Generative Adversarial Networks for Data Augmentation

Daniel Appenzeller
Karlsruher Insititut für Technologie (KIT)
Karlsruhe, Germany
ukets@student.kit.edu

Karam Daaboul
Karlsruher Institute of Technology (KIT)
Karlsruhe, Germany
daaboul@kit.edu

***Abstract*—Large amounts of data are often critical for training good performing neural networks. In cases where data is limited it is important to make good use of the available data. This paper provides an overview of multiple techniques for data augmentation that can be used to improve performance in these cases. It presents different architectures using generative adversarial networks and some techniques to stabilize the training of these architectures.**

***Index Terms*—data augmentation, gan, training stabilization**

I. Introduction

In recent years, neural networks have become increasingly popular for many different applications, such as autonomous driving, translation, or image classification. In many of these cases neural networks provide much better results than more traditional methods like the k-means algorithm for image classification [1]. However the performance of a neural network does not only depend on the networks architecture but also to a large degree on the quality and size of the data set used to train it. In the case of autonomous driving, it is relatively easy to generate more data. Simply driving a car with built-in sensors or cameras is all that is required to generate data.

For many other applications like the automatic classification of CT scans in medicine, data generation is not as easy. When a lot of human interaction is required to generate or label the data it can be very time consuming and expensive. In such cases where large data sets are scarce, large amounts of data are still critical to achieve good performance [2]. Here, it is especially important to make efficient use of the available data. A method that can be used for this is data augmentation.

The main goal of data augmentation is to enhance a data set by adding modified copies of existing data to increase the total amount of data available for training a neural network. Many different methods can be used for data augmentation. These can be classified into different groups, which are illustrated in Figure 1. Basic image manipulation methods like flipping or rotating can improve the resulting network. However, these methods are somewhat limited in the amount of data they can generate, and in some cases the generated image might differ from the original in significant ways [3]. Flipping for example can change the position of some organs in a CT scan from the one side of the body to the other.

An alternative to these basic image manipulation methods is the use of a generative adversarial network (GAN) [5]. They

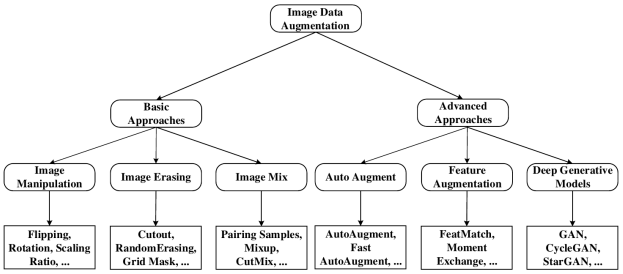


Fig. 1: A taxonomy of image data augmentation methods [4].

are a type of neural network that can be trained to generate new images which can in turn be used to train another neural network. Using GANs for data augmentation significantly increases the amount of data that can be generated for training. Often they can also be combined with simpler methods, like image manipulation to enhance the data set further. This paper provides an overview of multiple architectures enabling the use of GANs for data augmentation. In addition, some necessary techniques are presented to stabilize the training of these architectures.

II. Background

A basic GAN consists of two largely independent neural networks. One is a generator G_θ and the other a discriminator D_ψ. A high level overview of the architecture is illustrated in Figure 2. G receives a noise vector sampled from a distribution $p_z(z)$ as input and generates a fake data point. D receives real and fake data points and outputs a scalar. $D(x)$ represents the probability that x is a real data point instead of a fake one. These two networks are trained simultaneously where D tries to maximise the probability of assigning the correct label to a data point and G tries to minimize $f(-D(G(z)))$ and fool the discriminator into assigning the wrong label. Overall G and D contest each other in a zero sum min-max game with the value function $V(G, D)$ [6]:

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{data}}[f(D(x))] + \mathbb{E}_{z\sim p_{latent}}[f(-D(G(z)))]$$

Here f is a concave function and a common function used for f is $f(x) = -log(1 + exp(-x))$. The goal when training

is to find a Nash equilibrium. In that case both the generator and the discriminator have optimal parameters (θ^*, ψ^*) and neither can improve their performance by changing their parameters slightly. It can be shown that if both the generator and the discriminator are powerful enough to approximate any real-valued function, the unique Nash equilibrium of this game is given by a generator that produces the real data distribution and a discriminator that is 1 everywhere on the data distribution, meaning it only believes that the data inside the distribution is real [7].

