DEGAS: Differentiable Efficient Generator Search

Sivan Doveh and Raja Giryes
Department of Electrical Engineering, Tel Aviv University
Tel Aviv, Israel 69978
{sivandoveh@mail,raja@tauex}.tau.ac.il

Abstract

Network architecture search (NAS) achieves state-of-the-art results in various tasks such as classification and semantic segmentation. Recent techniques have managed to reduce the search for these architectures from several months to just a few days. So far no attempt has been made to search for Generative Adversarial Models (GANs). Since their introduction, GANs have been very popular for image generation. Yet, their training is considered to be a difficult task. The GAN framework is composed of two components: (i) a generator that tries to generate images that look real, and (ii) a discriminator that discriminates between generated and real images. Learning both architectures simultaneously can turn into a very hard problem. In this work, we relax this problem by searching only for a generator architecture. Our architecture search algorithm is inspired by the DARTS approach, and the generator architecture is searched by using the Global Latent Optimization (GLO) procedure. After the generator architecture is selected, we just plug it into an existing framework for GAN training. For CTGAN, which we use in this work, the new model outperforms the original inception score results by 0.25 for CIFAR-10. It also gets better results than RL based search.

1 Introduction

Generative Adversarial Networks (GANs) [3] have become a very successful framework for image generation.

The architectures that are used for the generator in the GAN works are usually inspired by DC-GAN [13] and ResNet [5]. Yet, they are all manually designed.

In recent years, neural architecture search (NAS) based approaches have successfully found models that have outperformed the state-of-the-art in various tasks. While some of these methods require a huge amount of computational resources [15, 19], Efficient NAS [12] and the differentiable architecture search (DARTS) [7] reduced the computational cost significantly to only a few days.

Contribution. In this work, we aim at finding a generator using an efficient neural architecture search. The operations within the generator, which include components such as up-sample operations, are different from the common classification networks. The length of the generator is also much smaller. Thus, we can search all the layers in the network, unlike most of the currently exiting NAS methods (e.g., DARTS) that assume a repeating structure that builds the network and searches only for it. Also, the loss function is very different in GAN training compared to the case of classification.

To overcome the instability in GAN training, we search for the generator model using the GLO training protocol, which lets us search for the generator using a reconstruction loss. This way the generator search is decoupled from the discriminator and thus we do not need to deal with an untrained discriminator that will have a negative influence on the learning of the generator.

At the end of the search, we train the found generator architecture using an existing framework for GAN training, the CTGAN protocol [18]. The search improves the performance in this framework

3rd Workshop on Meta-Learning at NeurIPS 2019, Vancouver, Canada.

both in terms of FID [6] and inception score (IS) [16]. This shows the validity of using the reconstruction loss in the search. Moreover, on all the datasets used in this work, the search took only a couple of days on a single GPU, achieving better results than Wang and Huan [17] that use reinforcement learning. This demonstrates the advantage of the proposed approach for generator search.

2 Background

Generative Adversarial Networks. The common practice shows that training a Generative Adversarial Networks (GANs) [3] is a difficult task due to mode collapse, diminishing gradients and non-convergence to the Nash equilibrium. Much attention has been given recently to find a solution to these difficulties. The Wasserstein GAN (WGAN) [1] use earth mover's distance instead of the common KL or Jensen divergence. WGAN-GP [4] added a regularization that improved the model by enforcing the discriminator function to be differentiable 1-Lipschitz. CAGAN [9] introduced an adversarial consistency loss between different discriminators. CTGAN [18] adds additional regularization over WGAN-GP on the discriminator loss function that penalizes the network if two random perturbations of the inputs have large distance on the output. In this work, we use the CTGAN training framework and its discriminator for testing the generator architecture we have found.

GLO [2] replaced the adversarial training that requires having a discriminator, with another optimization strategy that only trains a generator. Though the image quality produced by GLO is not as good as the one of a generator trained with a discriminator, we use it in our search phase due to its stability.

