67103 Assignment 3
Omer Siton - 316123819

**Loss Saturation.** Train the generator together with the discriminator following the classic GAN paradigm taught in class. For this purpose explore the case where multiple training optimization steps are applied to the discriminator per a single iteration for the generator. Try three different GAN losses:

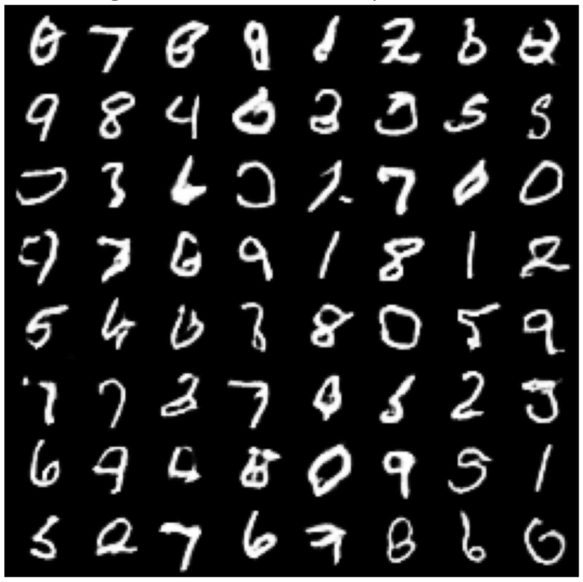
(i) The original GAN cross-entropy
(ii) The non-saturating version of this loss (as mentioned in class)
(iii) Least-squares loss (L2)

See which losses lead to saturation and explain why this is the case.

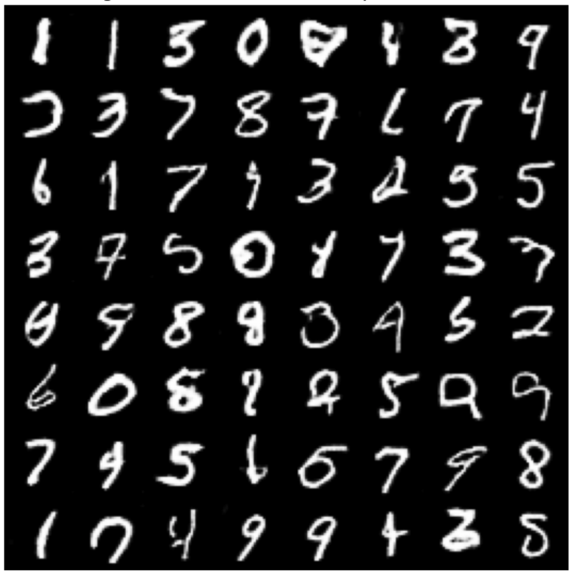
Based on the tests I performed and the material learned in class, it seems that increasing the number of discriminator optimization steps (D-steps) per generator iteration can lead to the collapse/saturation of the training process. This suggests that a value of 1 for D-steps performs the best, while higher values may have a negative impact. It's because the task of the discriminator at the beginning is much simpler than the generator so it could lead to vanishing gradients in the generator and for the training process to reach saturation.
Vanishing gradients occur when the gradients become extremely small during backpropagation, making it difficult for the model to learn effectively. This can result in slow convergence or no convergence at all. In the context of GANs, vanishing gradients in the original saturated loss can hinder the training progress and prevent the generator from improving over time.
The non-saturating loss and the least-squares loss (L2 loss), do not suffer from vanishing gradients. These alternative loss functions have been designed to address the limitations of the original GAN saturated loss. By using different loss formulations, and especially maximizing Log(D(G(Z)) rather than minimizing Log(1-D(G(z)). These losses provide more stable training dynamics and avoid the vanishing gradient problem. As a result, they enable the generator to learn more effectively and generate higher-quality samples.

i. The original loss led to saturation very quickly in the training process. The only case that avoided saturation was when D-steps was 1.
ii. The non-saturating loss led to the best results and was able to construct data that looks very similar to the original data while not getting saturated in the training process.
iii. The L2 loss led to good results. The benefit of the least squares loss is that it gives more penalty to larger errors, in turn resulting in a large correction rather than a vanishing gradient and no model update.

