A Technical Deep Dive into Drag Your GAN (DragGAN)

Chen Liu
chen.liu.cl2482@yale.edu

Abstract

This is neither a research article nor a review article. This is an **unofficial** technical deep dive into how DragGAN [1] works. **Treat this as a blog post.**[1] When we read the DragGAN paper, we had multiple misunderstandings before we finally converged to an interpretation that seems convincing. Therefore, we believe it is beneficial to share our final interpretation to the community, in case anyone else is also having trouble parsing the methods section of the DragGAN paper.
Note: If you are an expert in this field, you may find this technical deep dive a complete waste of your time. This article is most suited to people with some background on deep learning and generative models, but have some difficulty comprehending the DragGAN methods by directly reading the paper.
Disclaimer: We, authors of this technical deep dive, have not yet consulted the authors of the DragGAN paper for the technical correctness of our explanations. The opinions and insights are solely based on our own understanding of the DragGAN paper and they may be wrong. The DragGAN authors **shall not be held responsible** for any mistake or misinterpretation in this article, but they **shall be properly cited** if you build upon their work in your research.

Contents

[1] We would have written a blog post instead if we were more familiar with that. However, typing up a document on overleaf seems much easier.

Preprint. Under review.

1 The DragGAN Madness

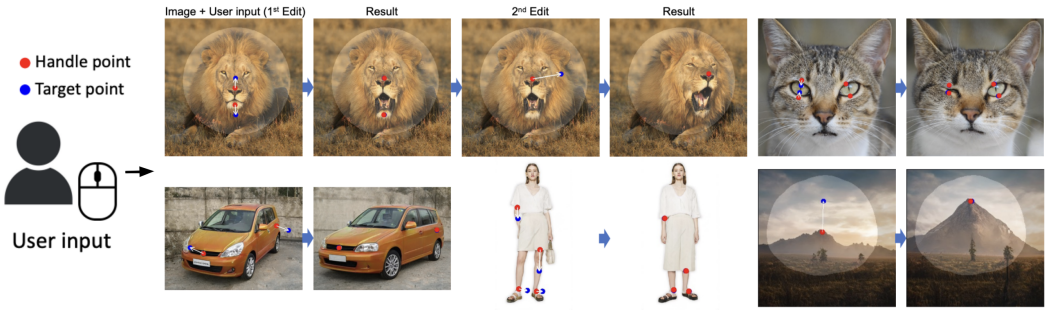

Figure 1: **Demonstration of DragGAN interface.** This is slightly re-made from Fig. 1 and 2 in [1].

DragGAN [1] has become such a big hit recently. The astounding demo from this paper to appear at SIGGRAPH 2023 has been re-posted through various venues including LinkedIn, Twitter, YouTube, and has attracted numerous attention. The GitHub page (`https://github.com/XingangPan/DragGAN`) has accumulated 13.8 thousand stars within as short as 2 weeks, even though it's currently a placeholder with the code to be released in June.

Why are people so excited about this work? The main reason is that it demonstrates a highly controllable and highly user-friendly interface for image generation. By clicking over a few places on an image, the users can manipulate the image seamlessly, and the results look as good as magic.

Specifically, the users can provide one or more pairs of (handle, target) points, and DragGAN will be responsible for "dragging" each handle point to the corresponding target point. As can be observed from the examples (Figure 1), this process is complicated and profound as it respects the shape, texture and continuity of the underlying object.

DragGAN ensures the object parts highlighted by the handle points (e.g., the nose and jaw of the lion) are exactly moved to the desired target locations, and it performs necessary deformations **in highly reasonable manners** — no crazy twists or impossible elongations whatsoever. Another amazing thing is that it performs what the authors referred to as "out-of-distribution image editing" by inferring contents that do not exist in the original image (e.g., the lion's teeth and tongue). Furthermore, it allows the users to optionally provide a binary mask, which indicates the region over which image editing is allowed.

