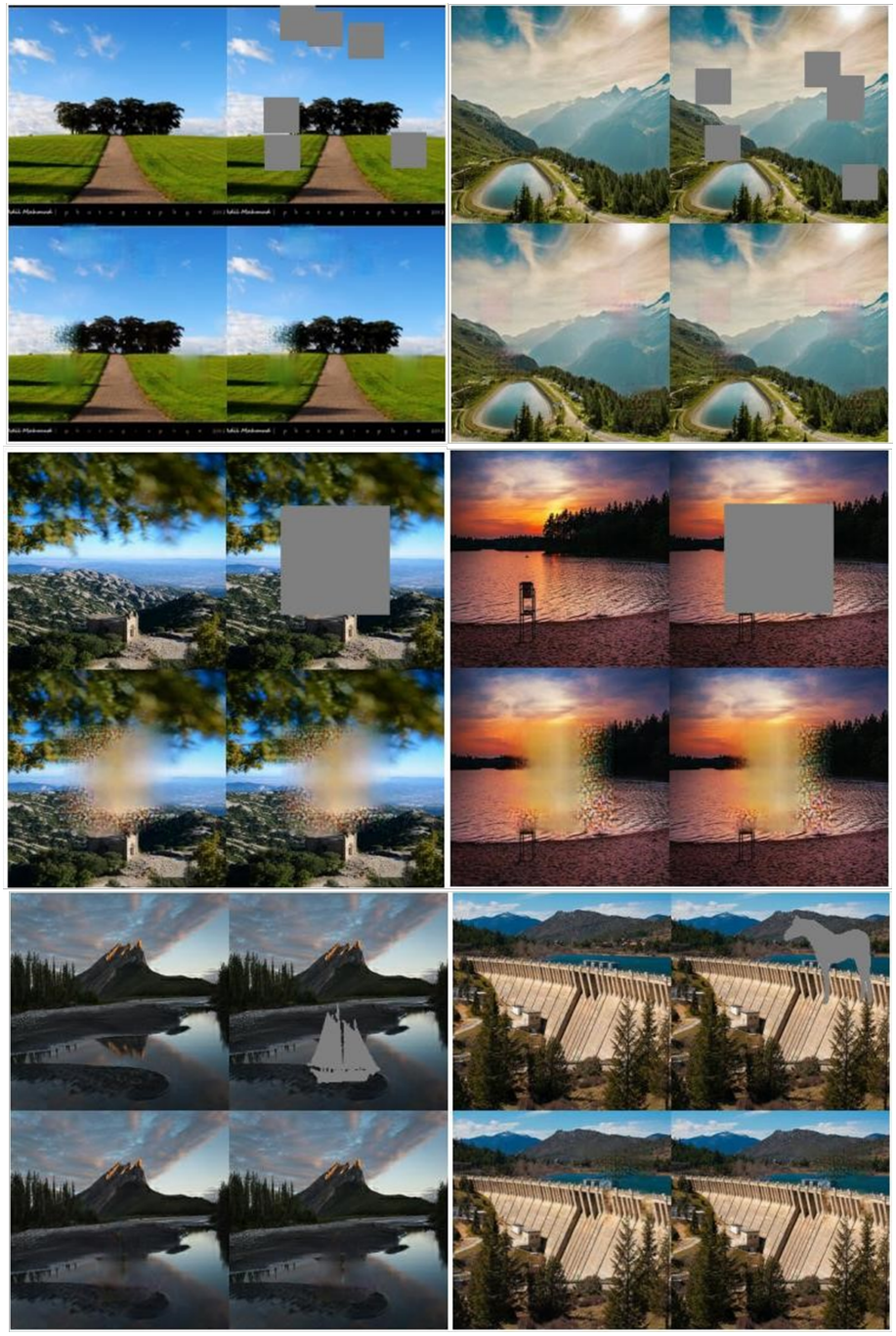

Figure 1: Results of the landscape set

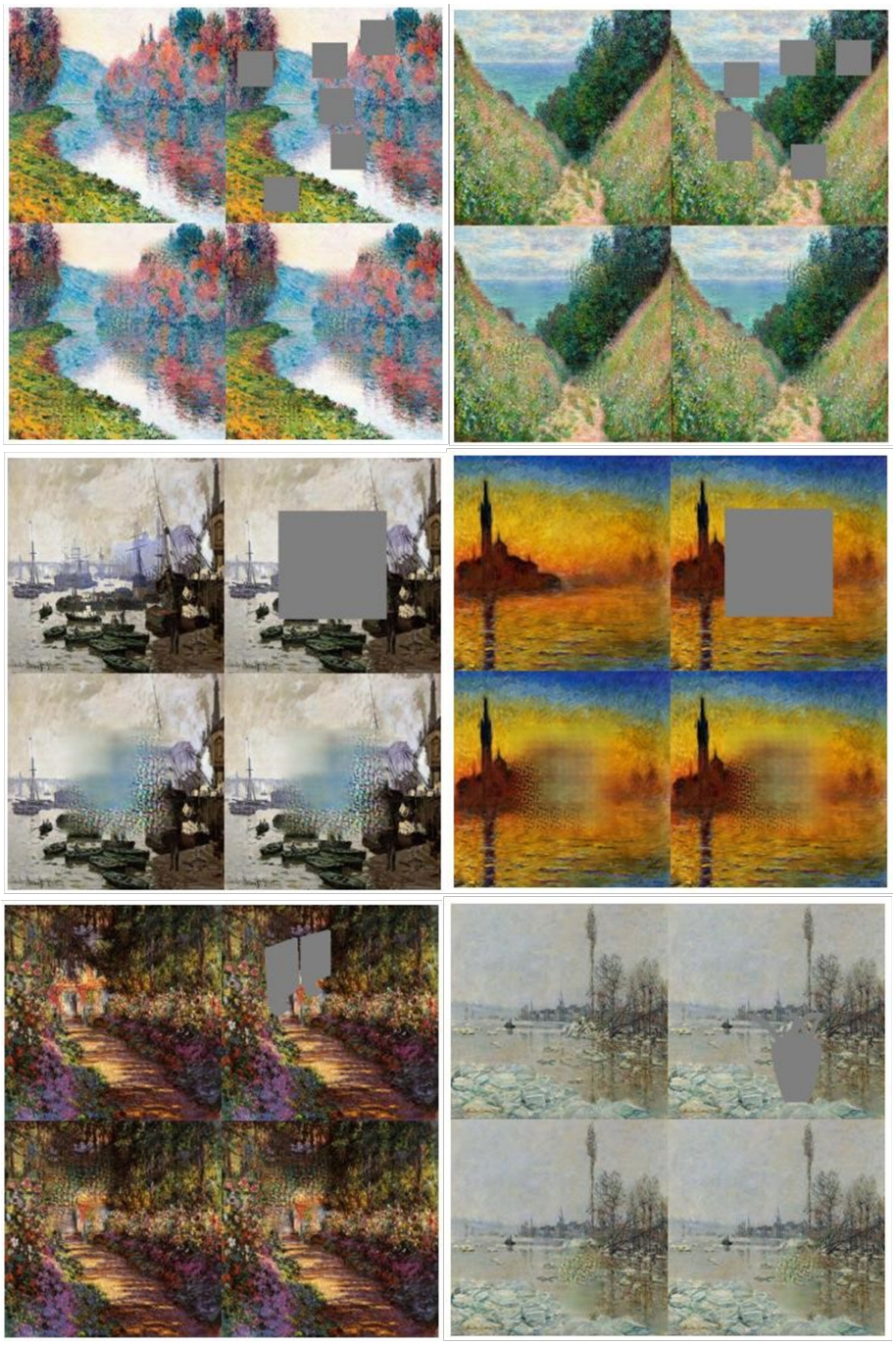

