Generating Synthetic Data for DNA Origami-based Information Storage Systems

William Hunter[1], Chandler Low[1] and Thomas Heinis[1]

[1]*Imperial College London, Exhibition Rd, South Kensington, London SW7 2BX*

Abstract

The impending information storage crisis is coming. Typical archival storage mediums must be replaced and re-written via expensive processes every 3-5 years. Exponentially increasing global data creation demands innovation in the methods we use to store this data. DNA oligos/origami offer a novel way to store information that is more durable, higher density, and scalable. With DNA information storage pipelines in their infancy, many areas of developmental progress are underway. One such area that requires attention is the back-end decoding and processing of the stored information. As the building material of DNA provides a continued and increasing scope to be used as the next-generation information storage medium, significant challenges arise. One such challenge is the cost-barrier to producing real microscopic images of folded DNA origami structures, which can amount to many thousands of pounds/dollars even if your institution is equipped with a well-resources bioengineering laboratory. Another barrier is time; real images can take weeks or months to produce. We therefore present a method for generating synthetic datasets for origami-based DNA information storage systems using a custom GAN pipeline. We find a customised ProjGAN the best performer for generating microscopy-based synthetic images of data stored in DNA origami structures. The work presented here therefore acts a useful tool for the further development of computational and back-end decoding pipelines for DNA origami information storage, that can now be developed without the aforementioned barriers. With DNA origami storage methods already here and many advancements on the horizon we are excited to present our synthetic data generation tool.

Keywords

DNA Storage, DNA Origami, Information Storage, Machine Learning, GAN, Data Science

1. Introduction

The collection and use of data have played a crucial role in modern society and continues to become more important. As of 2021, it is estimated that the total volume of data created and consumed worldwide is 74 trillion terabytes per year and this trend continues to grow exponentially [1]. As the rate of data creation and consumption increases rapidly, it is important to think about how we can store this data. Currently, the most common storage mediums are hard drives, magnetic tape and solid-state or flash storage. These are all categorised into silicon and alloy-based hardware, which have negative economic and environmental ramifications [2]. With our growing data demands, these methods of storing data are becoming ever less

BICOD21: British International Conference on Databases, December 09–10, 2021, Imperial College London, London, UK

william.hunter18@imperial.ac.uk (W. Hunter); ting.low17@imperial.ac.uk (C. Low); t.heinis@imperial.ac.uk (T. Heinis)

https://dnastorage.doc.ic.ac.uk/ (W. Hunter)

viable. Therefore, we must find alternatives to keep up with our growing data requirements. In recent times the viability for DNA (deoxyribonucleic acid) as an alternative solution to the incoming cold-storage crisis has increased. As present, two main systems of DNA information storage exist; Oligo-based storage [2] and DNA origami storage [3], where the former uses base-pair encoding to store information in sequences, and the later encodes the information within the folded DNA nanostructure. Technologies and computational pipelines for decoding data stored in oligos has been well developed [4][5][6], whereas the same cannot be said for the decoding of information stored in DNA origamis or nanostructures. DNA origami consists of folded nanostructures via self-assembly, following some architectural design with a scaffold and staples, for genesis work see [7][8]. Images of folded nanostructures are shown in figure 1, where *dots* (accumulated DNA strands or gold-nanoparticles) represent bit locations within the encoding architecture of the structure. NB. our work on this is incomplete and publication is upcoming. Although there are multiple ways of storing data within the origami, the general premise is that the dots represent bit locations in a designed fashion [3].

Generative Adversarial Networks (GANs) have become an exciting field of research within deep learning and generative modelling. GANs are models that contain two networks, known as the generator and discriminator, that are pitted against each other in a zero-sum game. The discriminator is a binary classifier denoting whether a sample is either one originating from the data or one produced by the generator.

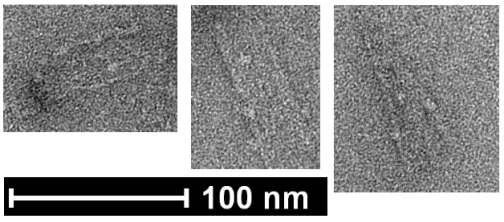

Figure 1: TEM images of real DNA origami with encoded data, produced within our research group at Imperial College London. From left to right there are one, two and three dots of information representing active bit locations.

