Diversification, Compression, and Evaluation Methods for Generative Adversarial Networks

Gonçalo Filipe Torcato Mordido

Publikationsbasierte Universitätsdissertation
zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)

am Fachgebiet Internet-Technologien und -Systeme
des Hasso-Plattner-Institut

eingereicht an der
Digital-Engineering-Fakultät
der Universität Potsdam

Dezember, 2021

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-53546
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-535460

Declaration

I herewith declare that I have produced this thesis without the prohibited assistance of third parties and without making use of aids other than those specified. Notions taken over directly or indirectly from other sources have been identified as such. All data and findings in the work have not been falsified or embellished. This thesis has not previously been presented in identical or similar form to any other German or foreign examination board.

The thesis work was conducted from July 2017 to June 2021 under the supervision of Prof. Dr. Christoph Meinel.

Gonçalo Filipe Torcato Mordido
Potsdam, Germany

Abstract

Generative adversarial networks (GANs) have been broadly applied to a wide range of application domains since their proposal. In this thesis, we propose several methods that aim to tackle different existing problems in GANs. Particularly, even though GANs are generally able to generate high-quality samples, the diversity of the generated set is often sub-optimal. Moreover, the common increase of the number of models in the original GANs framework, as well as their architectural sizes, introduces additional costs. Additionally, even though challenging, the proper evaluation of a generated set is an important direction to ultimately improve the generation process in GANs.

We start by introducing two diversification methods that extend the original GANs framework to multiple adversaries to stimulate sample diversity in a generated set. Then, we introduce a new post-training compression method based on Monte Carlo methods and importance sampling to quantize and prune the weights and activations of pre-trained neural networks without any additional training. The previous method may be used to reduce the memory and computational costs introduced by increasing the number of models in the original GANs framework. Moreover, we use a similar procedure to quantize and prune gradients during training, which also reduces the communication costs between different workers in a distributed training setting.

We introduce several topology-based evaluation methods to assess data generation in different settings, namely image generation and language generation. Our methods retrieve both single-valued and double-valued metrics, which, given a real set, may be used to broadly assess a generated set or separately evaluate sample quality and sample diversity, respectively. Moreover, two of our metrics use locality-sensitive hashing to accurately assess the generated sets of highly compressed GANs. The analysis of the compression effects in GANs paves the way for their efficient employment in real-world applications.

Given their general applicability, the methods proposed in this thesis may be extended beyond the context of GANs. Hence, they may be generally applied to enhance existing neural networks and, in particular, generative frameworks.

Zusammenfassung

Generative adversarial networks (GANs) wurden seit ihrer Einführung in einer Vielzahl von Anwendungsbereichen eingesetzt. In dieser Dissertation schlagen wir einige Verfahren vor, die darauf abzielen, verschiedene bestehende Probleme von GANs zu lösen. Insbesondere, fokussieren wir uns auf das Problem das GANs zwar qualitative hochwertige Samples generieren können, aber die Diversität ist oft sub-optimal. Darüber hinaus, stellt die allgemein übliche Zunahme der Anzahl der Modelle unter dem ursprünglichen GAN-Framework, als auch deren Modellgröße weitere Aufwendungskosten dar. Abschließend, ist die richtige Evaluierung einer generierten Menge, wenn auch herausfordernd, eine wichtige Forschungsrichtung, um letztendlich den Generierungsprozess von GANs zu verbessern.

Wir beginnen mit der Einführung von zwei Diversifizierungsmethoden die das ursprüngliche GAN-Framework um mehrere Gegenspieler erweitern, um die Diversität zu erhöhen. Um den zusätzlichen Speicher- und Rechenaufwand zu reduzieren, führen wir dann eine neue Kompressionsmethode ein. Diese Methode basiert auf den Monte-Carlo-Methoden und Importance Sampling, für das Quantisieren und Pruning der Gewichte und Aktivierungen von schon trainierten neuronalen Netzwerken ohne zusätzliches Trainieren. Wir erweitern die erwähne Methode zusätzlich für das Quantisieren und Pruning von Gradienten während des Trainierens, was die Kommunikationskosten zwischen verschiedenen sogenannten „Workern“ in einer verteilten Trainingsumgebung reduziert.

Bezüglich der Bewertung der generierten Samples, stellen wir mehrere typologie basierte Evaluationsmethoden vor, die sich auf Bild-und Text konzentrieren. Um verschiedene Anwendungsfälle zu erfassen, liefern unsere vorgestellten Methoden einwertige und doppelwertige Metriken. Diese können einerseits dazu genutzt werden, generierte Samples, oder die Qualität und Verteilung der Samples anhand einer Menge von echten Samples zu bewerten. Außerdem, verwenden zwei unserer vorgestellten Metriken so genanntes locality-sensitive Hashing, um die generierten Samples von stark komprimierten GANs genau zu bewerten. Die

Analyse von Kompressionseffekten in GANs ebnet den Weg für ihren effizienten Einsatz für reale Anwendungen.

Aufgrund der allgemeinen Anwendungsmöglichkeit von GANs, können die in dieser Arbeit vorgestellten Methoden auch über Kontext von GANs hinaus erweitert werden. Daher könnten sie allgemein auf existierende neuronale Netzwerke angewandt werden und insbesondere auf generative Frameworks.

Acknowledgments

Because family always comes first, I would first like to thank my amazing mother, Cristina, for teaching and showcasing to me the importance of being a genuinely good person at heart every day. To my inspirational sister, Andreia, I am forever grateful for the support and guidance that ultimately led me to pursue my research path. I would also like to thank my assertive dad, António, that gave me the right tools to build the work ethic that I have today. Because life is only worth living with the right people by your side, I am grateful to have found my lovely girlfriend, Taryn, with whom I can not wait to share my life experiences. I am grateful for my friends, which are gladly too many to reference and are spread across the world. I am excited they gave me the opportunity to follow and share their life paths.

Professionally, I would like to thank Prof. Dr. Christoph Meinel for giving me the opportunity to pursue my Ph.D. studies. My doctoral experience would not be the same without the team at the Machine Learning group. Namely, I thank PD Dr. Haojin Yang for his guidance throughout my stay in his group and Mina, Christian, Joseph, Ting, and Xiaoyin for the good times. I would like to especially thank Mina for providing me a different outlook on life. Like so, I would like to thank Pejman, which was always there for me whenever I needed it despite his own struggles, and Ihsan for constantly reminding me how simple life can be. Last but not least, I would like to make a special thanks to Dr. Alexander Keller for challenging me and making me grow as a person and researcher. His research practices and life outlook will remain with me throughout my life and career.

Contents

1 Introduction

This Chapter provides an overview of the research presented throughout this thesis. Namely, Section 1.1 describes the different research contexts in which the work presented in this thesis was conducted. Section 1.2 introduces the broad motivations for the overall research. Section 1.3 presents the thesis outline. Finally, Section 1.4 provides a summary of the main research contributions in this thesis. For readers unfamiliar with the fundamentals of deep learning, we refer to Goodfellow et al. [GBC16].

1.1 Research Background

The presented research was performed with a practical and interdisciplinary nature in mind and was conducted in several independent collaborations with two industry partners: SAP and NVIDIA. More specifically, SAP provided funding for the entirety of my Ph.D. studies. On the other hand, I also spent a total of 10 months of that time at NVIDIA in the form of two research internships: a 6-month internship at an early stage of my studies and a 4-month internship approaching the end. This Section provides a broader picture of the contexts from which the different research topics presented in this thesis originated.

Throughout most of the duration of my Ph.D. studies, my research was conducted with practical views in mind that may potentially benefit several application scenarios within different teams at SAP. During the first project period, I worked closely with the Conversational AI at SAP located in Palo Alto, CA, USA. The research goal was to enhance generative enterprise systems. To this end, I proposed to stimulate sample diversity in generative adversarial networks or GANs. Two refereed papers came out of this collaboration:

- *Dropout-GAN: Learning from a Dynamic Ensemble of Discriminators*, presented at KDD 2018 Deep Learning Day [MYM18]. Joint work with Haojin Yang and Christoph Meinel.
- *microbatchGAN: Stimulating Diversity with Multi-Adversarial Discrimina-*

tion, presented at WACV 2020 [MYM20]. Joint work with Haojin Yang and Christoph Meinel.