This technical deep dive aims to closely inspect **the methods section** of the DragGAN paper, and explain it to the level of detail that a researcher or student with sufficient practical experience in deep learning and generative models shall be able to comprehend and implement the method. Due to laziness, we have not tried on our own, but we believe we shall be able to based on our understanding.

2 Preliminaries

Most of you shall be already familiar with GAN, StyleGAN and StyleGAN2. Please feel free to skip this section in that case. We do not aim to provide too much additional background beyond what was mentioned in the DragGAN paper.

2.1 Generative Adversarial Networks (GANs)

As a quick recap, the vanilla GAN [2] consists of a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$. $\mathcal{G}$ generates an image of shape $H \times W \times 3$ from a d-dimensional noise vector. The discriminator takes in both the real images and the fake, generated images, and tries to distinguish them. $\mathcal{G}$ and $\mathcal{D}$ play a two-player minimax game, where the former tries to fool the latter while the latter tries to catch the former. If trained properly, $\mathcal{G}$ shall eventually learn to generate highly realistic images.

More details can be found in the GAN paper [2].

2.2 StyleGAN and StyleGAN2

An overall understanding StyleGAN/StyleGAN2 [3, 4] is slightly important because DragGAN is architecturally based on StyleGAN2. On the other hand, it is not critical to understand them in great details if your goal is to understand the methods section of DragGAN.

The following section is quoted from the DragGAN paper that describes the relevant background on StyleGAN/StyleGAN2:

> In the StyleGAN2 architecture, a 512 dimensional latent code $z \in \mathcal{N}(\mathbf{0}, \mathbf{I})$ is mapped to an intermediate latent code $w \in \mathbb{R}^{512}$ via a mapping network. The space of w is commonly referred to as $\mathcal{W}$. w is then sent to the generator $\mathcal{G}$ to produce the output image $I = \mathcal{G}(w)$. In this process, w is copied several times and sent to different layers of the generator $\mathcal{G}$ to control different levels of attributes. Alternatively, one can also use different w for different layers, in which case the input would be $w \in \mathbb{R}^{l \times 512} = \mathcal{W}^+$, where l is the number of layers. This less constrained $\mathcal{W}^+$ space is shown to be more expressive. As the generator $\mathcal{G}$ learns a mapping from a low-dimensional latent space to a much higher dimensional image space, it can be seen as modelling an image manifold.

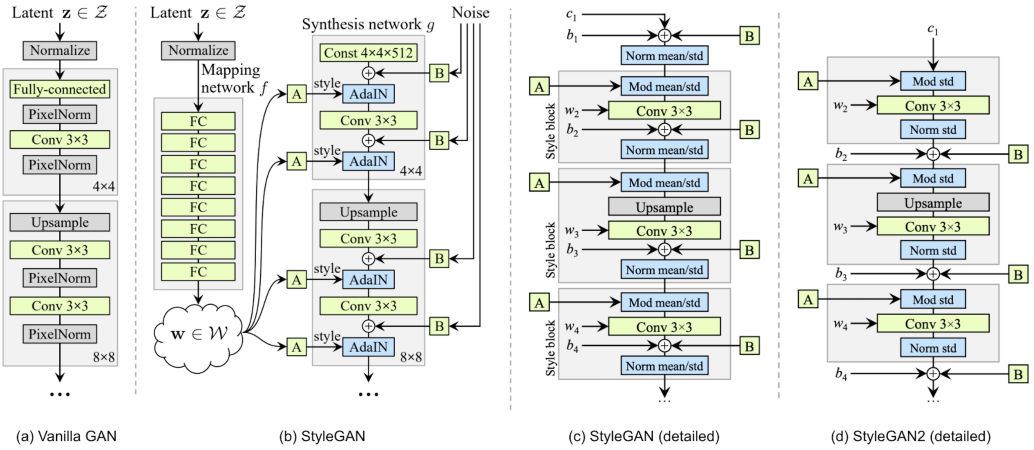

Figure 2: **Comparison between vanilla GAN, StyleGAN and StyleGAN2.** This is slightly re-made from Fig. 1 in [3] and Fig. 2 in [4].

