THIS DATASET DOES NOT EXIST: TRAINING MODELS FROM GENERATED IMAGES

Victor Besnier[* 1,2], *Himalaya Jain*[1], *Andrei Bursuc*[1], *Matthieu Cord*[1,2], *and, Patrick Pérez*[1]

[1]Valeo.ai, Paris, France
[2]Sorbonne University, Paris, France

ABSTRACT

Current generative networks are increasingly proficient in generating high-resolution realistic images. These generative networks, especially the conditional ones, can potentially become a great tool for providing new image datasets. This naturally brings the question: *Can we train a classifier only on the generated data?* This potential availability of nearly unlimited amounts of training data challenges standard practices for training machine learning models, which have been crafted across the years for limited and fixed size datasets. In this work we investigate this question and its related challenges. We identify ways to improve significantly the performance over naive training on randomly generated images with regular heuristics. We propose three standalone techniques that can be applied at different stages of the pipeline, *i.e.*, data generation, training on generated data, and deploying on real data. We evaluate our proposed approaches on a subset of the ImageNet dataset and show encouraging results compared to classifiers trained on real images.

Index Terms— Image classification, generative networks

1. INTRODUCTION

In recent years, generative networks have made great progress. The state-of-the-art generative adversarial networks (GANs) can generate diverse high-resolution images almost indistinguishable from real images. Among others one of the main objectives of learning generative models is to get an unbounded supply of data for training machine learning models, in particular deep neural networks. This could significantly reduce expenses and work required for data acquisition or could be helpful when the original data cannot be shared, *e.g.* due to privacy. Moreover, storing a generator network is often lighter than storing the entire dataset, for example BigGAN [1] takes 225MB while ImageNet [2] takes ∼150GB.

Using GANs to generate abundant and diverse training data has been a longstanding motivation for research on GANs. However until recently, the obvious low quality of the generated images (*e.g.* low resolution, many artefacts) discouraged any attempts of training classifiers with such images only. Researchers focused mostly on improving the

---

*Victor Besnier worked on this project as an intern at Valeo.ai, Paris.

Generated images from BigGAN

Randomly sampled   With proposed methods

Fig. 1: BigGAN can potentially generate diverse and photorealistic images. However, with random sampling, the diversity is much lower than a real image dataset of the same size. The proposed methods increase diversity and effective number of samples, leading to better performance for a classifier trained on generated images.

photorealism and diversity of the images thanks to the newly introduced metrics to measure these attributes, namely Inception Score (IS) [3] and Frechet Inception Distance (FID) [4]. Some tried using synthetic images for augmenting the existing set of real images to boost up performance [5, 6, 7].

The current state-of-the-art for conditional image generator is BigGAN [1]. BigGAN has received lot of attention as it can generate high resolution, photorealistic, diverse images for all 1,000 semantic classes of ImageNet. Motivated by the outstanding IS and FID scores of BigGAN, Ravuri and Vinyals [8] train a classifier over images generated from BigGAN and test it on the validation set of ImageNet. They find that there is a significant drop in accuracy, 27.9%, compared to the classifier trained on the real images from training set of ImageNet[1], concluding that IS and FID are not suitable metrics for this task.

In this work, we address this challenging question of drop in performance when deep classifiers are trained on synthetic generated data and tested on real images. We estimate that the drop is probably due to the fact that the generated images, when sampled randomly, are less diverse than those in the real dataset. However, since we have the generator we should be no longer limited in the amount of training samples we can have nor confined to randomly sampling from the generator

---

[1]We note that the training set of ImageNet is very large, ∼1,300 images per class, making it very challenging for a classifier trained on the same amount of generated data to outperform one trained on ImageNet.

(Fig. 1). To overcome this drop in performance, we propose in this paper three learning strategies: (1) a method to generate more informative data from the generator via latent code optimization; (2) a pragmatic strategy to use continuous sampling from a generator while training and hence, to increase diversity of the data; (3) a test time adaptation method to improve accuracy. Our three contributions may be combined and we will evaluate all the gains with BigGAN trained on ImagNet.

