Improving the GLO learning process

Yuval Helman, Jakov Zingerman, Ran Schreiber

Tel Aviv University, August 2019

1 Introduction

Unconditional image generation is the task of learning a mapping from a space, called the *latent space*, to an infinite image domain, by only looking at a finite subset of this domain. This mapping is the desired generator, we may randomly select a latent vector and use the mapping to synthesize an image from our image domain. A highly important property of the generating function, which is necessary for tasks that perform operations in the latent space (e.g. image interpolation), is that euclidean distances between latent vectors would be mapped to perceptually and semantically meaningful differences in the image space. Its private case of mapping interpolations of latent vectors into semantic interpolations in the image space is referred to as *continuity* in this report.

Image generators are useful for a variety of applications. They may be used for synthesizing new training images in cases where data is either lacking or too expensive, as medial data usually is [2, 10]; Such data is necessary for other machine-learning tasks, such as classification. In addition, image generators may be a fundamental building block for other algorithms; For example, the *NAM* algorithm by Hoshen and Wolf that learns an unconditional mapping between two image domains relies on the existence of a generator for the target domain [5].

1.1 Generative adversarial networks (GAN)

The task of image generation may be solved very well using *GAN*, in which two neural networks are simultaneously trained - one is the *generator*, and the second is the *discriminator* which estimates the authenticity of its input images [3]. However, *GAN*s are somewhat problematic. They are commonly very hard to train because the training process is usually highly sensitive to the choice of hyperparameters, which are the parameters that are not trained but are set manually; In addition, in most cases multiple initializations of the weights are required for gaining a fine generator because of the *mode collapse* phenomenon, in which the generator produces samples only from a few parts, called "modes", of the data distribution. These disadvantages have motivated researchers to find non-adversarial alternatives for the task of image generation.

1.2 Generative Latent Optimization (GLO)

One of the most effective non-adversarial algorithms for the task is *GLO* by Bojanowski et al [1]. As our proposed method is based on it, we will present it briefly.

Let us denote the training set by S; In order to find the generating function G, the algorithm finds an optimal latent vector z_x from a predefined latent space $\mathcal{Z}$ (in this project we define it to be $\mathbb{R}^d$ where d is its dimension) for each sample x in the training set. That is, the algorithm jointly learns the latent vectors $\{z_x\}_x$ and the weights of the neural network G. The objective is given by this formula:

$$\arg \min_{G,\{z_x\}_{x\in S}} \sum_{x\in S} loss(G(z_x), x) \qquad s.t. \{z_x\}_{x\in S} \subset \mathcal{Z}$$

The aforementioned *loss* is a weighted combination of the *Laplacian Pyramid Loss* and the l_2 loss; It compares how two given images are different.

Although the *GLO* algorithm is very effective, it has a few shortcomings. The algorithm maintains a latent vector for each image in the training set, and as the set is used altogether during the training process and is not split, the optimization is done over millions of parameters. We believe that it makes the training process much more difficult.

In addition, a major downside of the algorithm is the absence of a trained inverse mapping from the image domain to the latent space; Such a mapping is necessary for tasks that perform operations in the latent space. If the input image for such a task is taken from the training set, the *GLO* algorithm returns its optimal latent vector, and we are done; However, in the general case, we need to find its latent vector by performing a new optimization process: We define the latent vector z of the input image x to be $\arg\min_z loss(G(z), x)$, where $loss$ may be any function that compares how two images are different. This optimization makes the task extremely slow, and also unreliable since the objective is highly non-convex.

2 Our Method

Our method is based on the *GLO* algorithm and tries to improve it - we want it to train both a generator and its inverse mapping, and we want the learning process to be more effective. That is, we want it either to be faster or to achieve more pleasing results in the same training time.

The algorithm is comprised of several stages; we first present the whole process briefly and then discuss each stage in detail. First, the training set is split to some predefined number of subsets; Then, there is a sequence of stages, each one is a modified *GLO* algorithm that focuses on a small portion of the training set; Thereafter, a modified *GLO* algorithm that deals with the whole training set is run. Finally, An inverse mapping model from the image domain to the latent space is trained.