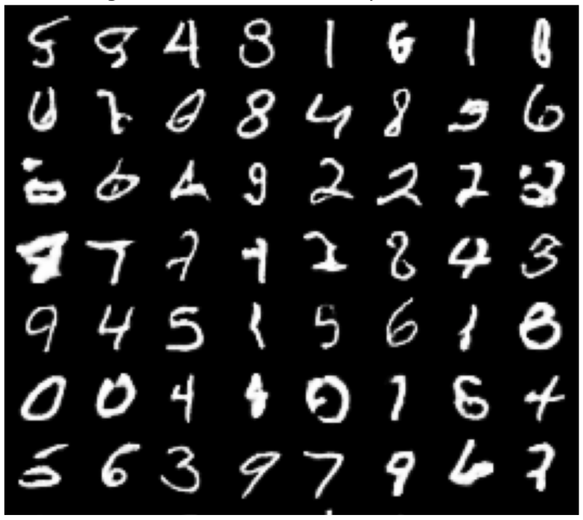
Fake Images criterion:BCELoss(), Dsteps:1, saturated:True



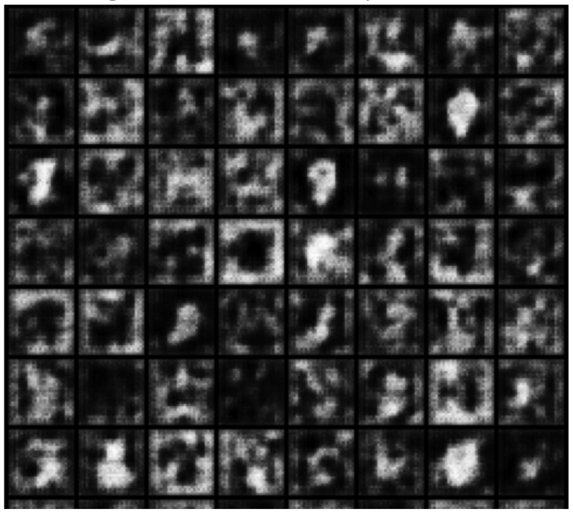
Fake Images criterion:MSELoss(), Dsteps:1, saturated:False



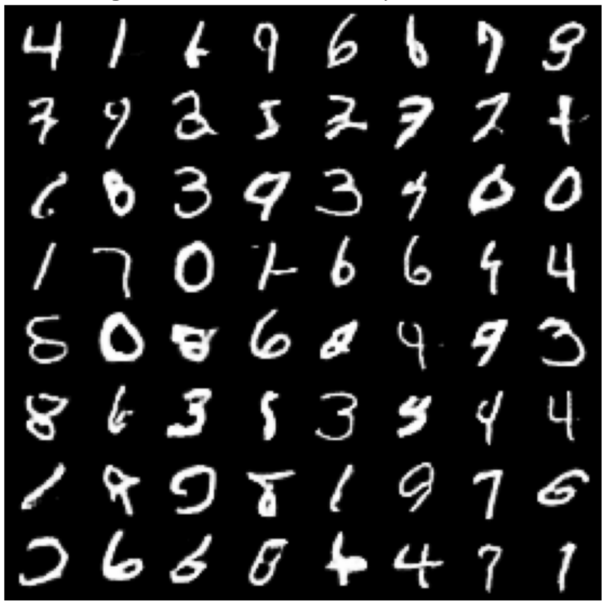
Fake Images criterion:BCELoss(), Dsteps:1, saturated:False