2. RELATED WORK

Improvement of GANs. GANs are notoriously unstable and difficult to train, requiring several specific architectural choices and hyperparameter tuning. Several works have improved stability of GAN training via optimization [9, 10] or architectural means [11]. BigGAN [1] is a pinnacle of both scientific and engineering expertise accumulated across years of dealing with GANs. BigGAN is a class-conditional GAN whose training benefits from large batch sizes, high resolution images, and skip-connections for the latent code in the generator. The quality of the generated images is improved through sampling from Gaussian noise with truncation. Given the quality of BigGAN samples, we consider it to study training over synthetic images.[2]

Latent code optimization. Exploring a GAN's latent space through iterative optimization has been proposed to generate more memorable images [12], different geometric and photometric transformations of a given image [13] or faces with specific attributes [14]. The focus of these works is to generate realistic images. However this does not ensure that a classifier trained on such images would do well. Our latent code optimization aims to improve the performance of the image classification network and we use it for mining helpful codes for learning.

Training over generated data. In the quest for more relevant metrics for evaluating GANs, recent studies started analyzing the performance of classifiers trained on synthetic images [15, 16, 8]. Shmelkov *et al.* [15] show that all popular GANs consistently under-perform by a significant margin against classifiers trained on real images. Ravuri and Vinyals [8, 16] show that even for highly photorealistic images, *e.g.* from BigGAN [1], synthetic classifiers lag behind real ones. Minor gains ($+0.2\%$) can be obtained by mixing real and synthetic images in specific settings. In this work, we show that nontrivial distribution information learned by recent GAN architectures can be mined and leveraged towards competitive classifiers trained with supervision from synthetic data only.

---

[2]Given the cost of training BigGAN (24-48 hours on a Google TPU v3 Pod with 128 core for the smallest model with 128×128 images), we do not train it ourselves and use the pretrained model provided by the authors.

3. APPROACH

The aim of this work is to propose methods to use off-the-shelf pre-trained *latent variable generative models* for training deep networks. Specifically we consider the set-up where, given only a pre-trained class-conditional GAN for image generation, one wants to train an image classifier with only images generated by this GAN.

Now to describe our approaches, we first introduce the notation. Let G be a pre-trained class-conditional generative network, which maps a latent code z given a class label y to an image x *i.e.*, $x = G(z, y)$. The generator is assumed of sufficient quality for the image x to be semantically classified as y by a human. We are interested in generating a dataset $\mathcal{X}^{gen}$ of images with labels which can be then used to train a classifier C_θ with parameters θ. A classifier takes an image as input and gives scores for each class,

$$s_x = C_\theta(x) \in \mathbb{R}^K, \tag{1}$$

where K is number of classes. We train the classifier by iteratively minimizing the cross-entropy loss over the training set. The loss on one image is defined as $-\log(s_x[k])$, where k is the true class of x.

Note that when training the classifier on generated images, we do not update the generator G. As noticed in [16], we also observe that if we simply sample the dataset $\mathcal{X}^{gen}$ from G to train C_θ, there is a significant drop in accuracy compared to training C_θ on a real dataset $\mathcal{X}^{real}$. Reducing this performance gap is our aim. To this end, we propose three distinct and independent methods, as described below.

3.1. Hard Sample Mining (HSM) with latent code optimization

Hard mining is a popular method in metric learning. The key intuition is to use harder or more informative samples from the training data to train the model. Such hard samples are interesting because they lead to increased robustness and better accuracy.

Now when training from generated images, we can use the generator to produce more informative images rather than drawing mere samples. The more informative or harder images are specific to the current state of the classifier. Let $C_\theta^{(i)}$ be the classifier at an iteration i during the training. The hard images for $C_\theta^{(i)}$ would be the images which are difficult to classify correctly by $C_\theta^{(i)}$, *i.e.*, with low classification score for the true class. For generating such an image for a given class y, we propose to optimize iteratively an original, random latent code z to minimize the score of the class predicted by $C_\theta^{(i)}$ (not of the true class, as will be discussed later) for new image $G(z_{\text{new}}, y)$ *i.e.*,

$$k^* = \arg\max C_\theta^{(i)}(G(z, y)), \tag{2}$$

Fig. 2: **Overview of the proposed HSM and DS methods for training.** In HSM, we minimize the maximum of logit over z. In DS, at every epoch we replace some part of the data with new samples. Note that the methods work at different stages of the full pipeline and can be applied independently.