Architecture search techniques. Several methods have been proposed for optimizing the parameters of neural networks and for finding new architectures. These include reinforcement learning [19, 15] and evolutionary (genetic) [14, 15] based approaches. Yet, these methods require very large computational resources (some needs thousands of GPU days!). Recent works have managed to reduce the high computational cost, to a few GPU days, without reducing the performances [7, 10, 12]. In parallel to our work, a reinforcement learning NAS based approach has been proposed for GANs [17]. Yet, their search time is extremely high as they require using 200 TITAN GPUs for 6 days.

3 GAN search

Our goal here is to efficiently search for a suitable generator architecture. Our assumption, which is demonstrated later empirically, is that a generator that is found based on an image reconstruction criterion can also be used to produce improved images when plugged in an existing GAN procedure. Our generator search method is inspired by DARTS [7]. It learns in an efficient way what operations to use in each layer of the generator from a pre-fixed set of operations that are all used at the beginning.

3.1 DARTS

DARTS [7] contains two phases: In the first one, it searches for the network architecture; and in the second, the new architecture is trained from scratch and then evaluated. DARTS searched network architecture is built from repeating structure of operations, which is called a 'cell'. Thus, it only searches for this structure. Each cell is composed of a feed-forward graph of feature maps that are connected between them by a mixture of operations. This *Mixed Operation* denoted by $\bar{o}^{(i,j)}$ is equal

$$\bar{o}_{i,j}(x) = \frac{\sum_{o \in \mathcal{O}} exp(\alpha_o^{(i,j)}) o(x)}{\sum_{o' \in \mathcal{O}} exp(\alpha_{o'}^{(i,j)})}, \quad (1)$$

where $o(x)$ is an operation on x, $\mathcal{O}$ is the group of all operations in the search space, and $\alpha_o^{i,j}$ is the learned weight for o. The α values in (1) are learned by optimizing the loss function w.r.t their values.

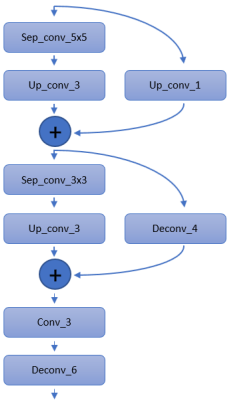

Figure 1: **CIFAR-10 found generator.**

In DARTS, two types of cells are being learned - Normal and Reduction. Normal cell outputs a feature map of the same size as the input. Reduction cell outputs a feature map of a smaller size than the input. The operations $o^{(i,j)}$ can be average/max-pool or types of convolutions with varying kernel

sizes and strides. The kind of operations that are used depend on the cell type. The complete network is composed of a concatenation of several cells.

In the search phase, to make the process faster, the network is smaller than the one that is used in the second phase. At the end of the search, pruning is done to most of the operations, based on their α values. The remaining operations (non-pruned ones) are the final cell structure.

3.2 The generator search

In our search strategy, we do not use the cells method. Since the commonly used generator networks are not as deep as the ones used in classification, we can search for the whole network altogether without using cells. Although many generator architectures are also built from several repeating operations, we exploit the relatively short length of the network and the efficiency of the search algorithm to make the search more flexible in finding new structures.

Search phase. In order to search only for a generator, we adapt the GLO method. This strategy trains a generator alone without the use of a discriminator by optimizing both with respect to the latent space input (z) and the generator weights. For each train image it matches a random noise vector z in the latent space, in order to train the generator to output for each z its corresponding image. Then, a reconstruction loss w.r.t the train image (compared to the generator output) is calculated. The reconstruction loss is a combination of a squared-loss ($\ell_2(x, \grave{x}) = \| x - \grave{x} \|_2^2$ and a Laplacian pyramid ($Lap_1(x, \grave{x}) = \sum_j 2^{2j} \mid L^j(x) - L^j(\grave{x}) \mid_1$, where $L^j(x)$ is the jth level of the pyramid).

Using alternating optimization between the GLO and α values, we perform one step of optimization with respect to the latent space input and generator weights as in GLO, followed by another step with respect to the α values of the MixedOp. The loss function is a combination L_2 and Lap_1 as in GLO.

Searched network structure. The searched network is divided into three parts. The first is fixed and inspired by ResNet and contains a linear operation with reshape. The second contains the architecture and operations that we search for. The last part is also fixed and contains bn+relu+conv+tanh, again, similar to ResNet.