In an effort to aid the development of computational pipelines for decoding data stored in folded DNA origamis we present a method for generating synthetic datasets for DNA origami data storage. We are able to produce such a method due to the recent advancements in GANs [9][10] and general machine learning, thus enabling us to fine-tune a computational method to produce usable and trainable data specific to DNA origami nanosturctures.

2. Method

In order to produce a large and diverse enough dataset of images to be used to train GANs we produce a python script that mimics photoshop manipulation of dot/bit locations onto a base image. Each image is 135 x 90 pixels. The dataset is composed of 10 classes, with each class corresponding to the number of dots contained within the image. There are a total of 100,000 images with 10,000 images for each class. Such a dataset could be replaced with real DNA images although this is unlikely at present due to the bottlenecks in throughput that currently imaging techniques, TEM/AFM, contend with. This data therefore provides the GAN training set.

The original GAN [9] uses two fully connected networks with ReLu activations for their architecture. The images produced by this implementation are not as refined as they need to be for our use case. We therefore must consider other GAN architectures and methods. DCGAN employs convolutions in the layers for both generator and discriminator in addition to batch normalisation and leaky ReLu activations. Utilising the capabilities of CNNs, the architecture proposed by Radford et al. [11] perform particularly well in tasks related to images. However, in some instances this architecture suffers from mode collapse. In an ideal scenario, a GAN should produce a variety of distinct outputs. However, the generator may occasionally produce especially good outputs which causes the generator to learn to produce only a small set of outputs. The best strategy for the discriminator is to reject this small range of outputs. However, the discriminator can be held at a local minima. This creates a scenario, known as mode collapse, in which the generator only optimises for a particular discriminator and produces similar outputs at each iteration. The cause of this scenario is usually a weak discriminator, where the discriminator remains at a local minima and fails to improve, leading to the generator becoming "lazy" and producing a less varied output. Arjovsky et al. [12] propose WGAN, which uses the Wasserstein Distance rather than the Jensen-Shannon Divergence for computing the losses for the generator and discriminator. The use of the Wasserstein loss is used to train the generator and discriminator, with the aim to increase the distance between scores for real and fake images. Discriminator Loss: $E[D(x)] - E[D(G(z))]$, Generator Loss: $-E[D(G(z))$. This loss means that larger scores correlate to fake images and lower scores to real images. Larger scores on real images incur a large penalty on the discriminator, which encourages the production of lower scores on real images. In turn, a larger score on fake images incurs a small loss for the generator, thus encouraging the production of higher scores on fake images. By utilising the Wasserstein loss, the discriminator can train optimally without being impeded by vanishing gradients, which ensures that it does not get trapped at a local minimum. The discriminator would be able to reject the generator output if it was continually producing a singular output, forcing the generator to adapt and produce a varied output. Miyato et al. [13] integrated hinge loss into their networks and in doing so, improved the overall performance of their networks. They used the following equations for loss: Discriminator Loss: $E[\min(0, -1 + D(x)] - E[D(G(z))]$, Generator Loss: $-E[D(G(z))]$. Conditioning on GANs utilises auxiliary data for contextual information to improve performance and output. The resulting network has two advantages: 1. The networks converge faster, as the additional information gives some basis for a distribution that the networks can fit upon. 2. The additional information can be used to control the output of the generator. i.e. using the labels to determine the class output of the generated sample. The cGAN utilises class labels from the data and concatenates them to the latent vector for generators and images for discriminators [10]. This helps the two networks learn on additional data, which improves the quality of the images produced. Expanding on the conditioning used in cGANs, the Projection Discriminator [14] seeks to condition the discriminator using a projection rather than via concatenation. In PyTorch, this exists in the form of an embedding layer which allows the discriminator to condition more accurately using a projection rather than a vector of labels. We perform conditional batch normalisation (CBN), $\hat{x}_i = \gamma(x_i - \mu)/\sigma + \beta$ where μ and σ are the mean and standard deviation of a batch and γ and β are hyper parameters used for conditioning, to replace batch normalisation layers within our generator to enable conditioning [15]. The exploration of GAN architectures results in our final architecture presented in figure

2.