For the next project iteration, I worked with the Conversational AI team located in Paris, France. This time, the goal was to focus on the evaluation of generative enterprise systems. To this end, I proposed several metrics applied to different data domains. Such metrics resulted in two refereed papers:

- *Assessing Image and Text Generation with Topological Analysis and Fuzzy Logic*, presented at WACV 2021 [MNM21]. Joint work with Julian Niedermeier and Christoph Meinel.
- *Mark-Evaluate: Assessing Language Generation using Population Estimation Methods*, presented at COLING 2020 [MM20]. Joint work with Christoph Meinel.

In the most recent project iteration, I worked with the SAP Innovation Center Network team located in Newport Beach, CA, USA. The general goal of this iteration was to compress such generative enterprise systems. I performed a thorough evaluation of compressed generative adversarial networks and developed additional evaluation metrics optimized to assess the compression effects in the generated sets. These efforts resulted in one paper that is currently under review at the time of this writing:

- *Evaluating Post-Training Compression in GANs using Locality-Sensitive Hashing*, under review [MYM21]. Joint work with Haojin Yang and Christoph Meinel.

Switching gears, in my time at NVIDIA, I worked on making neural networks more efficient by applying different compression techniques. Two refereed papers came out of this collaboration:

- *Instant Quantization of Neural Networks using Monte Carlo Methods*, presented at NeurIPS 2019 workshop on Energy Efficient Machine Learning and Cognitive Computing [MVK19]. Joint work with Matthijs van Keirsbilck and Alexander Keller.
- *Monte Carlo Gradient Quantization*, presented at CVPR 2020 workshop on Efficient Deep Learning in Computer Vision [MVK20]. Joint work with Matthijs van Keirsbilck and Alexander Keller.

The aforementioned works are also part of a granted patent [Kel+19a] and were further presented at NVIDIA's 2019 GPU Technology Conference [Kel+19b]. In this thesis, I further extend and discuss these works in the context of generative adversarial networks.

As briefly mentioned, I ended up interning again at NVIDIA by the end of my Ph.D. studies where I received the NVIDIA Recognition Award for my research contributions. The conducted research was presented at NVIDIA's 2021 GPU Technology Conference [MVK21a] and led to a paper [MVK21b] to be presented at INTERSPEECH 2021 as well as a filed patent [KMV21]. However, this research falls outside the scope of this thesis and it is, therefore, not presented.

Moreover, I was also a co-author in several refereed papers during my Ph.D. studies which have shown the successful extension of some of my research contributions in additional tasks and data domains:

- *Pseudo-Ground-Truth for Adversarial Text Generation using Reinforcement Learning*, presented at NeurIPS 2018 workshop on Deep Reinforcement Learning. First authored by Jonathan Sauder and joint work with Xiaoyin Che, Haojin Yang, and Christoph Meinel.
- *Best Student Forcing: A Simple Training Mechanism in Adversarial Language Generation*, presented at LREC 2020 [Sau+20]. Co-First authored by Jonathan Sauder and Ting Hu and joint work with Xiaoyin Che, Haojin Yang, and Christoph Meinel.

As suggested in this Section, the work presented in this thesis originated from several collaboration efforts. Therefore, the pronoun "we" instead of "I" is used in this document from here on.

1.2 Research Motivation

Generative adversarial networks (GANs) [Goo+14] were originally proposed as a two-model framework consisting of one generator (G) and one discriminator (D) that are trained together. More specifically, while D learns to distinguish real and fake samples, G attempts to generate realistic samples to fool D. Although being a widely used framework offering promising results across various domains [DMP19; ES16; HE16; Yan+18; Yu+17], GANs have also been inherently associated with instability in training.

One common problem is known as mode collapse [AB17; ACB17; Che+17; Kim+17; MNG17], observed when G's generated set exhibits high sample quality but low sample diversity when compared to the real set. Hence, G may successfully fool D by only generating samples from the same data mode, *i.e.* connected components of the data manifold, leading to the generation of similar fake samples. This suggests that G did not succeed in learning the real data distribution, as originally intended. As an example of mode collapse, considering a use case where the real data consists of images of several animals, G would only be capable of generating images of dogs or cats at the end of training. Our first research question is then: *How may we mitigate mode collapse?*

Although extending the original GANs framework to multiple generators [Gho+18; Hoa+18], discriminators [DGM17; Ngu+17], or both [CF18; Gan+17; GE18] have been proposed, we rely solely on increasing the number of discriminators to mitigate mode collapse. Augmenting the number of discriminators instead of generators offers benefits at inference time since the discriminator(s) are usually discarded after G's training. More specifically, we propose two novel multi-discriminator frameworks: Dropout-GAN [MYM18], which applies adversarial dropout by omitting the feedback of a given D at the end of each minibatch, and microbatchGAN [MYM20], which assigns a different portion of each minibatch (called microbatch) to each D to stimulate sample diversity. The increase in the number of models in the framework leads to yet another question: *How may we reduce the complexities of the different models in these multi-adversarial frameworks?*

The memory and computational cost of neural networks may be reduced by pruning redundant weights or neurons [HMD16; LDS90; Moc+18] as well as applying quantization to lower the precision of weights [CBD15a; LZL16; Zhu+17] as well as activations [Hub+16; Ras+16]. Using low-precision computations on top of pruning allows for efficient hardware implementations [Lin+16; VNM17]. One problem of popular compression methods, however, is the need to re-train or fine-tune the compressed network, introducing additional financial and environmental cost [SGM19]. To this end, we propose Monte Carlo Quantization (MCQ) which uses Monte Carlo methods and importance sampling to prune and quantize both weights and activations of pre-trained neural networks without any additional training. This leads to another question: *What about compression during training?*

Compression techniques may also be applied in the backward pass. Therefore,

pruning and quantization methods may also be applied to gradients during training [CBD15b; Gup+15; Lou+19; Wu+18; Zho+16]. The main benefits of gradient compression schemes come into play in distributed training settings [Ali+17; Ber+18; Kar+19; SCJ18; Sei+14; Wen+17], since the communication cost between the different workers may be reduced while improving the computational efficiency of each worker. Since common approaches mostly focus on either gradient quantization or gradient pruning separately, we propose to apply a similar procedure as MCQ to gradients – Monte Carlo Gradient Quantization (MCGQ) – combining pruning and quantization in one efficient sampling algorithm.

In the context of GANs, MCQ may be used to reduce the memory and computational costs of G post-training, without the need for any re-training. This is important when, for example, the computational resources are scarce or time is of the essence. Moreover, re-training a pre-trained model may also be problematic in cases where the accessibility to training data and settings is limited. On the other hand, MCGQ may be used during training to reduce both the complexity and communication costs of our multi-adversarial frameworks if training the different discriminators in a distributed setting scenario.

One important point that we have not yet introduced is the task of evaluating a generated set. The appropriate assessment of a generated set is of extreme importance to identify possible shortcomings in a model's generation process. When learning a discriminative, supervised task, evaluation is often straightforward by comparing a model's predictions against ground-truth labels. However, with generative, unsupervised tasks, the assessment of a model's capabilities is far more challenging. A relevant question is then: *How can we evaluate a generated set?*

Depending on the use case, single-valued metrics [Bińkowski+18; Heu+17; Sal+16] may suffice to assess specific conditional generation tasks, such as machine translation and text summarization, where we are interested in evaluating the similarities of a generated translation or summary to a specific reference translation or summary, respectively. On the other hand, if we consider unconditional generation, for example, it may be useful to have separate measures for the quality and diversity of the generated set [Kyn+19; Saj+18]. Such double-valued metrics (usually in the form of precision and recall) may enable the identification of possible shortcomings of a given generation system, such as the mode collapse problem in GANs. We propose several evaluation metrics in this thesis that cover different drawbacks of existing methods. Namely, we propose Fuzzy Topology

Impact (FTI), a finer-grained image evaluation approach that leverages fuzzy logic, and Mark-Evaluate (ME), a family of three language evaluation metrics based on population estimation methods that achieves a high correlation to human evaluation on challenging text generation tasks. To tie some ends, our final research question is: *How does compression affect the generated set?*

High compression levels are likely to distort the generation process. This is likely to lead to an increase of outliers which may negatively affect the overall assessment of a generated set, especially if using popular k-nearest neighbor (KNN) approaches [Kyn+19]. Hence, on top of assessing several popular GANs designs trained on several high-quality datasets, we propose two new metrics that use locality-sensitive hashing (LSH) to increase outlier robustness as well as reduce the computational complexity of the overall evaluation process. Our study suggests that GANs may be compressed to low bit-width, ultimately leveraging specific application scenarios where sample quality is potentially more critical than sample diversity.