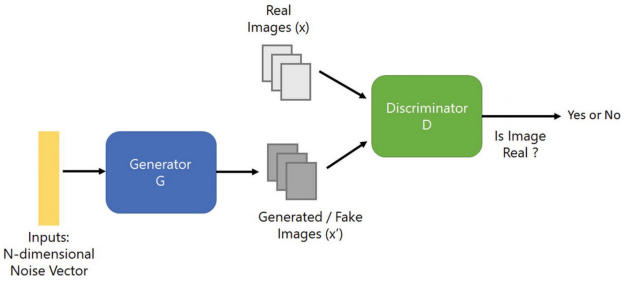


Fig. 2: Architecture of a basic GAN [8].

GANs are usually trained using simultaneous or alternate gradient descent. Both algorithms can be described as fixed point algorithms that apply some operator $F_h(\theta, \psi)$ to the parameter values of the generator and discriminator [9]. This is repeated until a fixed point is reached where $F_h(\theta, \psi) = (\theta, \psi)$. Simultaneous gradient descent corresponds to the operator $F_h(\theta, \psi) = (\theta, \psi) + hv(\theta, \psi)$ where $v(\theta, \psi)$ denotes the following gradient vector field [7]:

$$v(\theta, \psi) := \begin{pmatrix} -\Delta_\theta L(\theta, \psi) \\ \Delta_\psi L(\theta, \psi) \end{pmatrix}$$

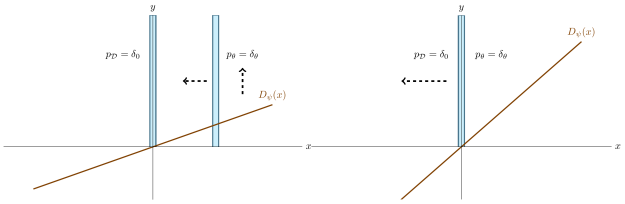
Similarly, alternating gradient descent can be described by an operator $F_h = F_{2,h} \circ F_{1,h}$ where $F_{1,h}$ and $F_{2,h}$ perform an update for the generator and the discriminator, respectively [7].

III. Training

Training GANs can be difficult even in a basic case. For more complex architectures such as the GenClass framework (Section IV-C) or GTNs (Section IV-D), these problems become magnified. In both cases, different techniques are used to make training of these architectures possible. For GenClass a gradient penalty is used to ensure convergence [7]. GTNs, on the other hand, use weight normalization for stabilization [10].

A. Convergence

The convergence of GAN training near an equilibrium point can be analyzed using the Jacobian at the equilibrium $F_h^{'}(\theta^*, \psi^*)$. If the Jacobian has eigenvalues greater than 1, the training algorithm will not converge, if they are smaller than 1 it will converge. Similarly, simultaneous gradient descent only converges if all Jacobian eigenvalues of the vector field $v(\theta, \psi))$ have a negative real part [9].



(a) $t = t_0$ (b) $t = t_1$

Fig. 3: Dirac-GAN illustration [7]

Even for a very simple GAN, there is no guarantee of convergence. This can be proved using a Dirac-GAN as a counterexample, visualized in Figure 3. For the Dirac-GAN both the generator and the discriminator have exactly one parameter. The training objective is given by

$$L(\theta, \psi) = f(\psi\theta) + f(0)$$

In the beginning, the discriminator pushes the generator toward the true data distribution and the slope of the discriminator increases (Figure 3a). When the generator reaches the target distribution, the slope of the discriminator is largest, pushing the generator away from the target distribution. This causes the system to oscillate and never converge (Figure 3b) [7]. The problem is that there is no incentive for the discriminator to move to the equilibrium discriminator, even if the generator reproduces the true data distribution exactly. The discriminator always pushes the generator away from the equilibrium which makes training unstable near the equilibrium point.