The key difference between a vanilla GAN and a StyleGAN is illustrated in Figure 2(a) and (b). While the former directly decodes a noise vector z over a series of convolutional blocks into a generated image, the latter first transforms z into an intermediate latent vector w, and then decodes w progressively over several resolutions. The image resolution progressively increases from 4×4 all the way to 1024×1024.

In StyleGAN2, the authors reworked the design of the synthesis network (Figure 2(c)), and isolated the adaptive instance normalization (AdaIN) operations into two consecutive parts. After some recombination, they introduced the new fundamental building block termed the **"style blocks"** (gray box in Figure 2(d)). The outputs from each style block is referred to as the **feature map** at that particular resolution.

More details can be found in the StyleGAN and StyleGAN2 paper [3, 4].

3 Methods of DragGAN

3.1 What DragGAN is and What It isn't

Before we dive deep into the details, here are some high-level summaries on the DragGAN method:

- It is not a new architecture. It is not a new training technique. It is a **latent space manipulation technique on a pre-trained StyleGAN2**.
- This manipulation technique requires some optimization, but not on the model itself, so it is not a method for "training". The optimization is performed on the feature map.
- In short, DragGAN proposes **a technique to manipulate the feature map** based on user input, and the manipulated image is generated by the pre-trained, weight-frozen StyleGAN2 using the manipulated feature map.
- Notably[2], the latent space manipulation is **not** performed ***directly*** on the feature map, but rather performed ***indirectly*** via optimization on the latent code (i.e., the vector serving as the input to StyleGAN2). *The latent code is the only thing that DragGAN directly modifies.*

Up to this point, there has been nothing difficult to comprehend. Now comes the harder part — the juicy meat of DragGAN — **motion supervision** and **point tracking**.

3.2 Unspoken Assumptions

Before talking about motion supervision and point tracking, we want to point out some major assumptions that the authors used without much explanation. These assumptions are implied to be true based on the fact that DragGAN works so well, but we nonetheless believe that it helps understanding if we point them out explicitly.

1. *The output image and the intermediate feature maps have a strong pixel-by-pixel correspondence if they are resampled to the same resolution.*

   This is a key assumption without which the entire DragGAN method is completely nonsense. This assumption can be shown to be true by analyzing the operations (convolution, upsampling, adaptive instance normalization) between the feature maps and the output image.

2. *The pre-trained StyleGAN2 has a good enough latent manifold such that a slightly and smoothly manipulated feature map shall result in a reasonable and realistic image.*

   This ensures the manipulated output image looks reasonable, and from a hindsight it seems true. However, we are unsure whether a more mathematically rigorous statement of this assumption can be / has been proven, since we are not experts in this topic.

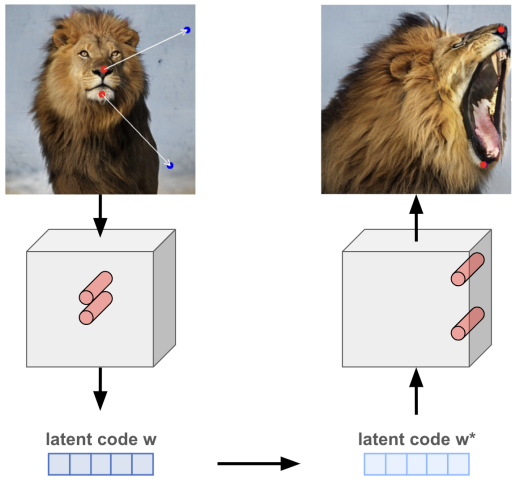

Figure 3: **An overly simplified sketch of the motion supervision and point tracking procedure.** Top row are images and bottom row are feature maps. This figure uses materials from Fig. 9 in [1].

[2]Many thanks to an anonymous friend for pointing this out. This bullet point is added for clarification.