$$z_{j+1} = z_j - \eta \nabla_{z_j} s_{z_j, y}[k^*], \qquad (3)$$

where $z_0 = z$ and, $s_{z_j, y}$ stands for $C_\theta^{(i)}(G(z_j, y))$, hence $s_{z_j, y}[k^*]$ is the score of class k^* with current classifier applied to image $G(z_j, y)$ (Fig. 2). Here all the parameters of G and $C_\theta^{(i)}$ are fixed and only the input of G, *i.e.*, the latent code, is optimized by back-propagating through G and $C_\theta^{(i)}$. The optimized latent code will lead to a difficult image for $C_\theta^{(i)}$. However this image is not necessarily a good or valid sample. Since we optimize only for sample difficulty, the latent code can change too strongly across iterative updates and it may no longer follow the Gaussian prior expected by the generator. We temper this drift and preserve the prior across updates by scaling z_j to have the same ℓ_2-norm as z_0: $z_j \leftarrow z_j \frac{\|z_0\|}{\|z_j\|}$.

Note that in the objective above we do not minimize the score of the correct class y, and we minimize instead the score of the highest confidence class (regardless of the true label). If the prediction is different from the true class, the image is already difficult and thus our objective will lead to an easier sample. This avoids generating overly hard samples, unlikely to be learned by the classifier.

Collecting a dataset with HSM. Since the difficulty of a sample is specific the performance of the model at a given iteration, in order to build a relevant and diverse dataset we should aggregate such samples from various intermediate stages of training. To this end, we propose a simple algorithm to use HSM for collecting a fixed size dataset $\mathcal{X}^{gen}$ of N samples within K semantic classes as given in Alg. 1. The underlying idea is to initialize the dataset with $M < N$ samples, train a classifier on this data, then add to the dataset M new samples collected with HSM for the classifier. We keep alternating between training the classifier and collecting M new samples with HSM until the dataset $\mathcal{X}^{gen}$ reaches N items. In Fig. 3, we provide a few qualitative examples to illustrate the impact of HSM over the generated images. Once the gathering of the entire set $\mathcal{X}^{gen}$ is complete, we train a new, final classifier on it.

Algorithm 1 Collecting fixed size dataset with HSM

1: Input: G, C, N, K ▷ N: dataset size, K: #classes
2: Output: $\mathcal{X}^{gen}$
3: **function** FILLDATASET(M, WITHHSM, C)
4:     $\mathcal{X} \leftarrow \{\}$
5:     **for** $k = 1 \cdots K$ **do**
6:        **for** $m = 1 \cdots M/K$ **do**
7:           $z \leftarrow \mathcal{N}(0, 1)$
8:           **if** WITHHSM **then**
9:              $z \leftarrow$ HSM(z, k, G, C) ▷ mine hard code for sample z for class k
10:           $x \leftarrow G(z, k)$
11:           $\mathcal{X}$.APPEND(x)
12: $i \leftarrow 0$
13: $C_i \leftarrow C$
14: $\mathcal{X}^{gen} \leftarrow$ FILLDATASET(M, FALSE, NONE)   ▷ $M < N$
15: **for** $i = 1 \cdots N/M$ **do**
16:     $C_i \leftarrow$ TRAIN($C_{i-1}, \mathcal{X}^{gen}$)   ▷ train C_{i-1} for an epoch
17:     $\mathcal{X}^{hsm} \leftarrow$ FILLDATASET(M, TRUE, C_i)
18:     $\mathcal{X}^{gen}$.APPEND($\mathcal{X}^{hsm}$)
19: RETURN $\mathcal{X}^{gen}$

Algorithm 2 Classifier training with DS and HSM

1: Input: G, C, K, r, N ▷ N: #samples per epoch
2: Output: C
3: $i \leftarrow 0$
4: $C_i \leftarrow C$
5: $\mathcal{X}_i^{gen} \leftarrow$ FILLDATASET(N, FALSE, NONE)
6: **for** $i = 1 \cdots I$ **do**
7:     $C_i \leftarrow$ TRAIN($C_{i-1}, \mathcal{X}_i^{gen}$)
8:     $\mathcal{X}^{hsm} \leftarrow$ FILLDATASET($rN/2$, TRUE, C_i)
9:     $\mathcal{X}^{new} \leftarrow$ FILLDATASET($rN/2$, FALSE, NONE)
10:     $\mathcal{X}_i^{gen} \leftarrow$ SUBSET($\mathcal{X}_i^{gen}$) of size $(1-r)N$
11:     $\mathcal{X}_{i+1}^{gen} \leftarrow \mathcal{X}_i^{gen} \cup \mathcal{X}^{hsm} \cup \mathcal{X}^{new}$
12: RETURN C_I

Fig. 3: **Effect of applying HSM at different iterations during classifier training.** The first image of each category is sampled randomly. The other two images are generated from HSM-computed codes at two different steps during training. The difference between the images shows that the effect of HSM is specific to the classifier.