Figure 1.1 highlights the different methods proposed throughout this thesis. As discussed in this Section, all methods are inherently connected and complement one another. For example, the proposed compression methods may be used to reduce the complexity added by the proposed generative multi-adversarial networks, with the compression effects possibly being evaluated using our LSH-based evaluation metrics. Alternatively, we may also directly assess the proposed generative multi-adversarial networks, as well as original GANs, using the other proposed evaluation metrics, depending on the task and data domain. Furthermore, our compression methods may also be used to possibly balance out our generative multi-adversarial networks in terms of improving the sample quality and sample diversity trade-off.

1.3 Thesis Outline

In this Chapter, we started establishing the research background in which the different work presented in this thesis was conducted. Then, we highlighted some of the research questions that fueled the discussed work. Furthermore, we provided a glance at the proposed methods as well as research topics that will be presented throughout this thesis. We will now proceed to outline the rest of the document and summarize the main discussions presented in each chapter. A summary of our main contributions is also presented at the end.

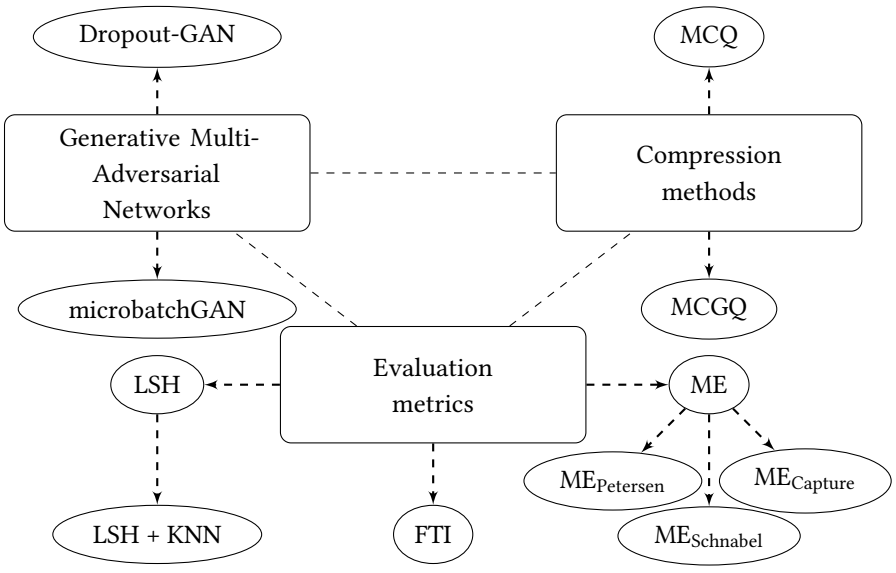


Figure 1.1: Thesis contributions. The discussed topics, represented as rounded rectangles, are inter-connected and complementary. The proposed methods for each topic are represented as ellipses.

In Chapter 2, we start by presenting our contributions to multi-discriminator GANs training. More specifically, we discuss our two novel frameworks, *Dropout-GAN* and *microbatchGAN*, that make use of an ensemble of discriminators to stimulate sample diversity in the generated set produced by G. Popular training modifications to stabilize and improve learning in GANs are also discussed.

In Chapter 3, we present our novel compression methods that make use of Monte Carlo methods and importance sampling. Specifically, *Monte Carlo Quantization* is used to quantize and prune weights and activations in a post-training setting. On the other hand, *Monte Carlo Gradient Quantization* extends the previous approach to compress gradients during training in a distributed training setting. Related compression methods for weights, activations, and gradients are also introduced.

In Chapter 4, we discuss our novel evaluation metrics to assess a generated set. Namely, one of the metrics assesses how a generated set and a real set differ by measuring their *Fuzzy Topology Impact*, using Fuzzy Logic and topological representations. Our family of metrics, *Mark-Evaluate*, assesses an evaluation

set by using population estimation methods that are popular in ecology. A discussion of widely used evaluation metrics is provided.

In Chapter 5, we build on our previously described efforts to assess the compression effects in the generated set of GANs. To this end, we propose two novel outlier-aware evaluation metrics based on locality-sensitive hashing. Even though GANs compression is a relatively new, albeit important, research topic, we analyze recent work and discuss future directions.

Finally, in Chapter 6, we highlight our contributions and present our final remarks.

1.4 Main Contributions

We summarize the main contributions presented in this thesis as follows:

- We present two novel and general multi-adversarial frameworks for GANs training, *i.e.* Dropout-GAN and microbatchGAN, and empirically show that they help mitigate the inherent mode collapse problem in GANs by promoting sample diversity in G during training.

- We propose Monte Carlo Quantization (MCQ) to quantize and prune the weights and activations of pre-trained neural networks using Monte Carlo methods and importance sampling techniques. The resulting compressed networks achieve close to full-precision accuracy without any additional training. Importantly, the complexity of the resulting networks is proportional to the number of samples taken.

- We extend the previous method by leveraging both pruning and quantization to compress gradients of neural networks during training. On top of reducing the communication exchanged between multiple workers in a distributed setting, we also improve the computational efficiency of each worker. Our method, called Monte Carlo Gradient Quantization (MCGQ), shows faster convergence and higher performance than existing compression methods on image classification and language modeling tasks.

- To evaluate the quality and diversity of a generated image set, we propose Fuzzy Topology Impact. Our metric retrieves two interpretable metrics,

which directly correlate to sample quality and sample diversity. Contrarily to previous topology-based methods, our method enhances evaluation by allowing for a finer-grained assessment due to the usage of fuzzy logic.

- Focusing on text evaluation, we present Mark-Evaluate: a family of 3 novel language metrics that empirically show sensitivity to mode collapse and quality detriment. Moreover, we achieve a high correlation to human evaluation on challenging text generation tasks, such as unconditional language generation, machine translation, and text summarization.
- We reduce the computational complexity of assessing an evaluation sample against a set of reference samples of existing KNN-based metrics by using locality-sensitive hashing. This is an important step since increasing the number of reference samples is likely to lead to a better assessment of existing evaluation methods. Our LSH-based metrics are also sensitive to outliers in both the generated and real sets that may otherwise negatively impact the proper evaluation using current KNN-based metrics.
- Finally, we show that existing compression methods may be successfully applied to pre-trained GANs, providing a trade-off between precision and recall: while sample quality is mostly retained, sample diversity is majorly affected by high compression levels. Hence, depending on the use case, compression may act as a simple, yet effective measure to balance existing GANs in a post-training setting, improving the precision and recall trade-off on several popular GANs.

2 Generative Multi-Adversarial Networks

This Chapter is based on two of our refereed papers: Dropout-GAN [MYM18] and microbatchGAN [MYM20]. We note that some of the results presented for Dropout-GAN are based on an extended arXiv version of the paper presented at KDD 2018 Deep Learning Day. Dropout-GAN has also been applied to the language generation domain in our subsequent work presented at LREC [Sau+20]. For readers unfamiliar with GANs, we refer to Goodfellow et al. [Goo+14].

In this Chapter, we introduce our novel generative multi-adversarial networks that aim at stimulating sample diversity in the generated set of G when training GANs. Hence, the proposed frameworks aim at mitigating the inherent mode collapse problem in GANs. Specifically, we provide an overview of the original GANs framework as well as existing modifications focused on increasing sample diversity in Section 2.1. We then present our contributions in the form of two new multi-adversarial frameworks – Dropout-GAN and microbatchGAN – in Sections 2.2 and 2.3, respectively. Finally, we present some concluding thoughts in Section 2.4.

2.1 Related Work

We will start by introducing GANs in Section 2.1.1 as well as discussing existing methods to stimulate sample diversity in the generated sets in Section 2.1.2.