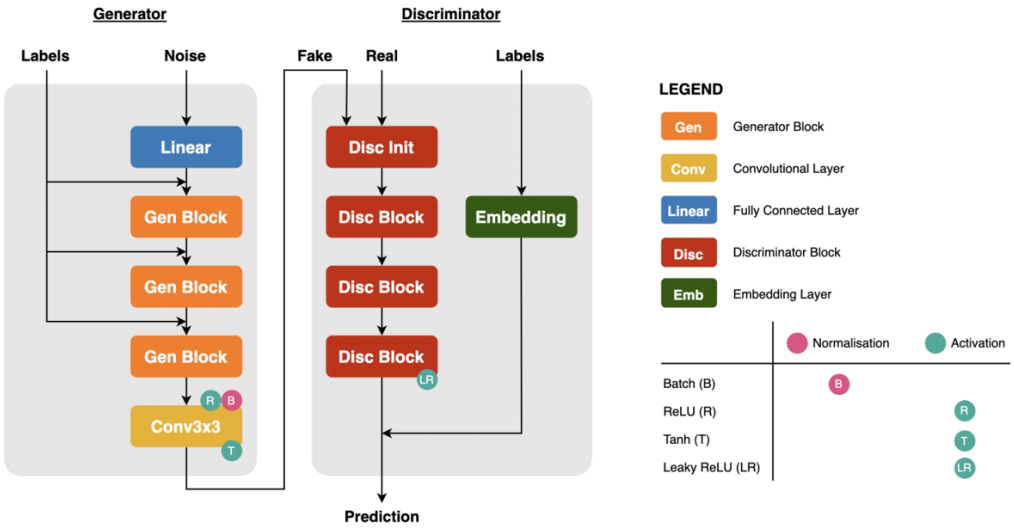

Figure 2: Custom GAN architecture for symthetic DNA origami information storage data generation. The generator feeding into the discriminator resulting in a prediction.

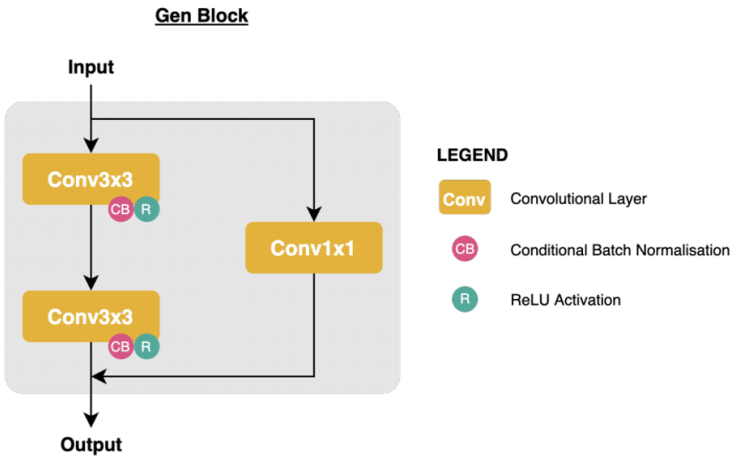

Figure 3: The architecture inside a Gen Block featuring a conditionally batch-normalised 2-convolutional layer and a 1-convolution.

The generator feature map size is defined to be 256. The generator itself is composed of a fully connected linear layer and then 4 generator blocks Figure (3), with the generator block only having one convolutional layer and a Tanh activation. The generator block is inspired by the residual block found in ResNet [16], with the skip connection passing through a 1×1 convolution. The rest of the block contains two 3×3 convolutional layers that employ

conditional batch normalisation for conditioning and ReLu as activations.