3.2. Dataset Smoothing (DS)

As previously discussed, the models trained on generated data $\mathcal{X}^{gen}$ have lower performance compared to $\mathcal{X}^{real}$ when using sets of equal sizes, *i.e.*, $|\mathcal{X}^{gen}| = |\mathcal{X}^{real}|$. This is caused by the lower visual diversity of the generated data in comparison to real data. We can afford to forego the dataset size constraint and augment the diversity through additional examples, since we have a generator that can practically generate unlimited training data. Yet storing such a large dataset is not practical either. So, it is more pragmatic to generate the images online while training. On the downside, generating data perpetually will increase the training time significantly due to the non-negligible cost of generating images. Moreover, training on this continuously changing data causes training instability, particularly in the first epochs. This is probably due to differences in statistics of the generated data across training batches.

To mitigate these drawbacks, we propose our approach of *dataset smoothing*. The key idea is to progressively change the dataset by only partially replacing the generated training data with new samples every epoch, aiming for a diverse but gradually changing dataset (Fig. 2). In detail, we start with a generated dataset $\mathcal{X}^{gen}$, train the classifier on it. Then, we replace a fraction r of $\mathcal{X}^{gen}$ with new randomly sampled images. This can be combined with HSM by applying it on some of the new images. The approach with HSM is summarized in Alg. 2. With the smooth changing of the dataset, we effectively have the diversity as in continuous sampling, while preserving a more stable and much faster training.

3.3. Batch Norm statistics Adaptation (BNA)

Batch Normalization (BN) [17] is a prominent architectural innovation in deep neural networks widely used for robustifying and accelerating training. BN stabilizes the distribution of hidden activations over a mini-batch during training by transforming the output of a layer to have zero mean and unit variance. BN layers first set the mean and variances of activations, and subsequently scale and shift them via a set of

trainable parameters to maintain expressivity.

Formally, consider a mini-batch $\mathcal{B}$ of size m and the respective activations $h_i^{(k)} \in \mathbb{R}, i = \{1 \dots m\}$ of a neuron k for each sample in this mini-batch. We omit k for clarity. BN transforms h_i as follows:

$$\hat{h}_i = \gamma \frac{h_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}} + \beta, \qquad (4)$$

where $\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} h_i$ and $\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^{m} (h_i - \mu_{\mathcal{B}})^2$ are the mean and variance of mini-batch $\mathcal{B}$ for the considered neuron, $\varepsilon > 0$ is a small constant , and $(\gamma, \beta) \subset \theta$ are learnable parameters.

During training, BN normalizes each mini-batch using its respective statistics $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ computed on the fly. At the same time BN estimates the mean and variance of the activations from the whole training set, denoted by $\bar{\mu}$ and $\bar{\sigma}^2$, through exponential running averages with sub-unitary update factor α. Formally, at training iteration t, the running mean and variances are given by:

$$\bar{\mu}^t = \alpha \mu_{\mathcal{B}}^{t-1} + (1 - \alpha)\bar{\mu}^{t-1} \qquad (5)$$
$$(\bar{\sigma}^2)^t = \alpha(\sigma_{\mathcal{B}}^2)^{t-1} + (1 - \alpha)(\bar{\sigma}^2)^{t-1}.$$

We denote the set of estimated statistics for all BN layers in our classifier network as $\omega = \{\bar{\mu}, \bar{\sigma}^2\}$. A neural network with BN layers is parameterized by two types of parameters: θ, learned by backpropagation and gradient descent, and ω, computed from feature activation statistics. We denote our classifier using BN as $C_{\theta,\omega}$.