Our algorithm trains two neural networks - an *encoder*, which is a mapping from the image domain to the latent space, and a *decoder*, which is a mapping from the latent space to the image domain.

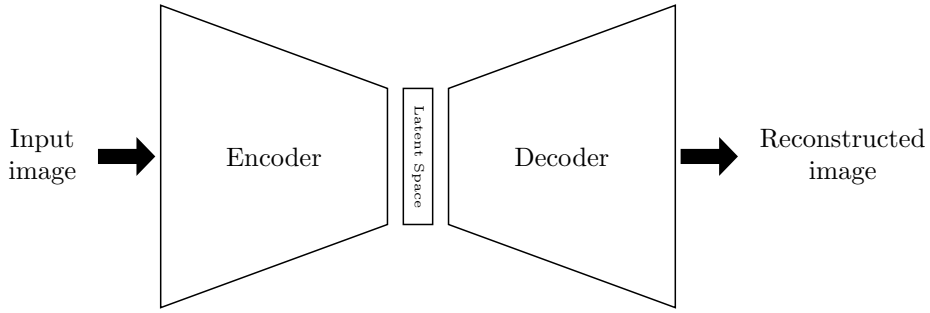

Figure 1: Illustration of the architecture

The *decoder* network, which functions as a generator, is trained in a few stages; Each stage, excluding the last one, involves a relatively small number of images, and thus a relatively small number of parameters that should be learned. We call this notion *incremental learning*, and we believe it may lead to a much more effective training process.

The algorithm has a geometric intuition: the image space may be viewed as a smooth manifold, and the *decoder* may be viewed as a parameterization of this manifold; the algorithm incrementally builds the image space - it starts by learning well a small number of points on the manifold (these points should be scattered so they "cover" the manifold as good as much as possible), then it learns other points, and so forth; Overall, the manifold is built incrementally.

2.1 Splitting the training data

As explained, each subset should "cover" as possible the image space. For achieving this, we perform a multistage process: We first cluster the whole training set to some predefined number of clusters; Next, we select a few representatives from each cluster to be the first set of images (the number of representatives is also predefined); Then we perform these two stages (clustering and selecting representatives) iteratively - each time using the images not selected in the previous stages - and eventually all the remaining images form the last set of images. This sequence of stages yields a sequence of disjoint sets whose union is the whole training set, and the images in each set are perceptually different.

For clustering a set of images we follow the *ResNet Classifier Clustering* proposed by Sage et al. [8]: The last pooling layer of a *ResNet-50* network pretrained on the *ImageNet* dataset is used for extracting useful features of the images; These 2048-dimensional features are dimensionality reduced using the *PCA* algorithm, and finally, we cluster the resulting vectors using the *k-means* algorithm.

The k representatives of each cluster of images are defined to be the k images that correspond to the k dimensionality reduced feature vectors that are nearest to the cluster's centroid (which is the mean of the points that form the cluster); The representatives are calculated using the *k-nearest neighbors* algorithm.

2.2 Modified GLO Algorithms

The *GLO* algorithm is run in three different variants in our algorithm - the first is run at the first stage involving a *GLO* algorithm (that is, the first stage after splitting the data), the second is run at all of the intermediate stages and the third is run at the last stage involving a *GLO* algorithm using the whole training set.

2.2.1 The first variant

The first variant of the algorithm involves the first subset of the training data and is identical to the original *GLO*, except for using the *VGG Perceptual Loss* instead of a weighted combination of the *Laplacian Pyramid Loss* and the l_2 loss. This loss has been shown to estimate the perceptual similarity between two images in a way that approximately coincides with human judgment [11], and is given by this formula:

$$\|x, y\|_{VGG} \stackrel{def}{=} \|x - y\|_1 + \sum_i \|\phi_i(x) - \phi_i(y)\|_1$$

$\|\cdot\|_1$ is the l_1 norm; $\phi_i(\cdot)$ is the vector of features that is extracted at the end of the ith block of a *VGG* network pretrained on the *ImageNet* dataset.