2.1.1 Generative Adversarial Networks (GANs)

The original GANs framework, introduced by Goodfellow et al. [Goo+14], consists of two models: a generator (G), that tries to capture the real data distribution to generate fake samples that look realistic, and a discriminator (D), that tries to do a better job at distinguishing real and fake samples. G maps a latent space to the data space by receiving noise as input and applying transformations to it to generate fake samples. On the other hand, D maps a given sample to a probability p, representing the likelihood of such sample coming from the real

data distribution. Ideally, with enough iterations, G starts generating such realistic samples that D is no longer able to distinguish from real samples. However, due to the training instability in GANs previously mentioned in Chapter 1, this ideal state of equilibrium is hard to reach in practice. More concretely, GANs introduce the following minimax game between G and D:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))], \quad (2.1)$$

with p_r being the real data distribution from which a real sample x may be sampled and $D(x)$ representing D's output. On the other hand, p_z is the noise distribution used to sample G's input z, while $G(z)$ represents its output.

D maximizes (2.1) by improving its classification task on correctly distinguishing between real and generated samples by assigning $p \approx 1$ to real samples and $p \approx 0$ to generated samples. On the other hand, G minimizes (2.1) by improving its ability to fool D, *i.e.* leading D to assign $p \approx 1$ to its generated samples.

In practice, however, $\log(1 - D(G(z)))$ is known to introduce vanishing gradients, which hinder training and are originated from D's ability to easily distinguishing real and fake samples due to the lack of realistic samples generated by G. To prevent this, the original paper [Goo+14] introduces a simple modification by maximizing $\log(D(G(z)))$ instead. Note that this modification does not change the tasks originally designed for each model. Nevertheless, to study the change in training behaviors when introducing a multi-adversarial setting, we experiment with both the original and modified variants in Section 2.2.

2.1.2 Stimulation of Sample Diversity in GANs

As discussed in Chapter 1, mode collapse is a common problem in GANs [AB17; ACB17; Che+17; Kim+17; MNG17], where G mostly produces similar samples that are able to fool D. In the end, this leads to a poor generation process since only samples from certain data modes are likely to be produced. A description of the initially proposed efforts to mitigate mode collapse follows below.

Several approaches were first introduced by directly modifying the models' objective functions to better approximate the real data distribution through discrepancy measurements [Li+17; MSG17; Sut+17], various divergence functions [NCT16; Ueh+16], energy-based functions [BSM17; Unt+18; ZML17], or unrolled optimization [Met+17]. In another line of work, adding an auto-encoder

model in the framework may be useful to penalize the lack of sample diversity [BSM17; Che+17; Wan+17; WB17]. Other approaches conditioned D's output to consider the similarity of samples in a minibatch [Sal+16] or by leveraging supervised learning to calculate the mutual information between samples and labels [Spr15] as well as the combined discrimination of multiple samples of a given class [Lin+18b].

Increasing the number of generators in the framework [Gho+18; Hoa+18] may also be a viable option. However, it comes with the increased cost of having multiple generative models after discarding the discriminator or discriminators, if using a multi-generator and multi-discriminator setting [CF18; Gan+17; GE18], at the end of training. Increasing solely the number of discriminators in the original GANs framework was also proposed in these early research stages. Specifically, D2GAN [Ngu+17] proposed two discriminators that have different objective functions and provide two complementary rewards to G: while one rewards samples coming from the real data distribution, the other rewards samples likely generated by G. However, this may compromise scalability to bigger discriminator sets, unlike our proposed frameworks. GMAN [DGM17] proposed to use several discriminators with the same common objectives. However, their focus was on restricting the feedback provided from the discriminator ensemble to G in different ways, simulating different levels of difficulty throughout training. However, this may compromise convergence while being task-dependent.

As previously mentioned, the methods above only present the initial efforts to mitigate mode collapse in GANs. The high popularity of GANs led to the extensiveness of the related work being too broad to be fully covered in this thesis. Since then, several popular methods were proposed to stabilize GANs training, some of which will be later discussed in Chapter 5. For now, we find that the presented methods provide appropriate coverage of the existing work at the time of writing of both our multi-adversarial frameworks – Dropout-GAN and microbatchGAN – which will be discussed next.

2.2 Dropout-GAN

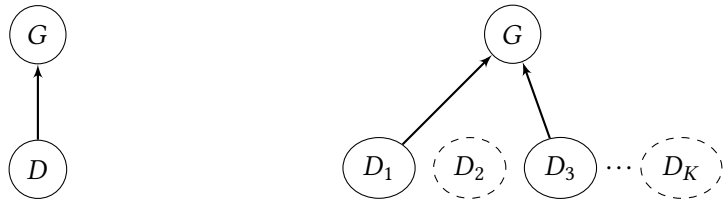
We present our first novel method in Section 2.2.1 and describe important implementation details in Section 2.2.2. In Section 2.2.3, we report experimental results on multiple datasets using different configurations of the proposed framework. Finally, we compare our solution against existing methods in Section 2.2.4.

2.2.1 Adversarial Dropout

Before we dive into describing our approach, let us first discuss dropout [Sri+14], a widely used neural network technique to prevent overfitting [BS13; GG16; War+14]. By omitting or dropping out neurons in a neural network with a probability d, called *dropout rate*, this technique aims at reducing neuron dependency. In other words, dropout prevents neurons to be entirely dependent on a specific set of other neurons to learn their weights, promoting generalization in the neural network by making it less prone to overfit.

Our framework, Dropout-GAN, leverages the dropout principles in generative multi-adversarial networks. More concretely, we first extend the GANs framework to a multi-adversarial setting by introducing an ensemble of K discriminators. Then, by dropping out the feedback of a given discriminator with a probability d, we disentangle G to rely on the feedback of a specific discriminator or discriminator set to learn how to generate realistic samples. Hence, G is guided by a dynamic ensemble of discriminators that is likely to change at every iteration.

With adversarial dropout in mind, mode collapse may be seen as a consequence of overfitting to the feedback of a static ensemble of discriminators. Hence, by continuously and dynamically changing the discriminator ensemble during training, G is stimulated to induce sample variety in the generated set to ultimately increase its chances of fooling the different possible discriminators that may be in the ensemble at a given iteration. Figure 2.1 illustrates the proposed multi-adversarial framework.



(a) Original GANs [Goo+14]. **(b)** Dropout-GAN.

Figure 2.1: We extend the original GANs framework (left) to K discriminators (right), where some of which are dropped out (dashed circles) according to some probability, leading to a random subset of feedback (represented by the arrows) being used by G at each training iteration.

Considering a Bernoulli variable $\delta_k \sim Bern(1-d)$ and a set of K discriminators $\{D_k\}$, with $k \in \{1, \ldots, K\}$, the output of a given discriminator D_k is only used to update G if $\delta_k = 1$. (Note that $P(\delta_k = 1) = 1 - d$.) Our initial modification of the minimax game's value function V' is then the following:

$$\min_G \max_{\{D_k\}} \sum_{k=1}^{K} V'(D_k, G) = \sum_{k=1}^{K} \delta_k \Big(\mathbb{E}_{x \sim p_r(x)} [\log D_k(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D_k(G(z)))] \Big). \tag{2.2}$$

Our second and final modification is to consider the possibility of no discriminator being left in the dynamic ensemble at a given iteration. This is not ideal since G would have no guidance. To accommodate this case, if all the discriminators are dropped out, we uniformly pick one discriminator D_j at random, with $j \in \{1, \ldots, K\}$, and follow the original GANs value function V, described in (2.1). Hence, we define our final value function, F, as:

$$F(G, \{D_k\}) = \begin{cases} \min_G \max_{\{D_k\}} \sum_{k=1}^{K} V'(D_k, G), & \text{if } \exists k : \delta_k = 1 \\ \min_G \max_{D_j} V(D_j, G), & \text{otherwise, for } j \in \{1, ..., K\}. \end{cases} \tag{2.3}$$

We note that we treat each discriminator independently, *i.e.* each discriminator is not aware of the existence of a discriminator ensemble. Hence, despite being dropped out or not, all discriminators still update their parameters at every iteration. Our training procedure is presented in Algorithm 1.

2.2.2 Implementation Details

We will provide important insights about the effects of using a different number of discriminators, K, as well as different dropout rates, d. Moreover, we discuss how the splitting of samples among the different discriminators is performed. The proposed framework was implemented using Tensorflow [Aba+16].