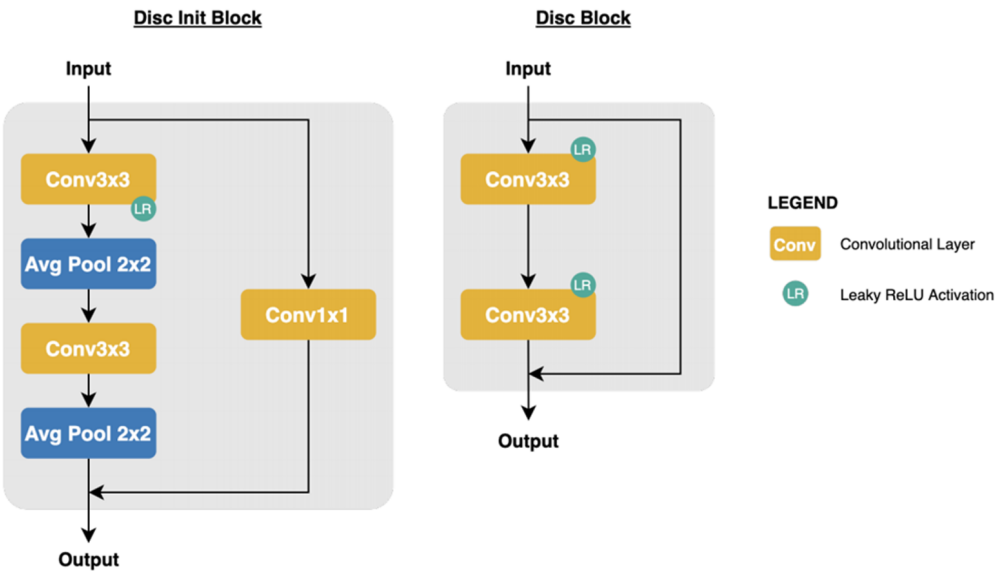

Figure 4: The architecture inside a Disc Block.

The discriminator feature map size is defined to be 128. The discriminator is composed of 4 discriminator blocks (Figure 4) and a final linear layer. The discriminator also has an embedding layer used on the data labels and this is used to provide conditioning to the network. Like the generator, the discriminator is built using our defined discriminator block. The discriminator block also uses skip connections to mitigate the effects of vanishing gradients. The structure of the block remains similar to the generator block, however the activation and average pooling layers are before and after the skip connection.

3. Results

For our GAN experimentation, we tested 4 models: DCGAN, WGAN, HingeGAN (WGAN with hinge loss), and ProjGAN (HingeGAN conditioned using a projection discriminator and conditional batch normalisation). With the exception of DCGAN, the other GANs are trained using iterations rather than epochs. There are 10,000 iterations for generator training with 5 discriminator training iterations for each generator iteration. An iteration is equivalent to one mini-batch from the data loader. The DCGAN is trained for 25 using binary cross-entropy loss.

The images displayed in Figure 5 are samples from the generator of each GAN variant. The samples in DCGAN and WGAN appear to be randomly generated noise. From these generated images, we can see that ProjGAN produces an image that approximates the dataset most convincingly. Looking at the output of HingeGAN, the image suggests that the generator is making improvements to produce realistic images but hasn't improved enough, leaving the blue line at the bottom. We can hypothesise that 10,000 iterations was not enough to allow

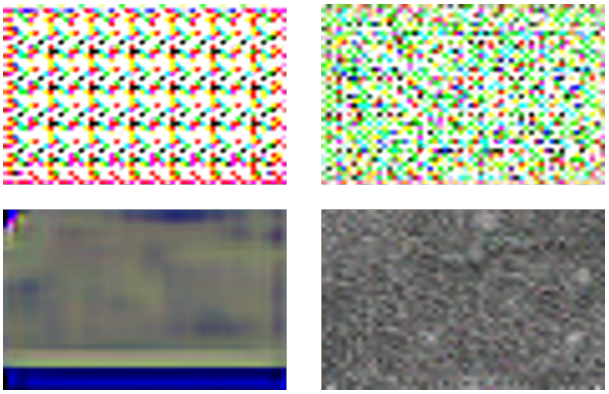

Figure 5: Sample images produced by the generators of different GAN-types at iteration 10,000. *top-left* - DCGAN, *top-right* - WGAN, *bottom-left* - HingeGAN, *bottom-right* - ProjGAN.

HingeGAN to converge and produce more realistic images. The output could also suggest that HingeGAN has experienced a form of mode collapse, where the discriminator is weak and is stuck at a local minima. The images produced by the DCGAN and WGAN could be a sign of mode collapse but they are more likely to indicate that the discriminator is too good. This prevents the generator from improving and causes the network to stagnate and produce random noise.