2.2.2 The second variant

The second variant of the algorithm focuses on one subset of the training data while trying to preserve the results of the previous stages. That is, let $n \leq 2$ be the total number of subsets obtained in the first stage of the algorithm. The first subset is processed by the first *GLO* variant described above; For each $1 < i \leq n$ there is a corresponding stage in the algorithm which focuses on it and tries to preserve the results of the previous stages (which trained the first $i-1$ subsets).

It is done as follows: Let us denote the subsets already trained by S_p (p denotes "previous"), and the subset we focus on by S_c (c denotes "current"); The algorithm finds an optimal latent vector z_x for each sample x in S_c, and tries to preserve the good *decoder*'s reconstruction of S_p (which has been obtained in the previous stages). That is, the algorithm jointly learns the latent vectors $\{z_x\}_{x \in S_c}$ and the weights of the neural network G; Of course, these weights are initialized with the learned values calculated in the previous stage. The objective is given by this formula:

$$\arg\min_{G,\{z_x\}_{x\in S_c}} \alpha\cdot\left(\sum_{x\in S_p}\|G(z_x),x\|_{VGG}\right)+\beta\cdot\left(\sum_{x\in S_c}\|G(z_x),x\|_{VGG}\right)$$

For each $x\in S_p$, z_x has been calculated in the previous stage and is now fixed; α and β are hyperparameters which control the significance of each term in the loss function.

2.2.3 The third variant

The third variant of the algorithm involves the complete training set. As in the first variant, the algorithm finds an optimal latent vector for each sample in the training set, but now there is a new term in the loss function which should improve the continuity of the *decoder*. Let us denote the training set by S, the objective is given by this formula:

$$\arg\min_{G,\{z_x\}_{x\in S}} \alpha\cdot\left(\sum_{x\in S}\|G(z_x),x\|_{VGG}\right)+\beta\cdot\left(\sum_{x\in S}\|G(z_x+\varepsilon(\sigma)),x\|_{VGG}\right)$$

α, β and σ are hyperparameters, $\varepsilon(\cdot)$ is a normally distributed random variable with zero mean and variance σ^2.

2.3 Training an Encoder network

After training a *decoder*, an *encoder* is trained as follows: Let us denote the training set by S, the objective is given by this formula:

$$\arg\min_{E}\sum_{x\in S} loss(E(x),z_x)$$

For each $x\in S$, z_x is the optimal latent vector which has been calculated in the training process of the *decoder*; The loss function used is a weighted combination of the l_1 and l_2 loss functions.

3 Experiments

We have conducted several experiments to estimate our algorithm; We have trained a *GLO* model (to which we would compare our models), we have trained our *decoder* without the loss term regarding the generator's continuity (we set its coefficient to zero), and we have trained our *decoder* with this term. Afterward, we have trained an *encoder* matching the *decoder* with the additional loss term.

Model	**Training time** (h:m)
GLO model	09:27
Decoder without the additional loss term	09:05
Decoder with the additional loss term	08:59
Encoder	About 01:30 [1]

Figure 2: The training time of the different models

3.1 The dataset

The dataset we have used is Jon Shamir's frogs dataset which consists of almost eight thousand 64*64 images [9]. The first six thousand images have been used as a training set, and the remaining ones as a testing set; We haven't applied any data augmentation methods.

[1] 5000 epochs have been run in 6 hours and 41 minutes, but the optimum has been achieved at the epoch 1161; Therefore the training time is approximately 1.5 hours.

Figure 3: A few samples from the dataset

As may be seen, the dataset is quite diverse - the frogs vary in their colors, shapes, sizes, and positions.

3.2 Implementation details

The repository of the project is available on Github [2]; It contains all the training and evaluation source files, bash scripts for settings up the environment, different trained models and the results of the experiments conducted. The code is based on the *PyTorch*, *NumPy*, *Matplotlib*, *scikit-learn* and *scikit-image* libraries in *Python 3.7*.

The dimension of the latent space used in the experiments is 100; The *decoder* network used is the generator of the *DCGAN* architecture [7]; The *encoder* network used is a variant of the discriminator of the *DCGAN* architecture - instead of the sigmoid layer in the end, we have a linear layer; The optimization algorithm used is Adam [6].