During testing, BN uses the estimated mean and variances for normalizing input activations from test samples, enforcing distributions of activations similar with training. BN assumes that both train and test images are sampled from similar distributions and the expectations of the activation values from the two sets are close to each other. Thus BN can be sensitive to differences between train and test data, which occur for instance in multi-domain training [18], domain adaptation [19]. Albeit BigGAN generates photorealistic samples, as we train exclusively on synthetic images and test on real images, the classifier $C_{\theta,\omega}$ might be sensitive to this small domain gap.

In order to mitigate this, we leverage BN layers for a simple unsupervised domain adaptation of $C_{\theta,\omega}$ from synthetic to real images. We adapt the estimated BN statistics for real images through a forward pass on unlabeled images from the training set, without backpropagation, as the running averages for $\bar{\mu}$ and $\bar{\sigma}^2$ are updated on the fly. In detail we compute ω^{real} from real images with Eq. 5, while keeping the parameters θ intact. We refer to this method as BatchNorm statistics Adaptation (BNA).

4. EXPERIMENTS

In this section we evaluate the three proposed approaches to learn with generated data. We first outline the datasets we use

	Fixed dataset				Continuous sampling				Long training	real images
HSM	-	✓	-	✓	-	✓	-	✓	-	
DS	-	-	-	-	✓	✓	✓	✓	✓	
BNA	-	-	✓	✓	-	-	✓	✓	✓	
Top-1 Accuracy	70.6	73.6	76.4	78.2	78.6	81.8	84.2	**85.6**	**88.8**	88.4

Table 1: **Results for ImageNet-10 real test images.** Performance of classifiers trained on generated images with all combinations of the proposed methods. Each classifier is trained for 150 epochs (except *Long training*, where we let DS run for 1500 epochs) over a set of $N = 13$K images; in case of *continuous sampling* we replace 50% (*i.e.*, $6,500$) of the images every epoch, while *fixed dataset* is the usual setup where no images are replaced during training. In all setups we use N images per epoch. First column, without applying any of the proposed methods, is the baseline. Each of the proposed methods individually shows improvement over the baseline. The combination of the methods further improves the results.

and describe the evaluation method. Then, we give the details of experimental setup followed by the experimental results and their analysis.

Dataset. As we train only on generated images, we use the real dataset just for testing and not for training. Since we use BigGAN as generator, which is trained to generate ImageNet-like images [2], we use ImageNet for evaluation. ImageNet is a large dataset of 1.3M labeled images, with 1K semantic classes. As the dataset is very large, we select 10 classes to evaluate our approaches with reasonable computation time and resources. We refer this subset of 10 classes as ImageNet-10. We select the classes of ImageNet-10 to mimick the ones from CIFAR-10 dataset [20][3]. The 10 classes are: *aircraft carrier*, *balloon*, *fire truck*, *gazelle*, *goldfish*, *labrador*, *red fox*, *redshank*, *race car*, *tabby cat*. We use as test set, images from the validation set of ImageNet for the selected 10 classes, with 50 images per class. Finally, we use real images from the train set of ImageNet-10 to train a classifier on real images to compare against.

Experimental setup. We use as evaluation metric the Top-1 accuracy on real images, *i.e.*, the percent of test images classified correctly. For our experiments, we use as generator a pre-trained BigGAN [1] that produces 128×128 images.[4] For all the experiments we use ResNet-18 as classifier. We use SGD optimizer with $lr = 0.1$, momentum, learning rate decay and weight decay $1e - 4$.

Results. We first assess the impact of our contributions individually and in combination (Table 1). We use as baseline a classifier trained on random $\mathcal{X}^{gen}$ without any of the proposed approaches. HSM brings gains from 1.4% to 3.2% whatever the combination. DS improves the accuracy by 7.4% to 8.2%. With BNA, which is just a post-training adaptation on real data, scores get a boost of 3.8% to 5.8%. Interestingly, all methods are complementary and any com-

Fig. 4: **Effect of replacement fraction r in DS.** Classification accuracy using DS on real images with varying r, *i.e.*, fraction of the dataset being replaced with new images every epoch. The figure shows plots for DS with and without BNA.

bination improves scores. Overall, by combining all three proposed methods, we outperform the baseline by 15%. By increasing the number of training iterations 10 times, we reach 88.8% accuracy.