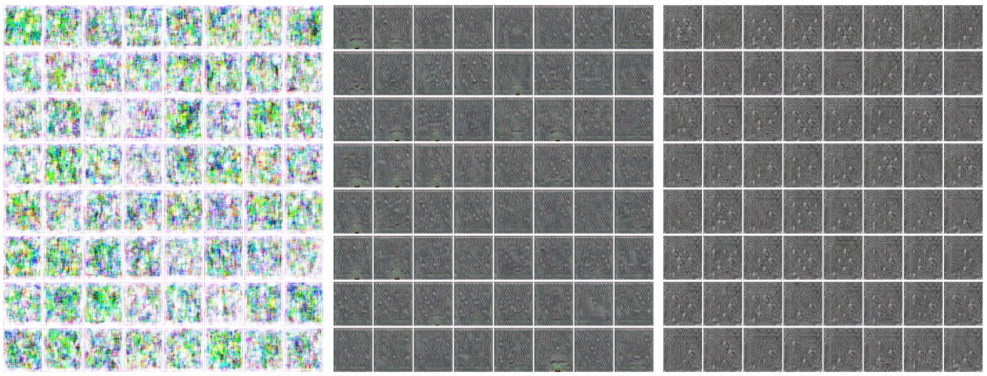

Figure 6: Evolution samples of ProjGAN: *left* - interation 0, *middle* - interation 5000, *right* - iteration 10,000.

Figure 6 depicts the intermediate outputs of ProjGAN over 10,000 iterations. We see that by iteration 5,000, the generator is starting to produce some images that resemble the data but are lacking fully-fledged features. By the $10{,}000^{\text{th}}$ iteration, we see that the generator has produced many samples that look as though they are from the original data. Given that ProjGAN produces images that are effectively indistinguishable from the authentic DNA images, there was no reason to continue with further iterations.

Figure 7 shows comparisons between samples generated by ProjGAN and a collection of specimens from the dataset. The fake images appear to be slightly more blurry than their real counterparts. Furthermore, the dots within the fakes are not as prominent as in the real images.

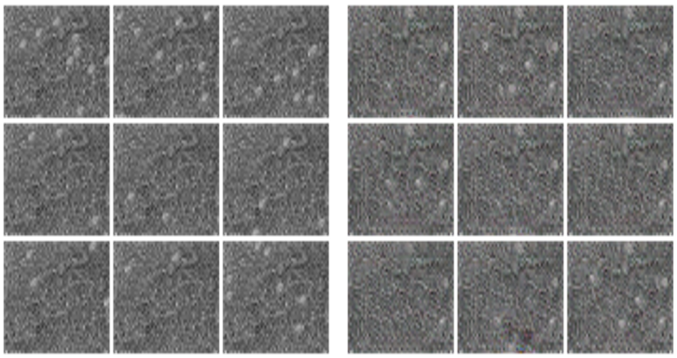

Figure 7: ProjGAN image sample at convergence where *left* - real, *right* - fake.

Table 1
FID scores of GAN architecture performance

Model	FID
ProjGAN	124.55
HingeGAN	520.99
WGAN	373.65
DCGAN	305.40

Finally, there are a few distinct blemishes in some of the fake images that indicate that they are not part of the data. From our visual comparison, we determined that the ProjGAN model performed better than the other architectures. To get a more conclusive perspective of this performance gap, we used an FID score to measure the quality of the generated images against the original data. We calculated the FID score using a PyTorch implementation of the metric and sampled 10,000 images each from the generators and the dataset. The results for the FID score are found in Table 1.

4. Discussion

From the analysis of the FID scores we see that ProjGAN has the lowest FID score of all of the tested models, reinforcing our conclusion that it is the best-performing model. An interesting observation we can make from the FID scores, is that of the HingeGAN. Subjectively, the images generated by the HingeGAN appear to be more similar to the data than the images produced by DCGAN and WGAN. However, when observing the FID score, the HingeGan produces the highest score with 520.99 over 100 points above DCGAN and WGAN and over 300 points for ProjGAN. We could justify this by inspecting images from the data, such as when looking at the real samples found in Figure 7. Another observation we have made is that when compared to the generated images, we would expect that DCGAN and WGAN would receive higher FID scores than they did in practice. We hypothesise that this is because the images sampled from the data have backgrounds that resemble noisy images. As the background occupies much of the image, the FID calculation may have deemed this to be similar to random noise found in

generated images and led to a lower FID score.