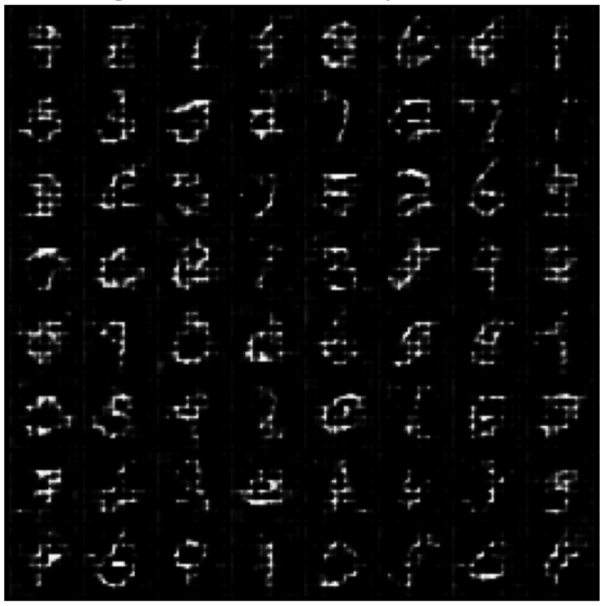
Fake Images criterion:BCELoss(), Dsteps:2, saturated:True



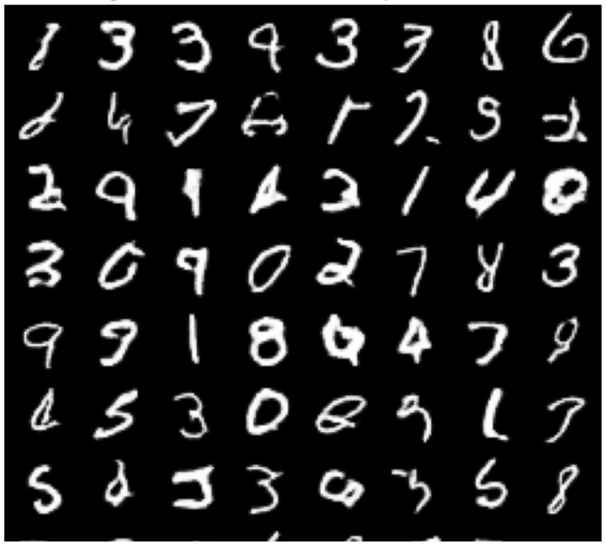
Fake Images criterion:BCELoss(), Dsteps:2, saturated:False



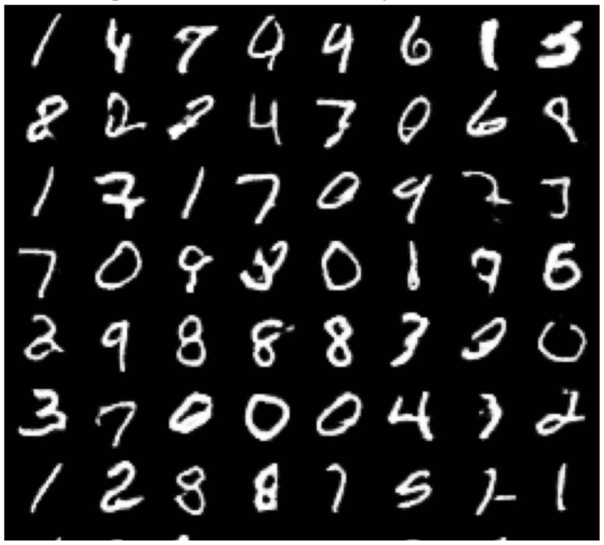
Fake Images criterion:BCELoss(), Dsteps:3, saturated:True



Fake Images criterion:MSELoss(), Dsteps:2, saturated:False



Fake Images criterion:BCELoss(), Dsteps:3, saturated:False



Fake Images criterion:MSELoss(), Dsteps:3, saturated:False

**Model Inversion.** One elementary operation that can be performed as a basis for various applications (such as image editing) is GAN inversion; namely, given an image $I$, find its source latent vector $z$ that reproduces that image, i.e., maps $G(z) = I$ where $G$ is the generator. This should be implemented on a trained $G$, and optimized over the latent space of $z$ vectors. Explain which features in the image are accurately reconstructed, and which aren't. Please provide an explanation of why you think this is the case.