Algorithm 1 Dropout-GAN.

Input: K discriminators, d dropout rate, B batch size
$m \leftarrow \frac{B}{K}$
for each iteration **do**
  for $k = 1$ to K **do**
    • Sample minibatch z_i, $i = 1 \dots m$, $z_i \sim p_g(z)$
    • Sample minibatch x_i, $i = 1 \dots m$, $x_i \sim p_r(x)$
    • Update D_k by ascending along its gradient:

$$\nabla_{\theta_{D_k}} \frac{1}{m} \sum_{i=1}^{m} [\log D_k(x_i) + \log(1 - D_k(G(z_i)))]$$

  end for
  • Sample minibatch δ_k, $k = 1 \dots K$, $\delta_k \sim Bern(1 - d)$
  if $\delta_1 = 0 \wedge \dots \wedge \delta_K = 0$ **then**
    • Sample minibatch z_i, $i = 1 \dots m$, $z_i \sim p_g(z)$
    • Update G by descending along its gradient using a random discriminator D_j, for some $j \in \{1, ..., K\}$:

$$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D_j(G(z_i)))$$

  else
    • Sample minibatch z_{k_i}, $i = 1 \dots m$, $k = 1 \dots K$, $z_{k_i} \sim p_g(z)$
    • Update G by descending along its gradient:

$$\nabla_{\theta_G} \sum_{k=1}^{K} \delta_k \Big(\frac{1}{m} \sum_{i=1}^{m} \log(1 - D_k(G(z_{k_i}))) \Big)$$

  end if
end for

Number of discriminators

As previously addressed in Section 2.1, GANs are prone to show signs of training instability. Previous works have shown that increasing the number of adversaries in the framework may act as a simple yet effective way of reducing such instability [DGM17; NBC17]. More specifically, the chances of G receiving positive feedback are greater if using an ensemble of discriminators, which has been shown to improve the training even when using the original GANs formulation [DGM17].

By correlating training instability with the magnitude of G's gradient updates throughout training, we observed that using a static ensemble of discriminators helped the gradients to converge to zero faster when compared to using only one discriminator. This suggests that G's learning is more smooth and continuous over time, instead of experiencing drastic changes which are likely not beneficial, especially after the early training stages. We further observed that increasing the number of discriminators in the static ensemble delays the gradient to converge which may be explained by the conflicting feedback from the different discriminators at the beginning of training.

Batch partitioning

Intuitively, having an ensemble of discriminators is most beneficial if each discriminator provides unique, but complementary, feedback. Hence, to encourage each discriminator to specialize in different aspects of the data, we partition both the real and fake batches among the different discriminators. Hence, at a given iteration, each discriminator is trained with a different set of real and fake samples. We observed that G's loss was higher when batches were partitioned, which suggests the reduction in G's capacity of fooling the entire discriminator ensemble.

In terms of total batch size, we keep the same number of samples at each discriminator batch, independently of the size of the discriminator ensemble. On the other hand, this implies that G has access to feedback from more samples, proportionally to the ensemble size. However, having weaker discriminators compared to G, by providing them access to fewer samples, is likely to lead to more positive feedback, which correlates better to improved learning than continuous negative feedback [DGM17; NBC17].

Dropout rate

Variability in the discriminator set is controlled by a dropout rate (d) which determines if a given discriminator's feedback is omitted to G. This creates a dynamic ensemble of discriminators that is likely to change at each iteration. Hence, our adversarial dropout may also be seen as a form of regularization in G's training. The rate at which the ensemble may change, directly depends on the used dropout rate, with $d \approx 1$ leading to an often varying ensemble whereas $d \approx 0$ leads to a nearly static ensemble. The different variance levels in the ensemble directly impact G's learning and deserve consideration.

We will see in Section 2.2.3 that $0.2 < d \leq 0.5$ leads to consistently improved results when compared to using a static ensemble of adversaries, *i.e.* $d = 0$. These values are in conformity to the reported values in the original dropout paper [Sri+14]. In our use case, this suggests that adding variability to the ensemble positively influences G's learning, however, too much variability in the feedback received by G is likely to lead to poorer learning overall. A discussion regarding the correlation between different dropout rates and the number of discriminators in the realism of the generated set follows below.

2.2.3 Experimental Results

We start by evaluating Dropout-GAN on three image datasets: MNIST [LCB10], the 10-handwritten-digit dataset, CIFAR-10 [Kri09], the 10-class dataset of vehicles and animals, and CelebA [Liu+15], the cropped celebrity faces dataset. We borrowed the DCGAN-like architecture [RMC16] and training settings proposed by GMAN [DGM17]. Namely, G makes use of 4 convolutional layers with 128, 64, and 32 channels for the first three layers and either 1 channel (MNIST) or 3 channels (CIFAR-10 and CelebA) in the last convolutional layer. On the other hand, D employs the earlier 3 convolutional layers in reverse order with an additional fully connected layer at the end for classification purposes. Note that, even though all discriminators share the same architectural design, their weights are initialized differently.

The main metric used throughout this Section is the Fréchet Inception Distance (FID) [Heu+17], which retrieves a distance measure reflecting the similarity of the real and fake distributions. FID is discussed and evaluated in detail in Chapter 4. For now, it suffices to say that FID is more robust to noise while showing a higher correlation to human evaluation compared to other single-valued metrics, such

Table 2.1: Minimum FID obtained across 40 epochs on the different datasets. For each dataset, the overall best FID is presented in bold. For each framework configuration, the best FID is underlined.

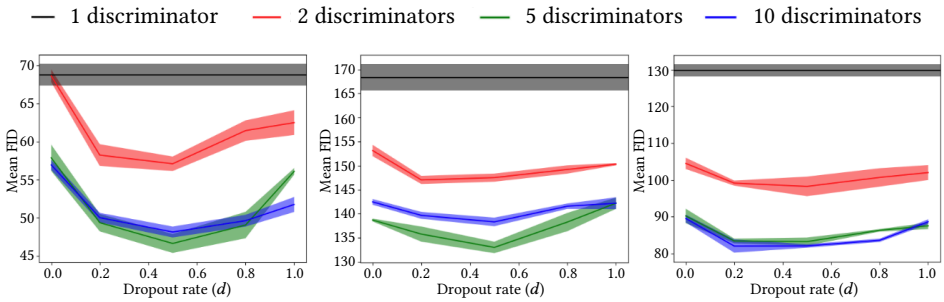
	MNIST	CIFAR-10	CelebA
1 disc.	21.71 ± 0.39	104.19 ± 0.07	53.38 ± 0.03
2 disc.; $d = 0.0$	24.88 ± 0.13	106.54 ± 0.38	52.46 ± 0.08
2 disc.; $d = 0.2$	22.34 ± 0.29	103.55 ± 0.13	46.60 ± 0.03
2 disc.; $d = 0.5$	22.08 ± 0.09	103.20 ± 0.05	45.90 ± 0.04
2 disc.; $d = 0.8$	21.87 ± 0.10	103.60 ± 0.03	46.82 ± 0.14
2 disc.; $d = 1.0$	23.56 ± 0.29	104.73 ± 0.19	51.17 ± 0.01
5 disc.; $d = 0.0$	21.47 ± 0.40	95.75 ± 0.15	45.89 ± 0.05
5 disc.; $d = 0.2$	21.70 ± 0.12	90.59 ± 0.35	**36.36 ± 0.11**
5 disc.; $d = 0.5$	19.25 ± 0.12	89.74 ± 0.35	38.10 ± 0.54
5 disc.; $d = 0.8$	20.26 ± 0.07	90.77 ± 0.70	41.22 ± 0.24
5 disc.; $d = 1.0$	20.54 ± 0.15	95.71 ± 0.03	41.56 ± 0.18
10 disc.; $d = 0.0$	22.62 ± 0.10	99.91 ± 0.10	43.85 ± 0.30
10 disc.; $d = 0.2$	19.12 ± 0.01	91.31 ± 0.16	41.74 ± 0.14
10 disc.; $d = 0.5$	**18.18 ± 0.44**	**88.60 ± 0.08**	40.67 ± 0.56
10 disc.; $d = 0.8$	19.33 ± 0.18	88.76 ± 0.16	41.74 ± 0.03
10 disc.; $d = 1.0$	19.82 ± 0.06	93.66 ± 0.21	41.16 ± 0.55

as the popular Inception Score [Sal+16]. More importantly, FID is also able to detect lack of diversity in the generated set and may be used to assess signs of mode collapse [Luc+18].