If the data distribution is absolutely continuous, then this is not the case and the GAN actually converges to an equilibrium point. However, this assumption may not be satisfied for many applications of GANs, such as natural images. In cases where the distribution is absolutely continuous but concentrated along a lower-dimensional manifold, the Jacobian eigenvalues are very close to the imaginary axis and the convergence is very slow [9].

There are some techniques that can be used to improve the behaviour of GANs and ensure convergence.

1) Instance Noise: One such technique is instance noise, which attempts to fix the convergence problem from a slightly different perspective. Instead of seeking the Nash equilibrium, a GAN can be viewed as an iterative algorithm where the discriminator's job is the estimation of the likelihood ratio between the true data distribution and the distribution generated by the generator [11]. The discriminator is first trained by logistic regression between the generative model q_θ and the true data p. Then a logarithmic likelihood ration $s(y) = \log(\frac{q_\theta(y)}{p(y)})$ is estimated from the discriminator. Finally the parameters of the generator θ are updated via stochastic gradient descent with the objective function $\mathbb{E}_{y \sim q_\theta}\, s(y)$.

In theory this algorithm should converge however there are several assumptions made that do not always hold in practice. The log likelihood ratio is finite, and there is a single optimal discriminator. Both of these assumptions are not true if the distributions q_θ and p do not overlap. This is

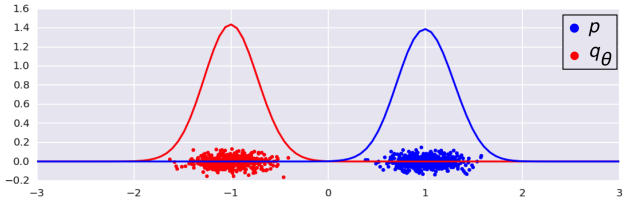


Fig. 4: Standard, non-noisy distributions [12]

the case particularly early during training where q_θ may be degenerate and, for example, with natural images p may be concentrated on a lower-dimensional manifold. When these distributions do not overlap log likelihood ratio is infinite and there are multiple optimal or near optimal discriminators. This is illustrated in Figure 4 where both distributions can be separated by multiple different discriminators. For a fixed q_θ and p training could lead to multiple different near optimal discriminators depending on the initialisation, all of which would provide different gradients to the generator.

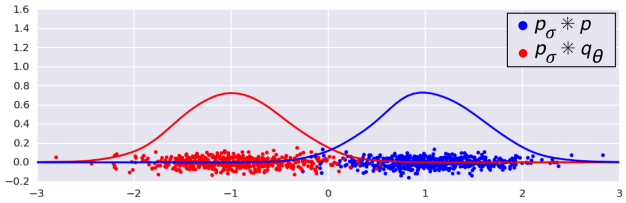


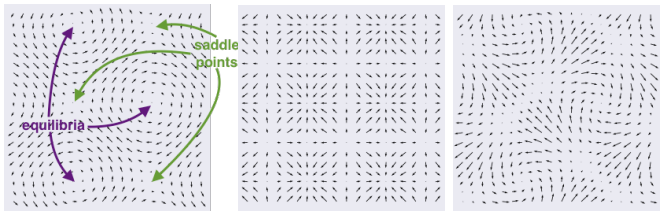
Fig. 5: Distributions with added instance noise [12]

Instance noise attempts to solve this problem by making the discriminator's job harder. This is done by adding random noise p_σ to the samples from both q_θ and p. As a result, these noisy distributions now overlap, which means that the log-likelihood ratio is no longer infinite and that there is only one optimal discriminator. These noisy distributions are illustrated in Figure 5.

In practice, this seems to work as expected and stabilizes GAN training. Initially there is some critical amount of noise that is required to ensure that the GAN converges. As the generator improves during training and gets closer to the true data distribution the amount of noise added to the samples can be decreased.

2) Consensus Optimization: Another approach that attempts to fix the problems of simultaneous gradient descent is the use of a consensus optimization. The issue with simultaneous gradient descent, compared to normal gradient descent, is that the vector field may be non-conservative. One example of such a non conservative vector field is $v(\theta, \psi) = (\cos\psi, \sin\theta)$ which is illustrated in Figure 6a. This vector field has multiple equilibrium points, however close to each point the gradients are circular. Because of this gradient descent would never find the equilibrium and only move in a circle.