3.3 Reconstructing training images

The images in the training set have been reconstructed both by a generating function learned using the *GLO* algorithm, and by a generating function learned using our algorithm: The reconstruction of an image x from the training set is defined to be $G(z_x)$, where G is the learned generator and z_x is the optimal latent vector of x that the algorithm computes.

The first row of Figure 3 presents a few frogs images from the training dataset, the second presents their reconstructions using a *GLO* model, the third presents the reconstructions using our model without the loss term regarding the generator's continuity, and the last row presents the reconstructions using our model including this loss term.

[2] https://github.com/ransch/MLApplicationsForGraphics/

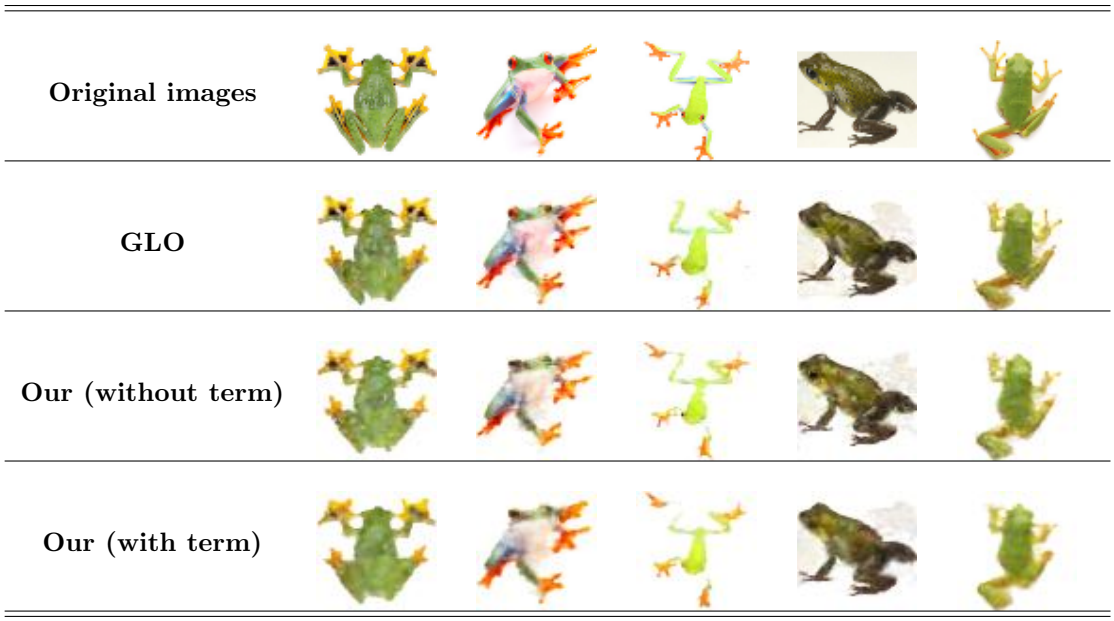

Figure 4: A few reconstructions of training images

Our method hasn't achieved more pleasing results than the *GLO* algorithm; An interpretation of the results and proposed further work may be found at the Discussion Section.

3.4 Interpolating training images

Reconstructions of training images have been interpolated both by a *GLO* model and by our model: Given two images x and y, we take their optimal latent vectors z_x and z_y (computed by the algorithm), split the interval between them to some desired number of points, and apply the learned generating function to each one of these points.

The first row in each section in Figures 4-6 presents an interpolation done by a *GLO* model, the second presents an interpolation done by our model without the loss term regrading the generator's continuity, and the third presents an interpolation done by our model including this loss term. More interpolations may be found at the end of the report.