Minimum FID

We present the minimum FID over 40 epochs for different configurations, calculated from 10K fake samples at the end of every epoch. Results obtained over 2 runs are presented in Table 2.1 (the same applies for the following tables in this Chapter). The inferior results with 10 discriminators on CelebA may be due to longer training being required since this is the most challenging dataset.

We observe that the best minimum FID across the different datasets and ensemble size were often obtained with $d = 0.5$, even though competitive results



(a) MNIST. **(b)** CIFAR-10. **(c)** CelebA.

Figure 2.2: Mean FID calculated using different dropout rates and number of discriminators across 40 epochs on the different datasets. The convex representations suggest the benefits of using mid-range dropout rates.

were also achieved with $d = 0.2$ and $d = 0.8$. Overall, we see that applying adversarial dropout achieves lower minimum FID values than using a static ensemble, *i.e.* $d = 0$, or the original GANs framework, *i.e.* $d = 0$ and $K = 1$.

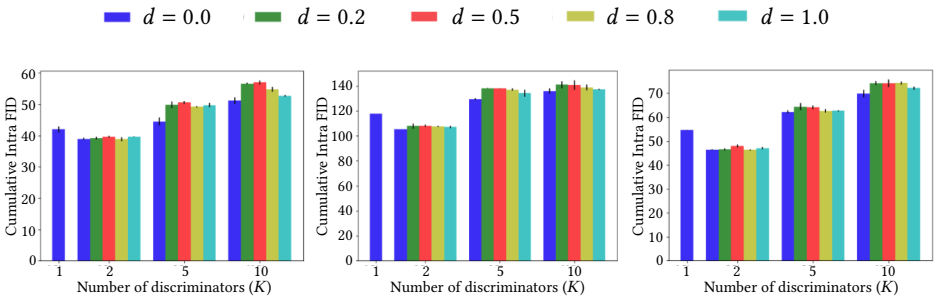
Mean FID

To have a better understanding of the realism of the generated set during training, we also calculated the mean FID across 40 epochs. Smaller mean FID values suggest the generation of more realistic samples over time. Results are shown in Figure 2.2. We observe that mid-range dropout rates tend to perform better over time over the different datasets. Moreover, the advantage of using a discriminator ensemble instead of only one D is also noticeable across the board.

Even though using 10 discriminators seems to have an identical mean FID then using 5 discriminators, we note that this may be explained by G requiring more time to learn from the additional, perhaps contradictory, feedback in the early to mid iterations. However, the better performance in terms of minimum FID when using 10 discriminators previously presented in Table 2.1 implies that G is eventually able to produce more realistic samples in the later training stages.

Cumulative Intra FID

We further introduce Intra FID, which is used to compare 10K generated samples from subsequent epochs. The goal of this metric is to assess how samples differ



(a) MNIST. **(b)** CIFAR-10. **(c)** CelebA.

Figure 2.3: Cumulative Intra FID using a different number of discriminators and dropout rates across 40 epochs on the different datasets. Higher values suggest more sample diversity over time.

over time, specifically in terms of sample diversity. We used the cumulative Intra FID for each model to assess its overall generation process. Results are presented in Figure 2.3.

We observe that increasing the size of the discriminator ensemble leads to an increase of cumulative Intra FID across the different datasets and dropout rates. We also see a slight increase of cumulative Intra FID in the mid-range dropout rates, especially when using 5 or 10 discriminators.

2.2.4 Method Comparisons

We now compare Dropout-GAN against existing diversity-inducing methods in a variety of metrics and datasets.

Mode coverage on a toy dataset

We start by visualizing the mode coverage of different frameworks on a 2D mixture of 8 Gaussian distributions. The goal of this toy experiment originally proposed by Metz et al. [Met+17], is to treat each distribution as a data mode. We used 8 discriminators and followed D2GAN's architecture and training details. Results are presented in Figure 2.4.

We observe that our framework, Dropout-GAN, covers all data modes across all training stages, unlike the original GANs framework which shows clear signs of mode collapse. Compared to the other discriminator-based methods, *i.e.*

unrolled optimization of D (UnrolledGAN) and dual-discriminators (D2GAN), Dropout-GAN shows fewer signs of noise, especially in the early training steps. We note that MGAN makes use of multiple generators plus an additional classifier while using a bigger architecture. We find this to likely be a big contributor to MGAN's superior results, especially later on in training.

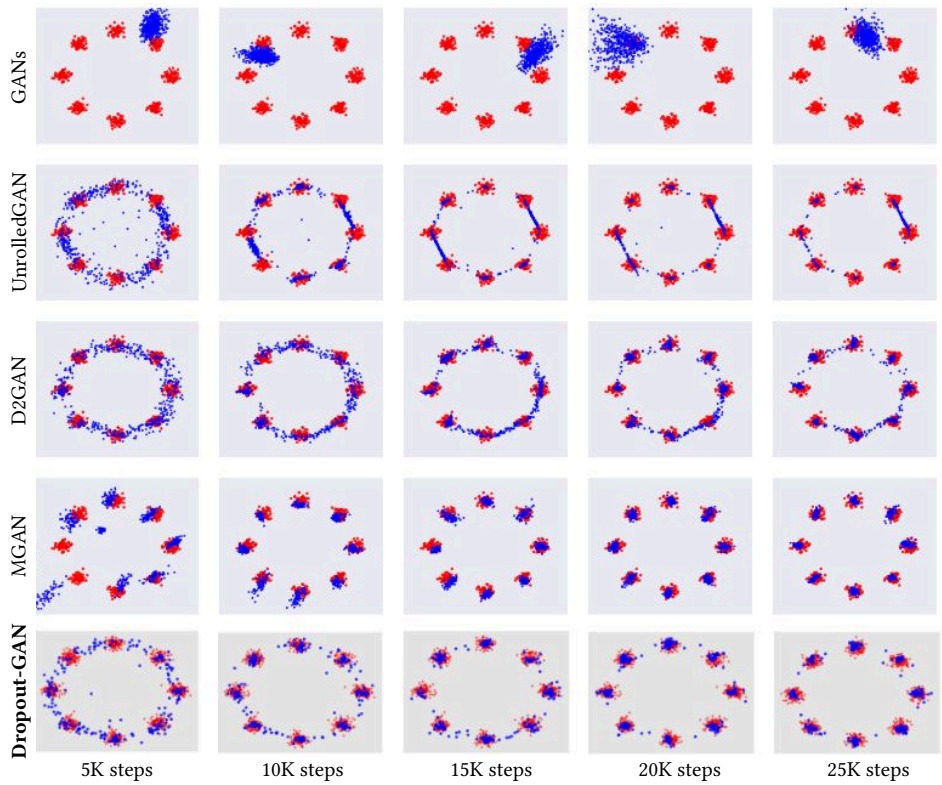


Figure 2.4: Method comparisons on a toy dataset. Real data is presented in red while generated data is presented in blue. Dropout-GAN successfully covers all data modes throughout training.

FID on MNIST, CIFAR-10, and CelebA

We will now go back to quantitative evaluation by assessing the minimum FID of different frameworks trained on MNIST, CIFAR-10, and CelebA for 20, 40,

and 100 epochs, respectively. We compare Dropout-GAN against other popular GANs modifications presented by Lucic et al. [Luc+18]. Namely, we consider the original and modified formulations, *i.e.* GANs and modGAN [Goo+14], respectively, as well as applying gradient penalty, *i.e.* DRAGAN [Kod+17], and using an unbounded output for each D, *i.e.* LSGAN [Mao+17]. We used a bigger version of the convolutional architecture previously presented in Section 2.2.3 by simply doubling the number of channels. Results are presented in Table 2.2.

Table 2.2: Minimum FID of the different methods on MNIST, CIFAR-10 and CelebA. Lower values are better. In our adversarial dropout variants, we used $K = 2$ and $d = 0.5$.