As in DARTS, we use MixedOp to connect between feature maps but with different types of operations in each MixedOp. We use two types: (i) Normal operations that keep the size of their input; and (ii) up-sample operations that increase the size of their input. We also prune connections at the end based on their value, as we explain hereafter.

The normal operations employed are inspired by DARTS and ResNet architecture. They include a combination of bn+Relu+conv with kernel sizes of 1 and 3, Max and Average pooling, skip connection, separate convolutions(see [7]) and dilated convolution with kernel sizes of 3 or 5. The up-sample operations used are inspired by the DCGAN and ResNet architectures and include bn+Relu+deconvolution with kernel sizes of 4 and 6 and bn+Relu+nearest neighbor upsampling + convolution with a kernel size of 1 or 3.

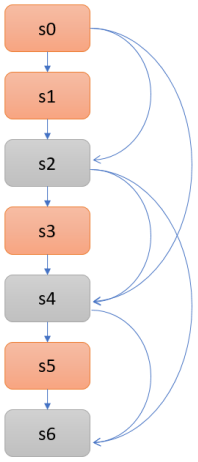

Figure 2: **Network structure before pruning.** Has two operations types: normal (arrows to gray blocks) and up-sample (arrows to orange blocks).

Figure 2 presents the connections used in the search between the different feature maps. We define normal and up-sampled feature maps as the feature maps that are created using normal and up-sample operations respectively. In other words, the input MixedOp to a normal/up-sampled feature map is a normal/up-sample MixedOp. As can be seen in the figure, the up-sampled feature maps are connected between them with residual connections and are connected to the normal feature map before them.

Pruning phase. At the end of the search phase, we perform pruning. The pruning consists of two stages. In the first stage, for each feature map, we keep only one connection to its previous feature maps. The connection is selected to be the operation with the largest value of α. In the second stage, if the connection that was selected in the first stage is residual (which may leave the previous feature map unconnected as the operation skips it), we will also add another operation by selecting from the direct connections the one with the highest α. The rationale behind our pruning strategy is to allow having skip connection in the network structure but without enforcing them.

GAN training. After finding the generator architecture with our search, we can use it to replace the generator in any given GAN framework, e.g., CTGAN [18]. Then, we simply train the new generator with the discriminator of that framework.

4 Experiments

We used CIFAR-10 and STL datasets for our experiments. For having a quantitative evaluation, we measured the FID [6] and Inception score (IS) [16] on CIFAR-10 and STL. A larger version of the figures presented here appears in the supp. material, together with STL and CIFAR-10 generated images using the network stated in the table and the found generators. Additional experiments using supervised training to CIFAR-10 and unsupervised train and search to CelebA are also in the supp. material. For calculating the IS, we employed the same method as in [11] and other works and for calculating the FID we used the same method as in [6].

Table 1: CIFAR-10 unsupervised

Method	IS	FID
Real data	11.24	2.1
SN-GAN [8]	8.22 ± 0.5	11.8
WGAN-GP [4]	7.86 ± 0.07	14.1
CTGAN [18]	8.12 ± 0.12	-
CAGAN [9]	8.35 ± 0.09	-
AGAN [17]	8.29 ± 0.9	30.5
DEGAS(ours)	8.37 ± 0.08	12.01

Table 2: STL unsupervised

Method	IS	FID
Real data	26.08 ± 0.26	3.5
SN-GAN [8]	$9.10 \pm .04$	40.1
WGAN-GP [4]	9.05 ± 0.12	-
CAGAN [9]	9.51 ± 0.14	-
AGAN [17]	$9.23 \pm .08$	52.7
DEGAS (STL net)	9.22 ± 0.08	40.25
DEGAS (CIFAR net)	9.71 ± 0.11	28.76

4.1 Search results and generalization

We searched for a generator for CIFAR-10 and STL using the scheme described above. In all cases, we do not use the labels provided with the data (in CIFAR-10 and STL) in the search. Yet, we show in supplementary material that the generator architecture that is found without labels shows good performance also when the labels are added. We show also the transferability of the model across datasets.