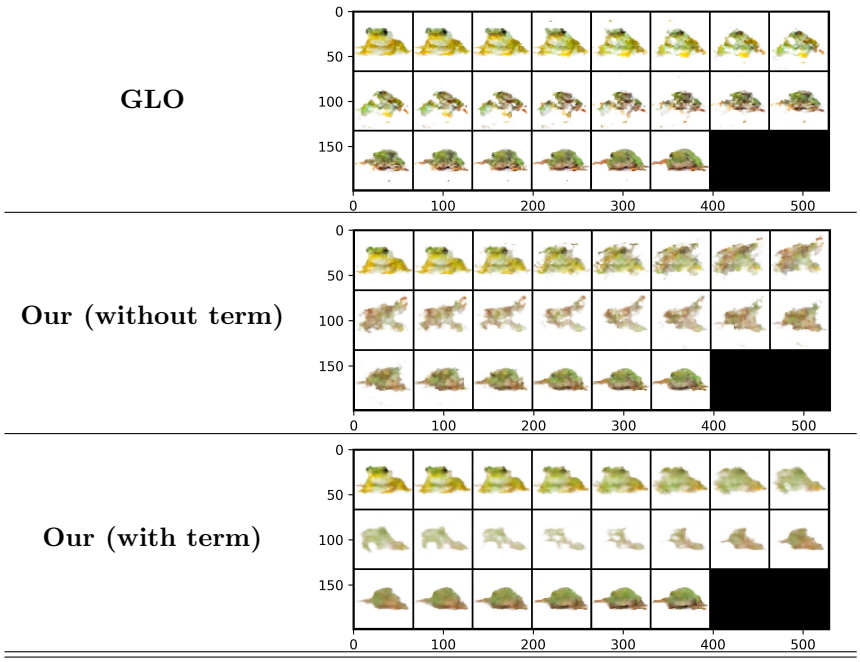

Figure 5: An interpolation of training images (i)

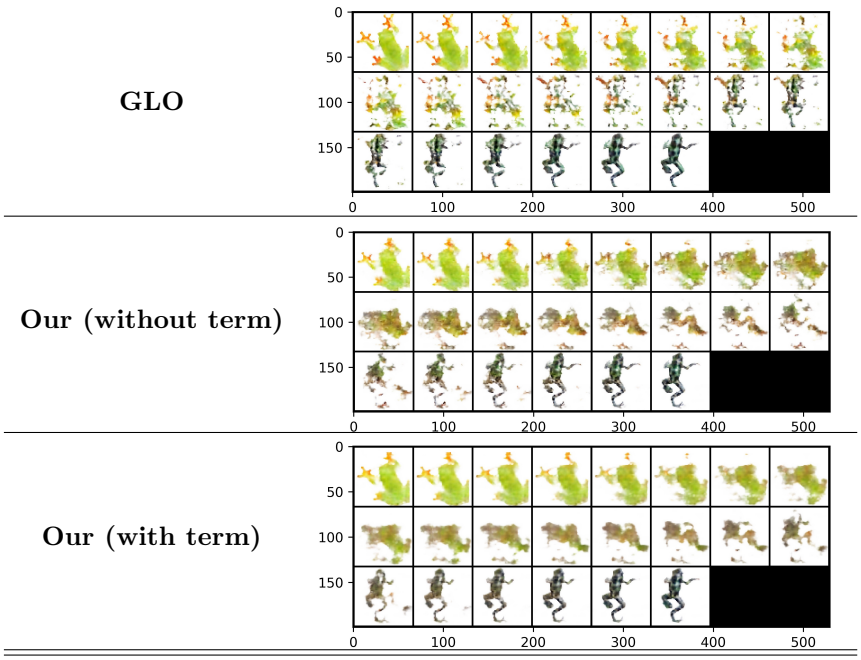

Figure 6: An interpolation of training images (ii)