To solve this problem it is possible to first construct a conservative vector field $-\Delta L(x) := -\frac{\delta}{\delta x}|v(x)|_2^2$ illustrated in Figure 6b, which is conservative because it is defined as the gradient of a scalar function L [9]. This new vector field $-\Delta L$



(a) Non-conservative field v (b) Conservative field $-\Delta L$ (c) Combined field $v - 0.6\Delta L$

Fig. 6: Vector field illustrations [13]

has the same fix points as v and gradient descent converges to a local minimum, which is also a fix point of v. The issue with this is that there is no way to tell whether the fix point to which we converge is an equilibrium in v or just a saddle point.

This can be solved simply by taking the linear combination of the original vector field v and the constructed conservative field $-\Delta L$. The result is a combined field $v - \gamma\Delta L$ illustrated in Figure 6c. This combined field is still non-conservative but is better behaved than the original vector field. How well behaved the vector field is can again be quantified by looking at the eigenvalues of the Jacobian of the vector field. These eigenvalues can be controlled using γ and increasing γ makes the vector field more conservative. Setting γ too large however introduces spurious equilibrium points again which are only saddle points of v. In practice, it is important to find a good middle ground for γ so that gradient descent converges, but does so to an actual equilibrium point.

B. Stabilization

1) Weight Normalization: A technique that can help stabilize neural network training in general is weight normalization [10]. In a standard artificial neural network the output of a neuron is computed by taking the weighted sum of its inputs $y = \phi(w * x + b)$, where w is a weight vector b is a bias, x is an input vector, ϕ is an element wise nonlinearity and y is the output of the neuron. Such a network is trained by stochastic gradient descent on the parameters w and b.

It is possible to reparameterize the weight vector w as $w = \frac{g}{||v||}v$ where v is vector, g is a scalar, and $||v||$ is the Euclidean norm of v. This separates the direction of the weight vector w from its norm. Weight normalization can help make the network more robust to different learning rates and speed up convergence.

IV. Architectures

When using a GAN for data augmentation to improve classifier performance, the GAN can simply be used to increase the amount of data available for training. There are also multiple different GAN architectures designed specifically for data augmentation, some of which directly integrate both the GAN and the classifier in the same architecture

A. CycleGAN

A popular GAN architecture for image-to-image translation is a CycleGAN [14]. It is able to translate images from one domain to another, like, for example, translating horses to zebras and vice versa. This has many different applications such as photo-enhancement or style transfer but can also be used for data augmentation in certain situations.

An example of this is the use of a CycleGAN to transform contrast CT scans into non-contrast scans [15]. Convolutional neural networks can be used for the segmentation of CT scans to identify different organs. They can achieve high performance in these tasks, but like most neural networks, this requires large amounts of training data. The problem with training a network in this case is that most data available for training consists of contrast CT scans, whereas in the real world non-contrast scans are also very common. Using a network trained on contrast CT scans to segment non-contrast CT scans results in very poor performance.

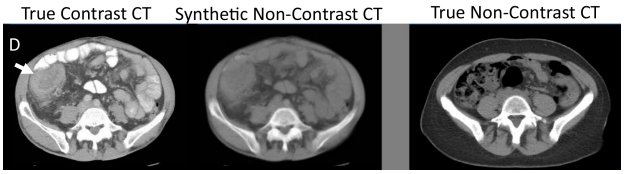


Fig. 7: CT scan image translation [15]

To improve performance a CycleGAN can be used to transform the contrast CT scans from the larger data sets like Figure 7 and augment the non contrast data set. Compared to standard augmentation methods like simple image manipulation, the use of a CycleGAN improves segmentation performance for non contrast CT scans. For certain organs, like Figure 8 the performance improvements are very significant. Here, a network trained on data augmented by a CycleGAN achieves a result that is fairly close to that of an expert, whereas a network trained with standard augmentation methods fails completely.