I choosed to use the non-saturating loss with 2 iterations for the discriminator per 1 iteration for the generator which performed really good.
Overall, the results for this experiment are pretty impressive. As you can see in the image bellow, most of the digits are very similar to the real input.
The high-level features and overall structure of the image are typically accurately reconstructed. These features include global shapes, contours, and the arrangement of major objects in the image. The generator network of the GAN is trained to capture and generate realistic images, so it has learned to represent these fundamental features in the latent space. Therefore, during inversion, the model can effectively map the given image back to its corresponding latent vector, leading to accurate reconstruction of these high-level features.
While the global features of the image captured well, the fine-grained details and local variations weren't a success. These features encompass small textures, intricate patterns, or pixel-level details that may not be fully represented in the latent space. The generator's capacity to model such fine details might be limited by the complexity of the network architecture, we know that deeper CNN lead to more complex features captured by the model, or the resolution of the training data. Consequently, during inversion, the reconstructed image lacked some of these subtle and specific details, resulting in a slightly smoothed or less-detailed appearance.

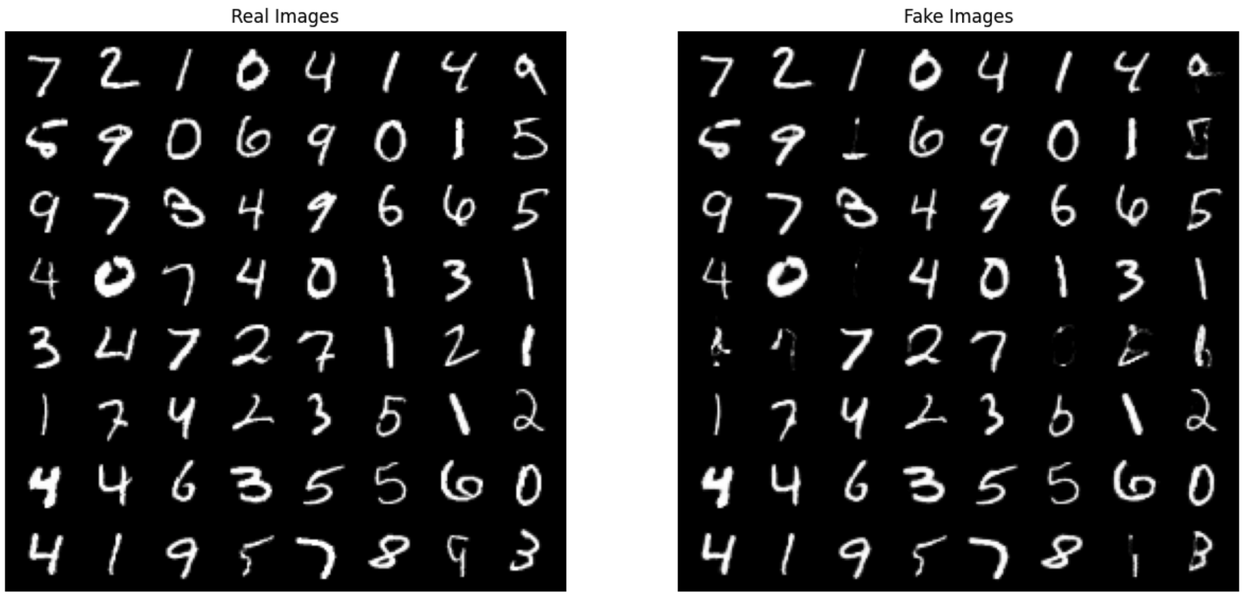
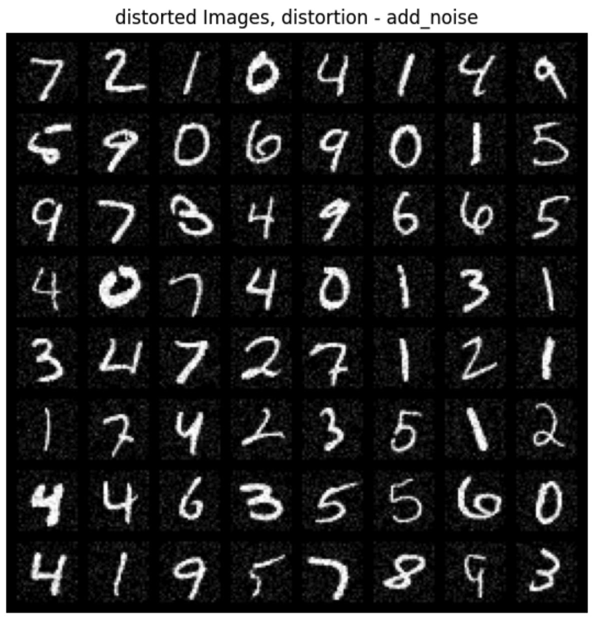