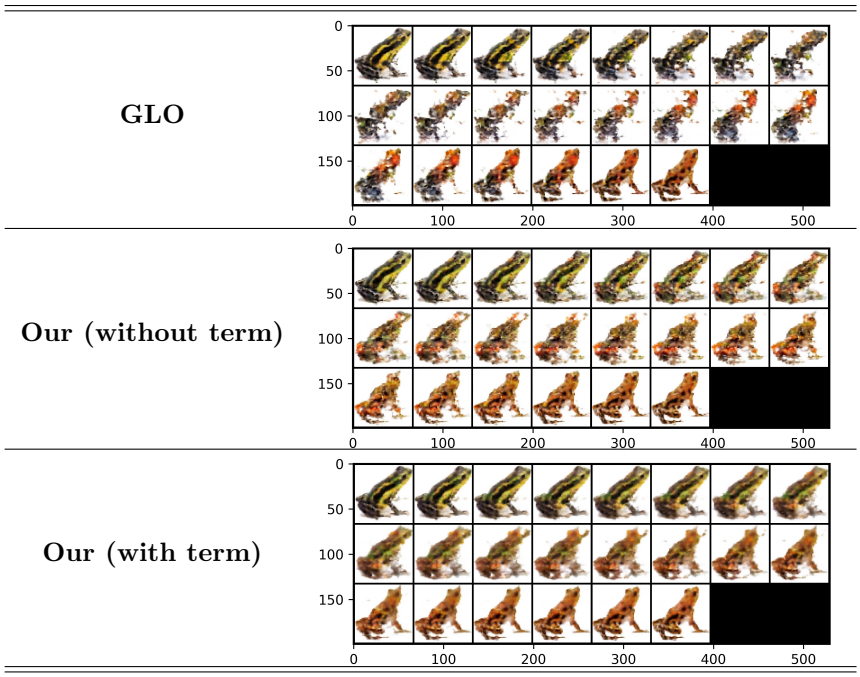

Figure 7: An interpolation of training images (iii)

As can be seen, the interpolations done by our models (both with the loss term regarding the generator's continuity and without it) are perceptually more pleasing than the ones that are done by a *GLO* model. By looking at the interpolations done by the *GLO* model, it seems that the generating function is quite non-continuous, while our models appear to be much more continuous. Moreover, the model which includes the additional loss term seems to interpolate better than the one which doesn't include it.

3.5 Encoding training images

Images from the training set have been reconstructed using an *encoder* and a *decoder*: The reconstruction of an image x is defined to be $D(E(x))$, where D is the learned *decoder* (that is, the learned generator) and E is the learned *encoder*. The *decoder* used is the one which has been trained using the additional loss term (regarding the continuity).

Figure 8: A few reconstructions of training images

As can be seen, the results are yet to be satisfactory with the acquired model - the images have no texture and are blurry. Again, we interpret and propose further work at the Discussion.

3.6 Encoding test images

Images from the test set have been reconstructed using an *encoder* and a *decoder*; The *decoder* used is the one which has been trained using the additional loss term (regarding the continuity).

Figure 9: A few reconstructions of testing images

4 Discussion

In this project, we have tried to improve the *GLO* algorithm by dealing with its downsides discussed above (Section 1.2). As can be seen, The *encoder* has not been trained satisfactorily yet; There may be several reasons for this - the network's architecture might not fit well, and maybe the network should be trained jointly with the *decoder*, and not separately. In addition, our models haven't achieved more pleasing reconstructions of training images than the *GLO* model. However, we have seen that both the notion of *incremental learning* and the additional loss term succeed in improving the quality of the trained *decoder* as it seems to be more continuous.

It may be interesting to repeat the experiments conducted with a different choice of the latent space $\mathcal{Z}$, and to adapt the *GLANN* algorithm by Hoshen and Malik [4] (which is based on *GLO* and outperforms it) to our method.