Previous work has concluded that GANs have the greater potential in producing higher quality realistic images than that produced by variational autoencoders (VAEs) [17]. This can be justified by the fact that the objective for VAEs optimises on the variational lower bound, an approximation of the true distribution of the data. Whereas there is no such bound that exists for GANs, as the approach is optimising for a zero-sum game, giving GANs greater potential in learning the data's underlying probability distribution. For these reasons, we have opted to use GANs to generate synthetic data sets in the final method.

The requirements for our method of generating synthetic data are that you must possess real-DNA origami images, but only a small number of them. With the knowledge that real DNA origami images are timely and costly to produce, your information storage pipeline should be somewhat operational before executing our method, although it is feasible to produce synthetic datasets with purely synthetic data, the hyperparameters of the GAN are likely needed to be tweaked.

5. Conclusion

A pipeline like the one described in this paper is a useful tool for the advancement of computational pipelines within the DNA origami information storage fields. The method's design architecture and implementation flexibility allows for groups and researchers working on different designs/implementations to make use of our resource. Our intention is to use the synthetic data generated in this article to train further networks that are needed in our DNA origami information storage pipeline. The training images used for the GAN are non-authentic insofar as they are pixel manipulations of real/authentic images, therefore the training data is based off authentic DNA images, but not at scale. The method outlined here helps to overcome that specific barrier for a time where large sets of authentic full images are available, which we predict would become available in the near future.

With the data storage crisis incoming, new systems are under development to utilise DNA as a novel storage medium. The potential for DNA to succeed traditional digital storage systems is strong, where DNA origami/nanostructure databases is one of the front runners. Due to the costs of fabrication to many research groups and industry partners, our method for generating synthetic datasets provides a way to further the development of back-end computational pipelines without these initial barriers.

References

[1] J. Gantz, D. Reinsel, The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east, IDC iView: IDC Analyze the future 2007 (2012) 1–16.

[2] R. Appuswamy, K. Le Brigand, P. Barbry, M. Antonini, O. Madderson, P. Freemont, J. McDonald, T. Heinis, Oligoarchive: Using dna in the dbms storage hierarchy., in: CIDR, 2019.

[3] G. D. Dickinson, G. M. Mortuza, W. Clay, L. Piantanida, C. M. Green, C. Watson, E. J.

Hayden, T. Andersen, W. Kuang, E. Graugnard, et al., An alternative approach to nucleic acid memory, Nature communications 12 (2021) 1–10.
[4] S. Chalapati, C. A. Crosbie, D. Limbachiya, N. Pinnamaneni, Direct oligonucleotide sequencing with nanopores, Open Research Europe 1 (2021) 47.
[5] J. Zeng, H. Cai, H. Peng, H. Wang, Y. Zhang, T. Akutsu, Causalcall: Nanopore basecalling using a temporal convolutional network, Frontiers in genetics 10 (2020) 1332.
[6] R. R. Wick, L. M. Judd, K. E. Holt, Performance of neural network basecalling tools for oxford nanopore sequencing, Genome biology 20 (2019) 1–10.
[7] H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, T. H. LaBean, Dna-templated self-assembly of protein arrays and highly conductive nanowires, science 301 (2003) 1882–1884.
[8] P. W. Rothemund, Folding dna to create nanoscale shapes and patterns, Nature 440 (2006) 297–302.
[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, arXiv preprint arXiv:1406.2661 (2014).
[10] M. Mirza, S. Osindero, Conditional generative adversarial nets, arXiv preprint arXiv:1411.1784 (2014).
[11] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint arXiv:1511.06434 (2015).
[12] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein generative adversarial networks, in: International conference on machine learning, PMLR, 2017, pp. 214–223.
[13] T. Miyato, T. Kataoka, M. Koyama, Y. Yoshida, Spectral normalization for generative adversarial networks, arXiv preprint arXiv:1802.05957 (2018).
[14] T. Miyato, M. Koyama, cgans with projection discriminator, arXiv preprint arXiv:1802.05637 (2018).
[15] V. Dumoulin, J. Shlens, M. Kudlur, A learned representation for artistic style, arXiv preprint arXiv:1610.07629 (2016).
[16] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[17] P. Pandey, Deep generative models, URL https://medium.com/@prakashpandey9/deep-generative-models-e0f149995b7c. (2019).

6. Appendix

The following tools can be found at www.dnastoragetools.com with links from https://dnastorage.doc.ic.ac.uk/. The Git repository will remain private until future works are published.