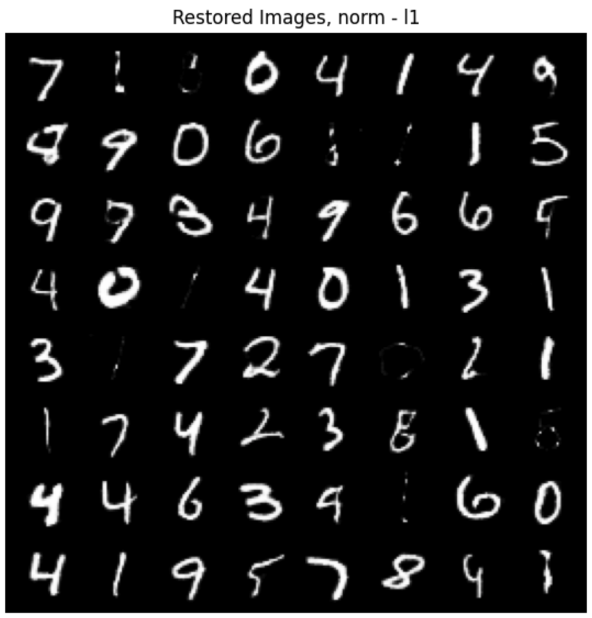
**Image restoration**. The generative abilities of a GAN, coupled with the ability to encode images in the GAN latent space allows restoring corrupted images. This can be done by searching for the closest latent vector in the GAN's latent space for the corrupted image, and using it to reproduce the restored image. Formally, we's like to solve *min_z* ||*G(z) - I* || where *I* is the corrupted image. This search should be invariant, as much as possible, to the corruption itself and hence the norm || . || should be carefully considered with respect to the corruption at hand. Consider this approach for two tasks:

1. restoration of noisy images (denoising), use i.i.d. Normal noise with m=0, and std around 0.1 (for pixels values within [0,1])
2. Filling missing pixels in images with holes (inpainting), where you delete random windows of around 8x8 pixels.

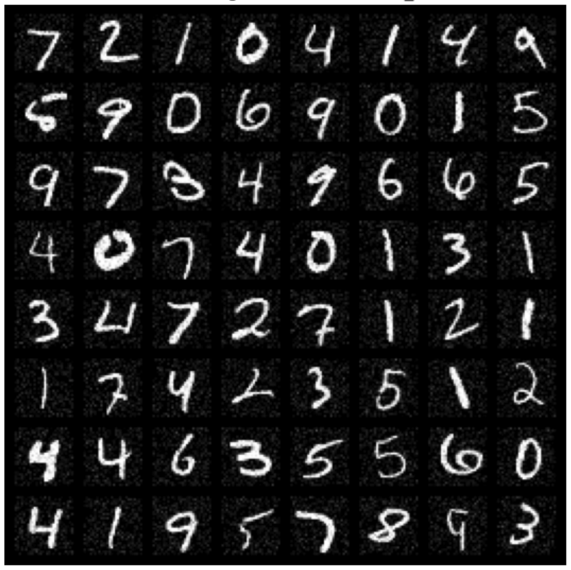
In your report, explain the norm you used and show the results it obtained.