3.3 Motion Supervision and Point Tracking

Purpose of motion supervision and point tracking In plain language, motion supervision and point tracking aims to manipulate the feature map (defined in section 2.2) such that the region around the handle points in the original feature map are "smoothly migrated" to the region around the corresponding target points in the manipulated feature map. An overly simplified sketch is provided in Figure 3. This sketch shall only be used for high-level conceptual understanding.

Why this achieves the purpose Prior to manipulation, the feature map is resampled to the same height/width as the output image using bilinear interpolation. Due to the first unspoken assumption, the object parts around the handle points in the original image will be moved to the target points in the manipulated output image, achieving the key purpose of DragGAN. Due to the second unspoken assumption, the manipulated output image will be reasonable and realistic.

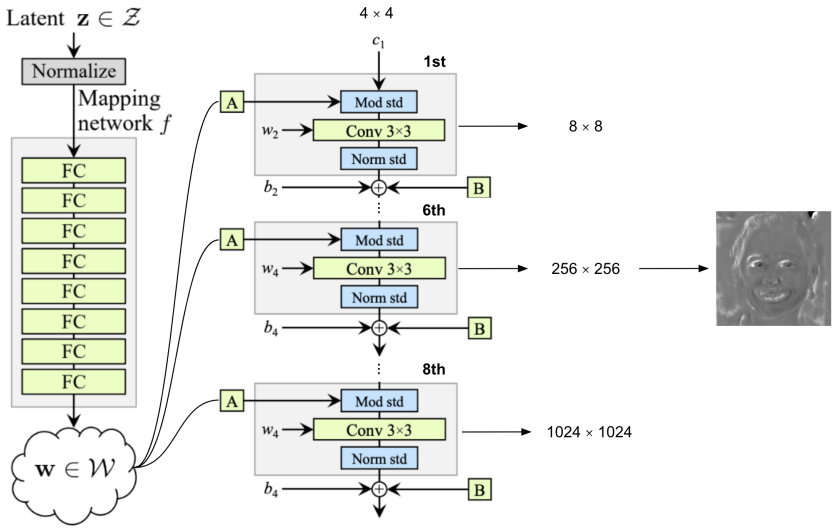

Figure 4: **The manipulated feature map is the output of the 6th style block.** StyleGAN2 style blocks starts from a 4×4 input c_1 and ends at a 1024×1024 feature map, with height/width doubled each block downward. The 6th style block shall have resolution 256×256. Please note that this figure only shows the heights/widths of the feature maps while not showing the channel dimension. This figure is modified from the previous figures.

Which feature map to manipulate As we have previously mentioned, the output of each style block of StyleGAN2 is a distinct feature map at that respective resolution. Which feature map does DragGAN manipulate? Section 3.2 of the DragGAN paper answers this question: it is the feature map at the 6th style block, which corresponds to the 256×256 resolution. This is illustrated in Figure 4. The DragGAN authors claimed this feature map shows a good trade-off between resolution and discriminativeness.

Details of motion supervision Let us denote the handle points as $\{p_i\}$ and the target points as $\{t_i\}$, with $i \in [n]$ if we have a total of n (handle, target) pairs. Let us further denote the points inside the circle centered at p_i with radius r_1 as $\Omega_1(p_i, r_1)$. Motion supervision is an iterative optimization process that manipulates the feature map $\mathbf{F}$ such that the region around each p_i is migrated to the region around t_i.

Probably to enforce smoothness and continuity on the feature map, instead of a brutal cut-and-paste or copy-and-paste from $\Omega_1(p_i, r_1)$ to $\Omega_1(t_i, r_1)$, the DragGAN authors decided to use a gentler method. They first find the unit vector pointing from p_i to t_i:

$$d_i = \frac{t_i - p_i}{||t_i - p_i||_2} \tag{1}$$

then they define the following loss to guide the iterative update on the feature map:

$$\mathcal{L} = \sum_{i=0}^{n} \sum_{q_i \in \Omega_1(p_i, r_1)} ||\mathbf{F}(q_i) - \mathbf{F}(q_i + d_i)||_1 + \lambda ||(\mathbf{F} - \mathbf{F_0}) \cdot (1 - \mathbf{M})||_1 \tag{2}$$

The second component is very easy to understand. $\mathbf{M}$ is the binary mask defining the editable area of the image[3]. $||(\mathbf{F} - \mathbf{F_0}) \cdot (1 - \mathbf{M})||_1$ simply enforces a constraint on the region outside the editable area to demotivate any change of the feature map. From the first unspoken assumption, we can deduce that, over the regions where the feature map is not changed, the generated image shall remain the same (at least shall not be changed by much) —- this is indeed supported by the DragGAN results.