Ablation studies. We study the impact of DS with continuous sampling, *i.e.*, dynamically generating new samples every epoch. We use as baseline a classifier trained over a fixed size dataset $\mathcal{X}^{gen}$ of N generated images, with $|\mathcal{X}^{gen}| = |\mathcal{X}^{real}|$. For DS, we train another classifier for which at every epoch we sample $r \times N$ new images and replace a part of the generated dataset with them. At each epoch we have a constant number of images N. In Fig. 4, we show the impact of the replacement fraction on classification accuracy. The fraction r is linearly related to the amount of samples generated and time it requires during training. Thus, it is compelling to have lower r. In general, increasing r improves accuracy, especially between *no replacement* and $r = 0.1$. The improvement diminishes quickly and almost saturates after $r = 0.5$ at the expense of longer training time. In the same setup, we can see that BNA systematically improves accuracy with up to $+5.8\%$. Interestingly, using BNA over DS with $r = 0.5$ has slightly better accuracy than replacing all ($r = 1$) which moreover takes longer to train.

---

[3]CIFAR-10 classes are broad categories, while ImageNet ones are fine-grain. Thus we select representative classes from ImageNet manually, *e.g.* *labrador* for *dog*. In place of CIFAR-10 classes with no equivalent (*frog*, *horse*) we use *red fox* and *balloon* reported as top and respectively worst performing classes in the experiments from [8].

[4]https://github.com/huggingface/pytorch-pretrained-BigGAN

5. CONCLUSIONS

In this work we address the task of training a classifier using solely generated data. We show that for practically unlimited amounts of data, several standard practices in training models could be revisited. We propose three standalone contributions to this effect and show their benefits both when used individually or combined. The limitations of this study are related to the reduced number of considered classes due to computation constraints and due to high variability in the quality of BigGAN images across classes. Given the fast pace of progress in GAN literature, steady improvements in the quality of the generated images are foreseeable. We expect that the same questions addressed here will still arise when training over generated data from future GANs and our proposed approaches are readily applicable in such context.

6. REFERENCES

[1] Andrew Brock, Jeff Donahue, and Karen Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[3] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, 2016, pp. 2234–2242.

[4] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017, pp. 6626–6637.

[5] Toan Tran, Trung Pham, Gustavo Carneiro, Lyle Palmer, and Ian Reid, "A bayesian data augmentation approach for learning deep models," in *Advances in neural information processing systems*, 2017, pp. 2797–2806.

[6] Swee Kiat Lim, Yi Loo, Ngoc-Trung Tran, Ngai-Man Cheung, Gemma Roig, and Yuval Elovici, "Doping: Generative data augmentation for unsupervised anomaly detection with gan," in *2018 IEEE International Conference on Data Mining*. IEEE, 2018, pp. 1122–1127.

[7] Giovanni Mariani, Florian Scheidegger, Roxana Istrate, Costas Bekas, and Cristiano Malossi, "Bagan: Data augmentation with balancing gan," *arXiv preprint arXiv:1803.09655*, 2018.

[8] Suman Ravuri and Oriol Vinyals, "Seeing is not necessarily believing: Limitations of biggans for data augmentation," 2019.

[9] Martin Arjovsky, Soumith Chintala, and Léon Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[10] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville, "Improved training of wasserstein GANs," *Advances in Neural Information Processing Systems*, pp. 5768–5778, 2017.

[11] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[12] Lore Goetschalckx, Alex Andonian, Aude Oliva, and Phillip Isola, "Ganalyze: Toward visual definitions of cognitive image properties," *arXiv preprint arXiv:1906.10112*, 2019.

[13] Ali Jahanian, Lucy Chai, and Phillip Isola, "On the"steerability" of generative adversarial networks," *arXiv preprint arXiv:1907.07171*, 2019.

[14] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou, "Interpreting the latent space of gans for semantic face editing," *arXiv preprint arXiv:1907.10786*, 2019.

[15] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari, "How good is my gan?," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 213–229.

[16] Suman Ravuri and Oriol Vinyals, "Classification accuracy score for conditional generative models," *arXiv preprint arXiv:1905.10887*, 2019.

[17] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[18] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi, "Learning multiple visual domains with residual adapters," in *Advances in Neural Information Processing Systems*, 2017, pp. 506–516.

[19] Woong-Gi Chang, Tackgeun You, Seonguk Seo, Suha Kwak, and Bohyung Han, "Domain-specific batch normalization for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7354–7362.

[20] Alex Krizhevsky, Geoffrey Hinton, et al., "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.