So I checked two different norms in this experiment, L1, and L2, on each distortion (1,2). Overall, the results are good for filling missing part and great for denoising, the model was able to restore the images in a decent way, in some cases it was thrown really far from the original distorted image but it's probably because of optimization process like any SGD (also I used a quite high learning rate for this task).
In other cases it was given not perfect restoration but in cases that make sense to be more complex for the model like edges around the patches.
The results for denoising:

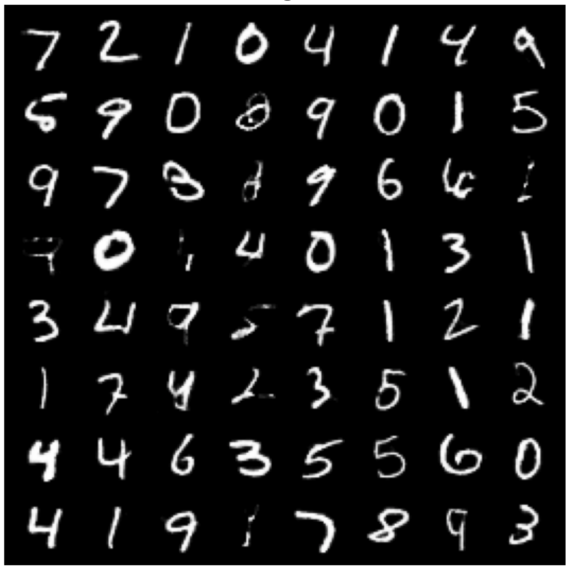




distorted Images, distortion - add_noise



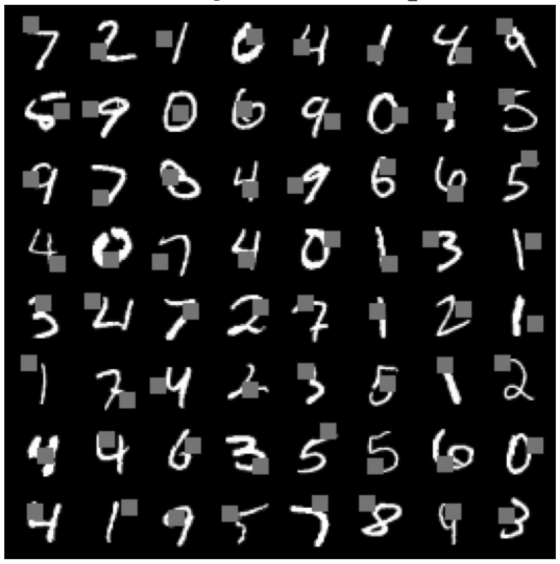
Restored Images, norm - l2



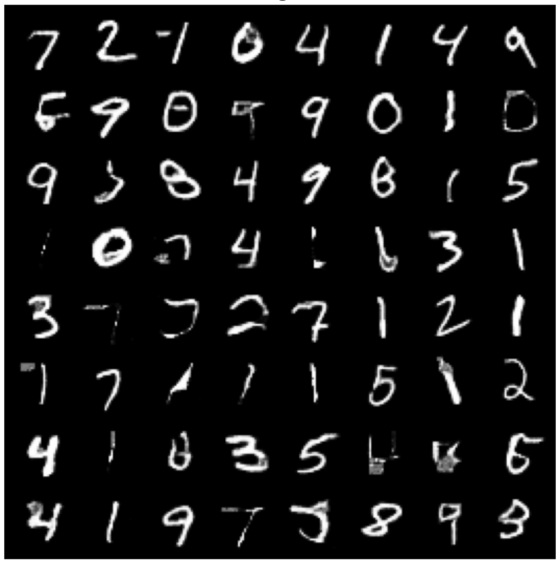
We can see from the results that the l2 norm was better for this task.
It encourages reconstructions that minimize the overall pixel-wise differences between the two images, so it fits for smoothing tasks such as this task.