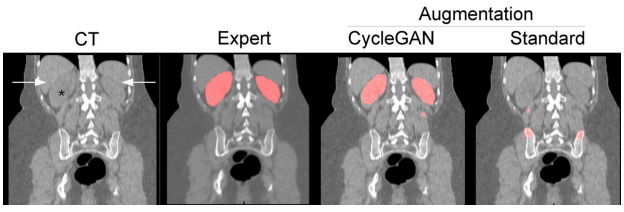


Fig. 8: CT scan segmentation [15]

Standard data augmentation methods can also be used in combination with a CycleGAN to further improve segmentation performance.

B. Data Augmentation Generative Adversarial Network

The approach used by Data Augmentation Generative Adversarial Networks (DAGAN) is also to directly augment the data-set used to train a normal classifier [16]. The architecture, shown in Figure 9, uses a form of transfer learning to achieve this. Transfer learning is based on the assumption that even if two class conditional distributions have no overlap they still share some commonalities, which means information can be transferred between domains.

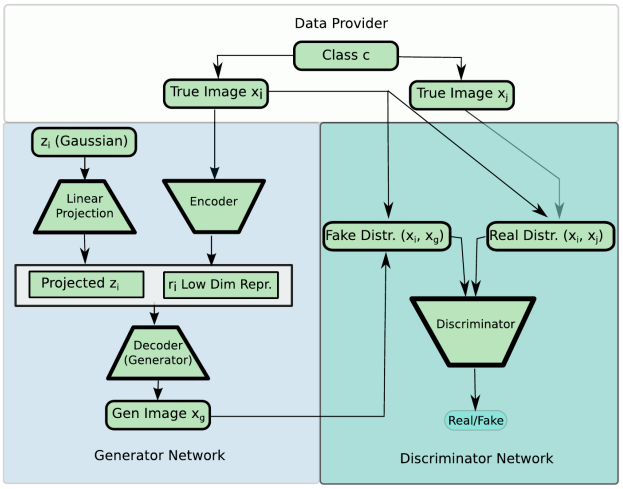


Fig. 9: DAGAN Architecture [16].

In this case we have a source domain where a lot of data is available and a low-data target domain. The goal is now to train a conditional GAN in the source domain and use it to generate data in the target domain. This approach gives it an inherent advantage over standard augmentation methods which only use the data from the target domain without the additional information that can be transferred from the source domain.

This is done by training the generator to generate images from the same class as an input image it receives. The input image is first encoded into a lower-dimensional representation and then combined with a randomly sampled vector. This combined vector is then decoded to produce the output image.

The discriminator then receives distributions counting either two real images or one real and one fake image and tries to distinguish between fake and real distributions. It is important for the discriminator to receive distributions containing the original image as well, not just individual output images, to prevent the generator from simply auto-encoding its input. A distribution ensures that the generator has to produce images that are related to the original image but still different. Both the generator and the discriminator do not receive class information, so their behavior generalizes over all classes.

After the generator is trained in the source domain, it can be used to create more images in the target domain. These images can then be used to augment the training data and improve the performance of the classifier. This seems to work fairly well even if the source and target domains are very different. In Figure 10 only the top left image is real, the other images are generated using a DAGAN network.

Typically a DAGAN improves classifier performance more than traditional data augmentation methods. It is however also

Fig. 10: Image sample augmented with DAGAN [16].

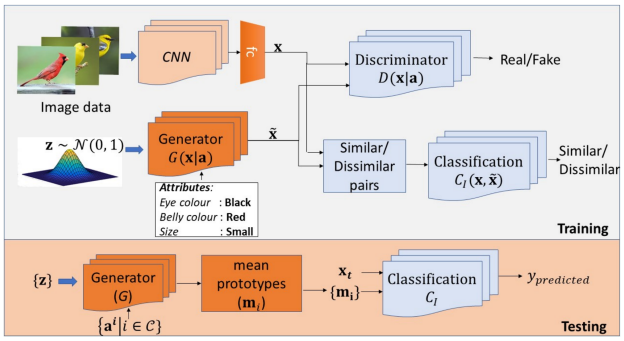


Fig. 11: GenClass Architecture [17]

possible to combine the generator with other augmentation methods and improve performance further.