The first component is a bit more complicated. $\mathbf{F}(q_i)$ denotes the feature map at point q_i, which is within the circular neighborhood of the handle point p_i. $\mathbf{F}(q_i + d_i)$ denotes the feature map at point $q_i + d_i$, which is q_i shifted toward t_i by a unit distance. $||\mathbf{F}(q_i) - \mathbf{F}(q_i + d_i)||_1$ encourages the feature map to be similar at the two specified points. However, the authors further specified that the gradient shall be detached at $\mathbf{F}(q_i)$, which means that the feature map can only be updated at $q_i + d_i$ but not at q_i. This ensures a uni-directional migration of the feature map content from q_i towards t_i but not the other way around.

Another tiny yet important detail to point out is that while the target points are fixed once given by the users, the handle point p_i is actually **moved** after each optimization step. This also explains why the first component is written as $||\mathbf{F}(q_i) - \mathbf{F}(q_i + d_i)||_1$ instead of $||\mathbf{F_0}(q_i) - \mathbf{F}(q_i + d_i)||_1$. The equation emphasizes that the motion supervision process iteratively migrates the feature map contents around **the handle point at the current optimization step** (not **the initial handle point**) towards the target point.

Will motion supervision cause feature duplication At first we thought this loss function will result in two identical copies of $\mathbf{F}(\Omega_1(p_i, r_1))$ at $\Omega_1(p_i, r_1)$ and $\Omega_1(p_i + d_i, r_1)$. However, at closer inspection, we believe this will not happen. The main reason is that, although the feature map at q_i cannot be updated due to the detached gradient when the loss is back-propagated for q_i, it will nevertheless be modifiable when the loss is back-propagated for a different $q_i' \in \Omega_1(p_i, r_1)$. Excuse the abuse of notation, as the original paper did not need to subset different points inside $\Omega_1(p_i, r_1)$.

Well, one may ask, here we only show that $\Omega_1(p_i, r_1)$ ***can*** be modified, but why ***will*** it be modified? We further explain this with an illustration (Figure 5). The red and blue circles represent $\Omega_1(p_i, r_1)$ and $\Omega_1(p_i + d_i, r_1)$ respectively — using the same convention as Fig. 3 in [1]. Let's look at 3 pixels: a, b and c. When we perform motion supervision on b, it will encourage b and c to become similar, and since b is not modifiable, this will encourage c to look like b. Similarly, b will look like a when motion supervision is performed on a. As a result, the feature map content at (a, b) will be propagated to (b, c). This behavior is further ensured by the fact that d_i is a unit vector which means $\Omega_1(p_i, r_1)$ and $\Omega_1(p_i + d_i, r_1)$ have significant overlap.

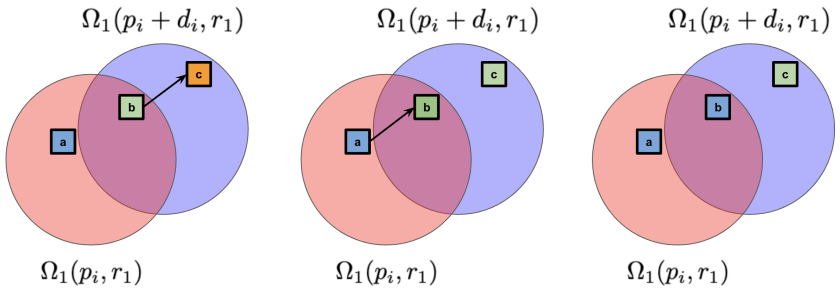

Figure 5: **Illustration of feature map propagation.** We show how the feature map content around $\Omega_1(p_i, r_1)$ is propagated to $\Omega_1(p_i + d_i, r_1)$ under motion supervision. However, this is a conceptual illustration, while the precise pixel-to-pixel propagation cannot be guaranteed.