The results for the inpainting task:

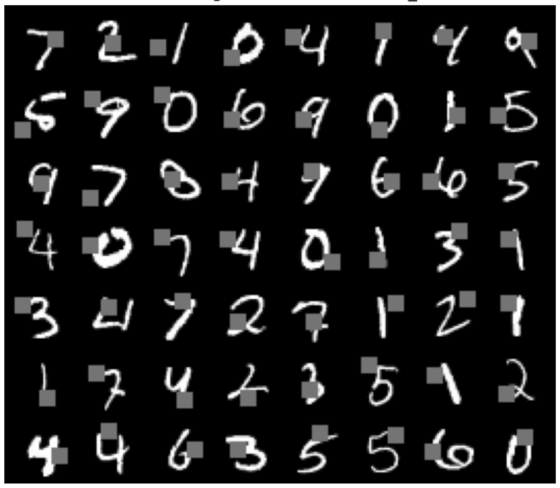
distorted Images, distortion - create_holes



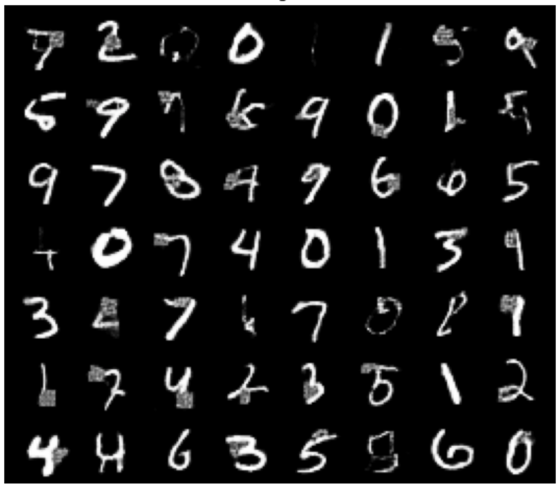
Restored Images, norm - l1



distorted Images, distortion - create_holes



Restored Images, norm - l2



Here l1 norm was much better in restoring the images. This norm promote sparsity and can capture sharp edges.
* I used grey patches because most of the image is black and white

1. Explain how the value of P(x) can be computed in the GLOW settings. Write down the explicit formula given the components computed/available in this method. Explain why P(x) cannot be computed in GAN and GLO.

The explicit formula for computing P(x) in GLOW can be derived as follows:

1. Assume z is a random variable following a simple distribution, such as a standard normal distribution.
2. Apply the invertible flow operations in GLOW to transform z into a sample x in the data space: x = f^(-1)(z), where f^(-1) is the inverse transformation function.
3. The probability density function of z can be written as P(z). Since z follows a simple distribution (e.g., standard normal distribution), P(z) can be easily computed.
4. To compute P(x), we need to consider the change of variables from z to x. According to the change of variables formula, we have:
   P(x) = P(z) * |det(dx/dz)|,
   where |det(dx/dz)| represents the absolute value of the determinant of the Jacobian matrix of the transformation from z to x.
5. The determinant of the Jacobian matrix can be computed based on the specific flow operations used in GLOW. Each flow operation has a corresponding Jacobian term, and the determinants of these terms can be computed analytically.