C. Unified Generator-Classifier

The GenClass architecture (Figure 11) does not use a standard classifier [17]. Instead, it integrates the generator and the classifier into a single framework. The goal of this approach is to improve the performance of the classifier for zero-shot learning, which means that the classifier does not need to be updated to properly identify images from a new class.

The generator in this case is similar to the one used in the DAGAN architecture, but instead of an encoded image, it takes an attribute vector that describes a given class. This attribute vector is assumed to be available even for unseen classes. It describes a given class and is used by the generator along with a random vector to generate new images from the same class.

The classifier in this architecture does not try to directly assign a class to an image. Instead it tries to calculate the similarity between two images. If they are similar, they are more likely to be from the same class.

This whole network is then trained in an end to end manner using a wasserstein loss and a gradient penalty for stabilization [7], [18]. The overall loss function of the framework is

$$L_{GC} = L_{WGAN} + L_{C_I}$$

where L_{WGAN} is the Wasserstein GAN loss combined with a gradient penalty for stabilization and L_{C_I} is the loss function of the classifier [7], [17], [18]. This means that the entire network is trained in an end-to-end manner, which helps the generator to learn to generate more relevant features that help reduce the classifiers loss as well.

During the testing of a new class, the generator creates some samples of the new class using the classes attribute vector. Then from these generated samples a mean is calculated that represents the new class and is then used for classification. The classifier compares a given image with all the calculated mean images and then chooses the class that is most similar to it. The resulting accuracy of this approach for zero-shot learning is in many cases higher than other techniques on the same data-set.

D. Generative Teaching Network

In the case of Generative Teaching Networks (GTNs), the classifier (learner in Figure 12) is also directly integrated into the network architecture but completely replaces the discriminator normally used to train GANs [19]. Instead of using adversarial training, GTNs train the generator and the learner cooperatively using meta training. This works by first training the learner in the inner loop until convergence using only generated data. Then the learner is tested on real data. The loss of the learner on real data is then used to train the generator, this way it learns to generate data that helps the learner minimize this loss.

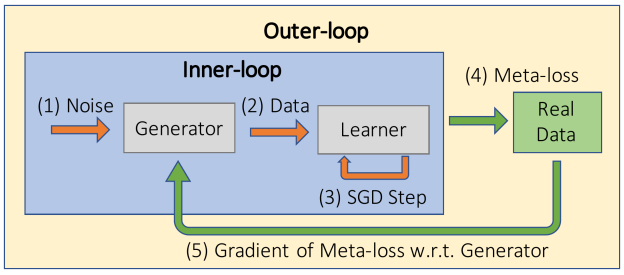


Fig. 12: Generative Teaching Network Architecture [19]

Typically, the performance of a learner trained only by the generator is slightly worse than that of the same learner trained on real data. The advantage of using a generator instead is that it can train the learner much more quickly. A real dataset might be very large with many similar data points. If the data points are too similar, they do not significantly progress the learner's training. Because the generator produces data specifically to help the learner train, it does not have the same problem. It is able to compress the original data and generate new data points that are not necessarily realistic, but make the learner progress faster. Figure 13 is an example of handwritten numbers sampled from a GTN trained on the MNIST dataset. Although they are still somewhat recognizable as numbers, they are not very visually distinct. Despite this a classifier trained on these samples reaches good performance much faster than one trained on real data.

An application of this is neural architecture search (NAS), another area of machine learning. Here various different learners are trained to find the best performing one for a specific application. The problem is that the training time for large data sets can become very long, which reduces the number

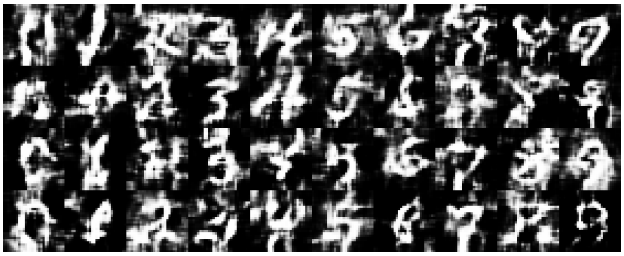


Fig. 13: Handwritten numbers sampled from a GTN [19]

of learners that can be tested. Using a GTN speeds up the training process, meaning more learners can be tested, which in turn makes it more likely to find one that is well suited to the application. GTNs generally outperform other architecture search methods, such as weight sharing, but can also be used in combination with these methods [19].