5 Additional Interpolation Comparisons

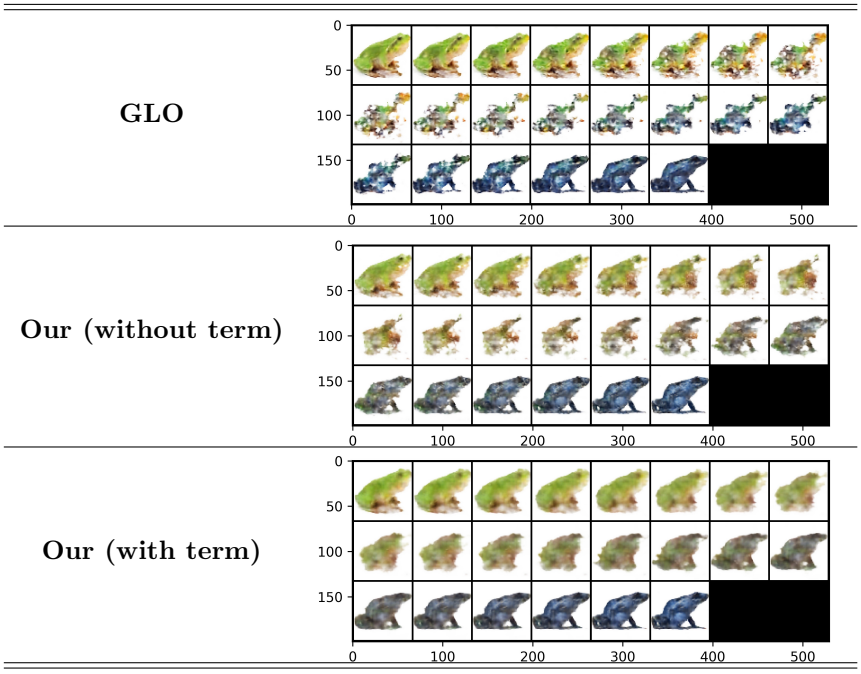

Figure 10: An interpolation of training images (iv)

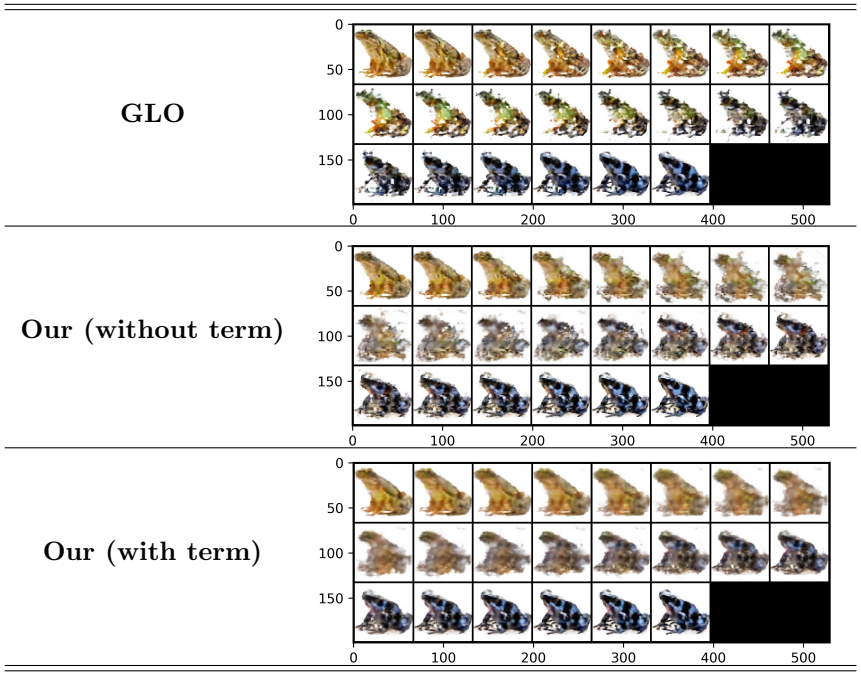

Figure 11: An interpolation of training images (v)

6 References

[1] Piotr Bojanowski et al. "Optimizing the latent space of generative networks". In: *arXiv preprint arXiv:1707.05776* (2017).

[2] Maayan Frid-Adar et al. "Synthetic data augmentation using GAN for improved liver lesion classification". In: *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*. IEEE. 2018, pp. 289–293.

[3] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[4] Yedid Hoshen, Ke Li, and Jitendra Malik. "Non-Adversarial Image Synthesis with Generative Latent Nearest Neighbors". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5811–5819.

[5] Yedid Hoshen and Lior Wolf. "Nam: Non-adversarial unsupervised domain mapping". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 436–451.

[6] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[7] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[8] Alexander Sage et al. "Logo synthesis and manipulation with clustered generative adversarial networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5879–5888.

[9] Jon Shamir. *Frogs dataset*. URL: https://github.com/jonshamir/frog-dataset.

[10] Hoo-Chang Shin et al. "Medical image synthesis for data augmentation and anonymization using generative adversarial networks". In: *International Workshop on Simulation and Synthesis in Medical Imaging*. Springer. 2018, pp. 1–11.

[11] Richard Zhang et al. "The unreasonable effectiveness of deep features as a perceptual metric". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 586–595.