Therefore, to compute P(x) in GLOW, we need to compute P(z) based on the simple distribution of z and calculate the determinant of the Jacobian matrix associated with the flow operations used in the model. By combining these components using the change of variables formula, we can obtain the value of P(x).
P(x) cannot be directly computed in GAN and GLO.
In GANs, the generator network learns to approximate the underlying data distribution without explicitly modeling the probability density function. The generator generates synthetic samples that resemble the real data, but it does not provide an explicit formulation of P(x). GANs focus on training the generator to fool the discriminator and do not directly model the density function.
In GLO, the model is trained to map data samples to a latent space. However, GLO also does not provide an explicit formulation of P(x) since it does not model the density function directly. GLO focuses on learning a mapping from data space to latent space, which allows for latent space manipulation and synthesis of new samples. The density function estimation is not the primary objective of GLO.

At the beginning of a GAN training, G and D are faced with problems with very different complexity. G must create authentic images (hard), D must differentiate between true images and the poorly-produced images that G creates at initialization. Hence, D converges much quicker to a decisive classifier with logit values close to 0 and 1. These values are obtained by small and large values right before its final output sigmoid activation (that maps its values to [0,1]). Show analytically, what happens to the gradients of D with respect to G in this scenario.

In the scenario where GAN training is in its initial phase, and the generator G is still producing poor-quality images, while the discriminator D has converged to a decisive classifier, the gradients of D with respect to G can be analyzed as follows:
Let's denote the output of the discriminator D for a generated sample by D(G(z)), and the output for a real sample by D(x). Since D has converged to a decisive classifier, D(x) will be close to 1 (indicating that it recognizes the real samples as authentic), and D(G(z)) will be close to 0 (indicating that it recognizes the generated samples as fake).
The loss function of the GAN discriminator defined as:
$L_D = -[\log(D(x)) + \log(1 - D(G(z)))]$
To update the generator G, we need to compute the gradients of the discriminator's loss with respect to G, i.e., $\nabla L_D/\nabla G$.
Using chain rule and applying the logarithmic derivatives, we can compute the gradients as follows:
$\nabla L_D/\nabla G = \nabla[-\log(D(x)) - \log(1 - D(G(z)))]/\nabla G$
the gradients of D with respect to G will be influenced only by the second term $\nabla[-\log(1 - D(G(z)))]/\nabla G$.
We can see that this term with respect to $\nabla G$ is close to $D(G(z))/ (1 - D(G(z))) * \nabla G$ which is close to 1/0 and therefore the gradients of D at that case are very high. This is way it could learn very quickly at the beginning.

GAN is trained by solving the following problem: min_G max_D V(D,G)  ~  min_G [max_D V(D,G)] . This implies that the optimization of G occurs over a converged / maximized D. In practice, it means that for every gradient descent (GD) step of G, multiple steps must be made over D. We know however, that this may lead to a saturation of the gradients of G as shown above. In order to avoid that, in practice a single GD iteration of D is applied every GD iteration of G. In a sense, one could equally argue that a different problem is solved, namely, **max_D min_G** V(D,G) ~ **max_D [min_G** V(D,G)**]**. Explain what this problem is solving by:

1. Explaining the what the inner problem **min_G** V(D,G) solves
2. Explain what the outer problem needs to be solved given the converged inner problem. Explain whether you think it is easy or hard for a limited capacity neural network.
3. What put-fall do you see in this inverted formulation assuming D fails to achieve the exact solution it needs to.

1. min_G V(D, G) solves the problem of - by given an image we want the fixed discriminator D to think the generated image is real. So we want to minimize log(1-D(G(z)).
   When D(G(z)) close to 1 it means the discriminator think the generated image is real and the term close to minimum.
2. The outer problem take into account that the inner problem minimized so we have a good generator G, than for the maximizing of D we want D to define which of the images are real and which are fake. In that case this task is no longer easy because G is good now and therefore it will have to be a rich expressive D with high number of neurons to classify those images correctly.
3. As we saw in class this situation could lead to mode collapse. D will not classify correctly between fake and real images and so in turn G will not learn how to fool D more smartly and will keep producing the same generated images that works best for it.