Another potential application of GTNs is generating training environments for reinforcement learning agents. For a simple task like CartPole training the GTN takes about as long as training a single learner with policy gradients. After the GTN is trained however it can train a learner to achieve maximum performance in a single gradient descent step [19]. This is unlikely to be the case for more complex problems, but these results are still promising for future applications of GTNs.

V. Conclusion

This paper presents several methods to stabilize GAN training and multiple GAN architectures specifically for data augmentation. Overall data augmentation is a very useful tool that can help improve performance and prevent overfitting when little data is available [20]. As neural networks become more popular and are applied to more niche problems, good data augmentation will also likely become increasingly important.

GANs seem to be a valid option for data augmentation and already perform better than traditional methods in many cases. As the quality of GAN samples improves, this advantage will likely only increase, making GANs an integral part of data augmentation. In addition to improved performance, GANs are also much more versatile and can be used for things like meta training or image translation which are not really possible with simpler data augmentation methods.

One problem GANs and data augmentation in general have little influence on are some more severe biases in a data set. For example, in a classification task where the classifier attempts to distinguish between different items of clothing, if the training data only contains images of shirts, the classifier will be biased. There is however no way for data augmentation to create images of pants from this data set and it will not improve the bias. Other biases like lighting or backgrounds can often be significantly reduced specifically using GAN based data augmentation.

References

[1] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," Nov. 2015.

[2] J. Cho, K. Lee, E. Shin, G. Choy, and S. Do, "How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?" Nov. 2015.

[3] S. Motamed, P. Rogalla, and F. Khalvati, "Data augmentation using generative adversarial networks (gans) for gan-based detection of pneumonia and covid-19 in chest x-ray images," Jun. 2020.

[4] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen, "Image data augmentation for deep learning: A survey," Apr. 2022.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," Jun. 2014.

[6] V. Nagarajan and J. Z. Kolter, "Gradient descent gan optimization is locally stable," Jun. 2017.

[7] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for gans do actually converge?" *International Conference on Machine Learning 2018*, Jan. 2018.

[8] M. Nayak, "Deep convolutional generative adversarial networks(dcgans)," 2018, accessed 2022-07-12. [Online]. Available: https://medium.datadriveninvestor.com/deep-convolutional-generative-adversarial-networks-dcgans-3176238b5a3d

[9] L. Mescheder, S. Nowozin, and A. Geiger, "The numerics of gans," May 2017.

[10] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," Feb. 2016.

[11] M. Uehara, I. Sato, M. Suzuki, K. Nakayama, and Y. Matsuo, "Generative adversarial nets from a density ratio estimation perspective," Oct. 2016.

[12] F. Huszár, "Instance noise: A trick for stabilising gan training," 2016, accessed 2022-08-21. [Online]. Available: https://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/

[13] ——, "Gans are broken in more than one way: The numerics of gans," 2017, accessed 2022-07-12. [Online]. Available: https://www.inference.vc/my-notes-on-the-numerics-of-gans/

[14] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," Mar. 2017.

[15] V. Sandfort, K. Yan, P. J. Pickhardt, and R. M. Summers, "Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks," *Scientific Reports*, vol. 9, no. 1, nov 2019.

[16] A. Antoniou, A. Storkey, and H. Edwards, "Data augmentation generative adversarial networks," Nov. 2017.

[17] A. K. Pambala, T. Dutta, and S. Biswas, "Unified generator-classifier for efficient zero-shot learning," May 2019.

[18] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," Jan. 2017.

[19] F. P. Such, A. Rawal, J. Lehman, K. O. Stanley, and J. Clune, "Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data," Dec. 2019.

[20] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, jul 2019.