Figure 2: Results of the Monet set

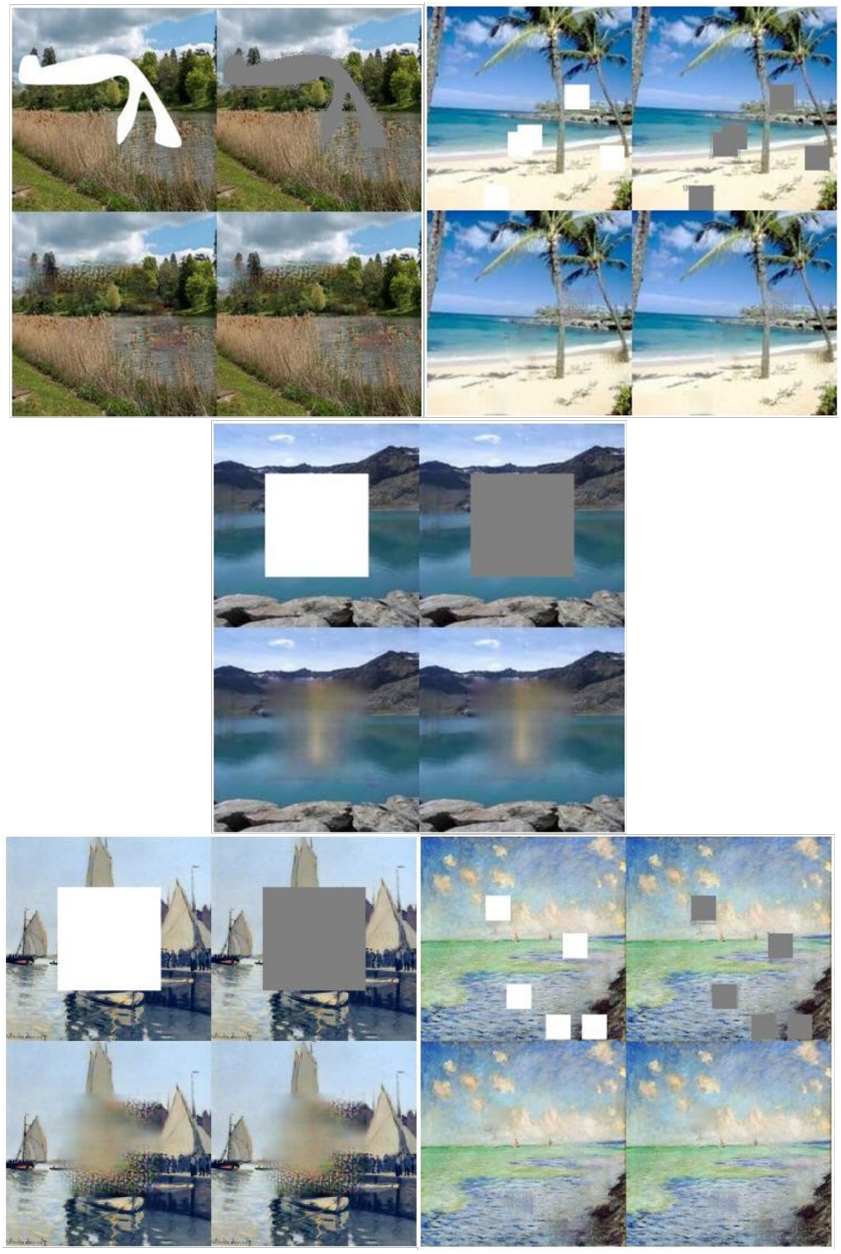

Figure 3: Additional Results