[3]In case the binary mask is not specified by the user, we believe an all-ones binary mask is implied.

One may further ask, what if motion supervision is performed on a before it is performed on b? Will that make all three pixels blue? The answer is no. In reality, the motion supervision is performed simultaneously on all such pixels during back-propagation, after the loss is computed for all pixels inside the circle. Hence there is no need to worry about this kind of "pollution".

In summary, the feature map contents around the original handle point will migrate towards the target point after each optimization step, and after sufficient number of optimization steps they will reach the target point. The potential "feature duplication" is prevented due to aforementioned reasons.

However, this statement is only true if there is a way to track the points along the optimization process, updating the location estimation for p_i at each optimization step. Even though the loss function encourages the propagation from each q_i to $q_i + d_i$, **the precise pixel-perfect propagation cannot be guaranteed**. This leads us to the point tracking method.

Details of point tracking As we mentioned above, the handle points are "updated" at each optimization step. More precisely, after each optimization iteration that manipulates the feature map, we need to re-localize the handle points in order to proceed to the next iteration. This re-localization job is performed by point tracking.

Unlike some prior works that track the points in the image space, the DragGAN authors decided to track the points in the feature space, again leveraging the first unspoken assumption. Let us denote the points inside the square patch[4] centered at p_i with side length of $2\,r_2$ as $\Omega_2(p_i, r_2)$. Let us additionally denote (beyond the notation of the DragGAN paper) p_i at the k-th iteration as p_i^k, and the feature map at the k-th iteration as $\mathbf{F_k}$. We can (using a slightly different notation from the DragGAN paper) define the following rule for point tracking:

$$p_i^{k+1} = \underset{q_i \in \Omega_2(p_i^k, r_2)}{\arg\min} ||\mathbf{F_{k+1}}(q_i) - \mathbf{F_0}(p_i^0)|| \tag{3}$$

This is a simple nearest-neighbor search over the square patch that looks for the point which is most similar to the initial feature map at the initial handle point. While we could potentially replace $\mathbf{F_0}(p_i^0)$ with $\mathbf{F_k}(p_i^k)$, we believe that the authors of DragGAN used the current formulation to mitigate drift error.

4 Discussion

DragGAN is a fascinating work that has recently gained remarkable popularity. As much as we wish everyone could read the DragGAN paper with zero effort, we suspect that might be too good to be true. Since we, researchers in an adjacent area, found it a bit challenging to parse the technical details in the first pass and indeed spent significant time self-correcting our interpretation of the DragGAN methods, we believe it may be valuable to pass on our interpretation to the broader audience who find that paper interesting. We will be very glad if we can help out at least a few peers who would love to dig into the technical details that is not covered by the few Medium posts available online.

Please don't forget that we are not the authors of DragGAN, and this article is by no means an official interpretation of the paper. Please don't hesitate to point out mistakes and misinterpretations. If you want to cite DragGAN, please cite the correct paper using the format mentioned in their official GitHub repository: `https://github.com/XingangPan/DragGAN`.

References

[1] Pan, X. *et al.* Drag your gan: Interactive point-based manipulation on the generative image manifold. *ACM SIGGRAPH 2023 Conference Proceedings* (2023).

[2] Goodfellow, I. *et al.* Generative adversarial networks. *Communications of the ACM* **63**, 139–144 (2020).

[4] Or equivalently, you can view this as a L1 circle. The authors probably thought of it this way. Otherwise why would they use the radius symbol r for definition?

[3] Karras, T., Laine, S. & Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 4401–4410 (2019).

[4] Karras, T. *et al.* Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 8110–8119 (2020).