	MNIST	CIFAR-10	CelebA
Real data	≈ 0.00	≈ 0.00	≈ 0.00
GANs ([Goo+14])	22.65 ± 0.13	70.23 ± 0.07	46.18 ± 0.07
Dropout-GAN	$\mathbf{14.63 \pm 0.18}$	$\mathbf{66.82 \pm 0.10}$	$\mathbf{31.25 \pm 0.09}$
modGAN ([Goo+14])	22.66 ± 0.11	79.58 ± 0.11	41.25 ± 0.03
Dropout-modGAN	$\mathbf{15.39 \pm 0.15}$	$\mathbf{67.57 \pm 0.14}$	$\mathbf{35.32 \pm 0.06}$
LSGAN ([Mao+17])	24.05 ± 0.15	83.66 ± 0.08	43.13 ± 0.04
Dropout-LSGAN	$\mathbf{15.41 \pm 0.21}$	$\mathbf{69.37 \pm 0.11}$	$\mathbf{37.58 \pm 0.10}$
DRAGAN ([Kod+17])	22.84 ± 0.15	80.57 ± 0.06	46.82 ± 0.06
Dropout-DRAGAN	$\mathbf{15.20 \pm 0.16}$	$\mathbf{66.90 \pm 0.09}$	$\mathbf{37.21 \pm 0.08}$

We observe that applying the proposed adversarial dropout reduces the minimum FID across the different GANs variations and datasets. For an unbiased comparison against the 1-discriminator baselines, we only used 2 discriminators for the adversarial dropout variants. Despite not reported, we note that using static discriminator ensembles, *i.e.* $d = 0$, showed worse and less consistent performance than our dynamic ensembles. Specifically, on CIFAR-10, our reported results lower the minimum FID of static ensembles by 7.25, 4.39, 3.96, and 9.93, on GANs, modGAN, LSGAN, and DRAGAN, respectively.

Comparison to GMAN on CIFAR-10

To further investigate the efficacy of adversarial dropout in a multi-discriminator setting, we directly compared Dropout-GAN to GMAN [DGM17] using 2 and 5 discriminators. We used different GMAN variations which use the mean loss of all discriminators, *i.e.* GMAN-0, the maximum discriminator loss in the ensemble, *i.e.* GMAN-1, or a learned hyper-parameter λ variant controlled by G, *i.e.* GMAN*. We also report results with the modified GANs variant, *i.e.* modGAN, as a baseline.

Since GMAN's reported results on CIFAR-10 assess the Inception Score or IS [Sal+16], we follow the same experimental setup. Higher IS values suggesting a more realistic generated set, both in terms of quality and diversity. Similar to FID, we discuss IS in more detail in Chapter 4. For a fair comparison, we used the architectural and training details reported by GMAN. Results are presented in Table 2.3.

Table 2.3: Inception score (IS) of a GANs baseline with modified loss and different variants of GMAN and Dropout-GAN on CIFAR-10. Underlined values represent the best scores within each framework variation. Bold values refer to the best scores under a certain number of discriminators.

	1 disc.	2 disc.	5 disc.
modGAN ([Goo+14])	5.74 ± 0.17	-	-
GMAN-0 ([DGM17])	-	5.88 ± 0.19	5.96 ± 0.14
GMAN-1 ([DGM17])	-	5.77 ± 0.16	6.00 ± 0.19
GMAN* ([DGM17])	-	5.54 ± 0.09	5.96 ± 0.15
Dropout-modGAN ($d = 0.2$)	-	5.95 ± 0.10	6.01 ± 0.12
Dropout-modGAN ($d = 0.5$)	-	**5.98 ± 0.10**	**6.05 ± 0.15**

We observe that all adversarial dropout variants outperform the modified GANs baseline. More importantly, we outperform GMAN's variants across different discriminator ensemble sizes. We further observe that using $d = 0.5$ consistently provides the best performances in our framework.

Inception Scores on CIFAR-10, STL-10, and ImageNet

We further assessed how the proposed adversarial dropout translates to larger image datasets. Namely, on top of CIFAR-10, we trained the different GANs on STL-10 [CNL11], containing 100K of both labeled and unlabelled images, and ImageNet [Rus+15], the 1M-image dataset. Despite being higher quality datasets, we downsized all images to 32x32 pixels so we could re-use the same architectural setting as before (Table 2.2) for the different datasets. However, we trained CIFAR-10 and STL-10 for 250 epochs and ImageNet for 50 epochs. The obtained IS are presented in Table 2.4.

Table 2.4: Inception score (IS) of different GANs on CIFAR-10, STL-10, and ImageNet. We used 2 discriminators and $d = 0.5$ for all the adversarial dropout methods represented in bold.

	CIFAR-10	STL-10	ImageNet
Real data	11.24 ± 0.16	26.08 ± 0.26	25.78 ± 0.47
GANs ([Goo+14])	5.35 ± 0.04	5.53 ± 0.03	7.30 ± 0.08
Dropout-GAN	**6.22 ± 0.09**	**7.20 ± 0.11**	**7.52 ± 0.13**
modGAN ([Goo+14])	5.49 ± 0.07	6.64 ± 0.05	6.96 ± 0.08
Dropout-modGAN	**5.90 ± 0.08**	**6.95 ± 0.09**	**7.26 ± 0.12**
LSGAN ([Mao+17])	5.76 ± 0.05	5.32 ± 0.06	6.92 ± 0.04
Dropout-LSGAN	**5.95 ± 0.07**	**6.88 ± 0.13**	**7.08 ± 0.13**
DRAGAN ([Kod+17])	5.65 ± 0.08	6.97 ± 0.09	7.41 ± 0.11
Dropout-DRAGAN	**6.22 ± 0.08**	**7.30 ± 0.13**	**7.54 ± 0.12**

We observe the benefits of applying adversarial dropout in terms of IS on the different GANs and datasets. This goes in accordance with the previous results using FID on the smaller datasets in Table 2.2.

In Figure 2.5, we provide a glance at a random subset of the generated images using the previous GANs trained with adversarial dropout on the different datasets. Overall, we observe high sample diversity, even on the larger datasets. The extensiveness of the empirical studies presented in this Section suggests the general applicability and success of our approach in mitigating mode collapse.

Figure 2.5: Generated samples on CIFAR-10, STL-10, and ImageNet using different GANs and adversarial dropout.

2.3 microbatchGAN

Now, we introduce our other multi-adversarial framework called microbatchGAN. We follow a similar structure as Section 2.2 and introduce our new framework in Section 2.3.1, together with theoretical discussions. Then, we discuss several implementation details of our new method in Section 2.3.2. Finally, we analyze different configurations of our approach in Section 2.3.3 and compare them to existing methods in Section 2.3.4.

2.3.1 Multi-Adversarial Microbatch Discrimination

In microbatchGAN, we also propose to use multiple discriminators, *i.e.* $K > 1$. However, instead of applying adversarial dropout, we introduce a new mecha-

nism: multi-adversarial microbatch discrimination. In sum, our approach starts by splitting the real and fake minibatches into different but complementary portions called microbatches. Then, we change the objective function of each D to distinguish between fake samples in its microbatch and the rest of the fake samples present in the microbatches assigned to the other discriminators. On top of this, similarly to the original GANs formulation [Goo+14], each D also distinguishes between samples in its assigned fake microbatch and samples from its assigned real microbatch. Figure 2.6 illustrates the proposed framework.