We use the DARTS [7] hyperparameters of CIFAR-10, except for the learning rate that is set to 3e-1 for CIFAR-10, and 3e-2 for STL. On 1 Nvidia TITAN-X GPU, the search took 28 hours for CIFAR-10 and 100 hours for STL. The unsupervised generator training time was 37 hours for CIFAR-10 and 85 hours for STL.

CIFAR-10. Figure 1 presents the network found for CIFAR-10. Notice that different operations are selected in each layer (both for the up-sample and normal operations), which demonstrates the advantage of not forcing the network into the cell structure. Table 1 compares the IS and FID scores of our model to other existing works. Notice that our results are on par with the state of the art methods, better than the RL based search and improves the one of CTGAN, which is our baseline.

STL. For STL, we performed two experiments: (i) searching on STL and then training on it; and (ii) taking the network found on CIFAR-10 and training it on STL. As the image size in CIFAR-10 and STL is different, we have increased the latent vector size by a factor of the image size ratio between CIFAR-10 and STL, in the CIFAR-10 network, which leads to the desired size at the output for STL.

As can be seen in Table 2, the architecture we found on CIFAR-10 transfers well to STL and even outperforms the one searched on STL. We believe that the difference between the results is due to the hyperparameters (initially designed for CIFAR-10) used in the search. We expect that a better hyperparameter selection will improve the search on STL.

4.2 Conclusion

This work introduced DEGAS, a method for searching efficiently generator models without the need to search for a discriminator. On CIFAR-10 and STL, DEGAS outperforms a parallel work to us for automated GAN search [17], both in terms of the search and train cost and terms of inception score. We have also demonstrated the transferability of our found generator both across datasets.The paper's main contribution is that it is the first to provide an efficient generator search strategy. It avoids long search time by using a continuous search space and the GLO framework.

References

[1] Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein gan. In *arXiv preprint arXiv:1701.07875*.

[2] Bojanowski, P.; Joulin, A.; Lopez Paz, D.; and Szlam, A. 2018. Optimizing the latent space of generative networks. In *International Conference on Machine Learning (ICML)*.

[3] Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in neural information processing systems, pages 2672–2680*.

[4] Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. 2017. Improved training of wasserstein gans. In *arXiv preprint arXiv:1704.00028*.

[5] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. In *arXiv:1512.03385*.

[6] Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; and Hochreiter, S. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information*.

[7] Liu, H.; Simonyan, K.; and Yang, Y. 2019. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*.

[8] Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y. 2018. Spectral normalization for generative adversarial networks. In *ICLR*.

[9] Ni, Y.; Song, D.; Zhang, X.; Wu, H.; and Liao, L. 2018. Cagan: Consistent adversarial training enhanced gans. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

[10] Noy, A.; Nayman, N.; Ridnik, T.; Zamir, N.; Doveh, S.; Friedman, I.; Giryes, R.; and Zelnik-Manor, L. 2019. Asap: Architecture search, anneal and prune. In *arXiv:1904.04123*.

[11] Odena, A.; Olah, C.; and Shlens, J. 2016. Conditional image synthesis with auxiliary classifier gans. In *ICML*.

[12] Pham, H.; Y Guan, M.; Zoph, B.; V. Le, Q.; ; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*.

[13] Radford, A.; Metz, L.; and Chintala, S. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. In *arXiv:1511.06434*.

[14] Real, E.; Moore, S.; Selle, A.; Saxena, S.; Leon Suematsu, Y.; Tan, J.; V. Le, Q.; and Kurakin, A. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning (ICML)*.

[15] Real, E.; Aggarwal, A.; Huang, Y.; and V. Le, Q. 2018. Regularized evolution for image classifier architecture search. In *International Conference on Machine Learning - ICML AutoML Workshop*.

[16] Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; and Chen, X. 2016. Improved techniques for training gans. In *Advances in Neural Information Processing Systems (pp. 2234–2242)*.

[17] Wang, H., and Huan, J. 2019. Agan: Towards automated design of generative adversarial networks. In *arXiv:1906.11080*.

[18] Wei, X.; Gong, B.; Liu, Z.; Lu, W.; and Wang, L. 2018. Improving the improved training of wasserstein gans: A consistency term and its dual effect. In *International Conference on Learning Representation(ICLR)*.

[19] Zoph, B., and V. Le, Q. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*.