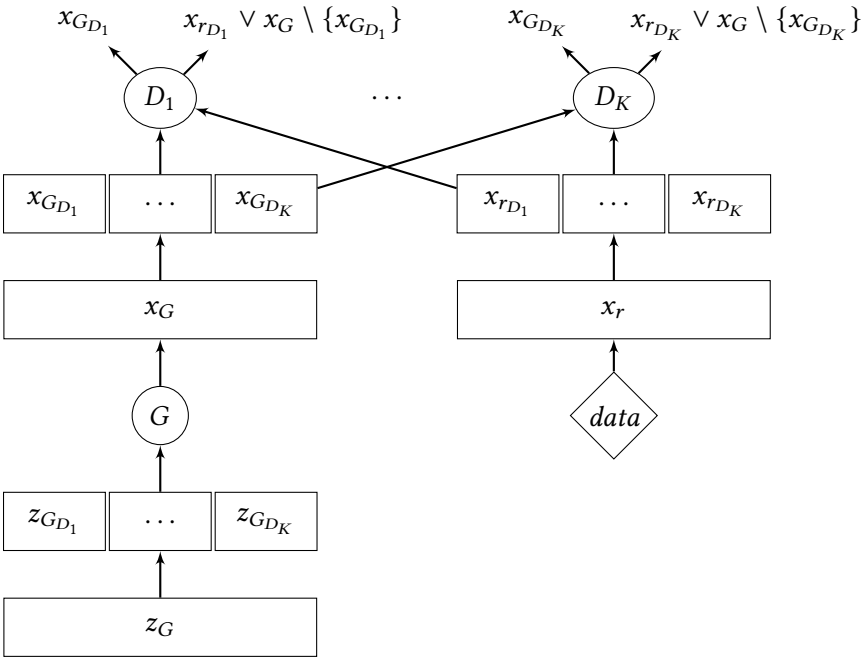


Figure 2.6: Our proposed framework, microbatchGAN, uses multiple discriminators and microbatch discrimination. Each discriminator D_k is assigned a unique fake ($x_{G_{D_k}}$) and real ($x_{r_{D_k}}$) microbatch obtained from partitioning a fake (x_G) and real (x_r) minibatch, respectively. Note that x_G is generated using a noise minibatch (z_G), which may also be partitioned and assigned to each D_k ($z_{G_{D_K}}$). D_k's task is then to distinguish between samples from its fake and real microbatches as well as samples from the rest of the microbatches assigned to the other discriminators, *i.e.* $x_G \setminus \{x_{G_{D_k}}\}$.

We gradually induce microbatch discrimination by introducing a diversity parameter α. Hence, the discriminator objectives change throughout training:

starting from simply distinguishing real and fake samples, increasing α stimulates each D to also distinguish between fake samples assigned to its microbatch and fake samples assigned to the other discriminator's microbatches. The crux of our algorithm is to ultimately promote sample diversity in G's generated minibatch through α, making the microbatch discrimination performed by each D harder. To prevent the production of identical samples, we initialize each D differently and train them with unique microbatches. Hence, each D is likely to focus on different data properties, making it unlikely that similar fake samples would continuously fool the entire discriminator ensemble.

Considering microbatch discrimination, *i.e.* $\alpha > 0$, we add a new term to the original GANs value function presented in (2.1). Particularly, we change each D's objective to assign low probabilities to fake samples from its microbatch but high probabilities to the fake samples assigned to the rest of the discriminators. Hence, within a fake minibatch, fake samples shall be given distinct probabilities by each D. Moreover, G's objectives are also changed to promote sample variety in the fake minibatch. Specifically, and conversely to each D, G minimizes its loss by fooling each D into assigning high probabilities to samples in its microbatch but low probabilities to the rest of the fake samples in the minibatch. Hence, all models in our framework benefit from sample variety. The value function F' of our minimax game is then:

$$\min_G \max_{\{D_k\}} \sum_{k=1}^{K} F'(D_k, G) = \sum_{k=1}^{K} \mathbb{E}_{x \sim p_{r_{D_k}}(x)} [\log D_k(x)] + \mathbb{E}_{z \sim p_{z_{G_{D_k}}}(z)} [\log(1 - D_k(G(z)))] + \alpha \times \mathbb{E}_{z' \sim p_{z_G \setminus \{z_{G_{D_k}}\}}(z')} [\log D_k(G(z'))], \tag{2.4}$$

where, considering K discriminators and $k \in \{1, \ldots, K\}$, $p_{r_{D_k}}$ and $p_{z_{G_{D_k}}}$ represent real and fake samples from D_k's real and fake microbatches, respectively, and $p_{z_G \setminus \{z_{G_{D_k}}\}}$ indicates the rest of the fake samples in D_k's fake microbatch. The incorrect microbatch discrimination by each D is penalized to different extent depending on the diversity parameter α. This implies each D follows its original GANs objective if $\alpha = 0$. Our training procedure is presented in Algorithm 2.

Algorithm 2 microbatchGAN.

Input: K discriminators, α diversity parameter, B minibatch size
$m \leftarrow \frac{B}{K}$
for number of training iterations **do**
- Sample minibatch z_i, $i = 1 \dots B$, $z_i \sim p_g(z)$
- Sample minibatch x_i, $i = 1 \dots B$, $x_i \sim p_r(x)$

for $k = 1$ to K **do**
- Sample microbatch z_{k_j}, $j = 1 \dots m$, $z_{k_j} = z_{(k-1)\times m+1:k\times m}$
- Sample microbatch x_{k_j}, $j = 1 \dots m$, $x_{k_j} = x_{(k-1)\times m+1:k\times m}$
- Sample microbatch z'_{k_j}, $j = 1 \dots m$, $z'_{k_j} \subset z_i \setminus \{z_{k_j}\}$
- Update D_k by ascending its stochastic gradient:

$$\nabla_{\theta_{D_k}} \frac{1}{m} \sum_{j=1}^{m} [\log D_k(x_{k_j}) + \log(1 - D_k(G(z_{k_j}))) + \alpha \times \log D_k(G(z'_{k_j}))]$$

end for
- Update G by descending its stochastic gradient:

$$\nabla_{\theta_G} \sum_{k=1}^{K} \Big[\frac{1}{m} \sum_{j=1}^{m} [\log(1 - D_k(G(z_{k_j}))) + \alpha \times \log D_k(G(z'_{k_j}))] \Big]$$

end for

Theoretical discussions

To take a deeper look at how our framework promotes sample variety in the generated set, we now provide some theoretical discussions that build on previous work [Goo+14; Ngu+17]. Namely, we consider a simplistic version of the proposed minimax game by not training, *i.e.* freezing, each D but training G until convergence. Moreover, we examine the most extreme case of mode collapse where each generated sample x is identical to one another. Hence, following the original GANs formulation in (2.1), we define mode collapse in its acute form as:

$$\text{For all } z' \sim p_g(z), G(z') = x. \tag{2.5}$$

► **Theorem 2.1.** In original GANs, mode collapse fully minimizes G's loss when we train G exhaustively without updating D. ◄

Proof. Let us consider an optimal (unique) fake sample x^* that maximizes D's output: $x^* = \underset{x}{\text{argmax}}\, D(x)$. Training G to exhaustion until it learns how to generate x^*, and making it independent of z, would result in mode collapse and a fully converged G. ■

► **Theorem 2.2.** In microbatchGAN, multi-adversarial microbatch discrimination, *i.e.* $K > 1$ and $\alpha > 0$, forces $x \sim p_g$ to be dependent of z for G to fully minimize its loss against a frozen discriminator ensemble. ◄

Proof. Considering (2.4), the value function between G and each D_k may be expressed as:

$$F'(D_k, G) = \mathbb{E}_{x\sim p_r}[\log D_k(x)] + \mathbb{E}_{x'\sim p_g}[\log(1 - D_k(x'))] + \alpha \times \mathbb{E}_{x''\sim p_g}[\log D_k(x'')]. \tag{2.6}$$

To fully minimize its loss in relation to D_k, G must find

$$x' = \underset{x}{\text{argmax}}\, D_k(x) \text{ and } x'' = \underset{x}{\text{argmin}}\, D_k(x), \tag{2.7}$$

which implies

$$D_k(x') \neq D_k(x'') \implies x' \neq x''. \tag{2.8}$$

Thus, sample generation must be dependent on the noise z to fully minimize G's loss regarding each D_k. Since, going back to (2.4), we simply sum all $F'(D_k, G)$ to calculate G's final loss, we extend this discussion to the discriminator ensemble, concluding the proof. ■

2.3.2 Implementation Details

The introduction of the diversity parameter α raises important questions regarding its effects on microbatch discrimination and consequent mitigation of mode collapse. Intuitively, small α values would likely neglect the importance of microbatch discrimination in our framework. On the other hand, large α values may induce too much weight on microbatch discrimination, possibly sacrificing sample quality in the end. We will discuss possible α implementations next.

Constant diversity parameter α

We first acknowledge the most straightforward implementation of directly setting α to a constant value during training. For a quick assessment, we make use of the toy dataset previously described in Section 2.2.4 consisting of 8 data modes from 8 Gaussian distributions. Similarly to Dropout-GAN, we use 8 discriminators in these experiments. Results are shown in Figure 2.7.

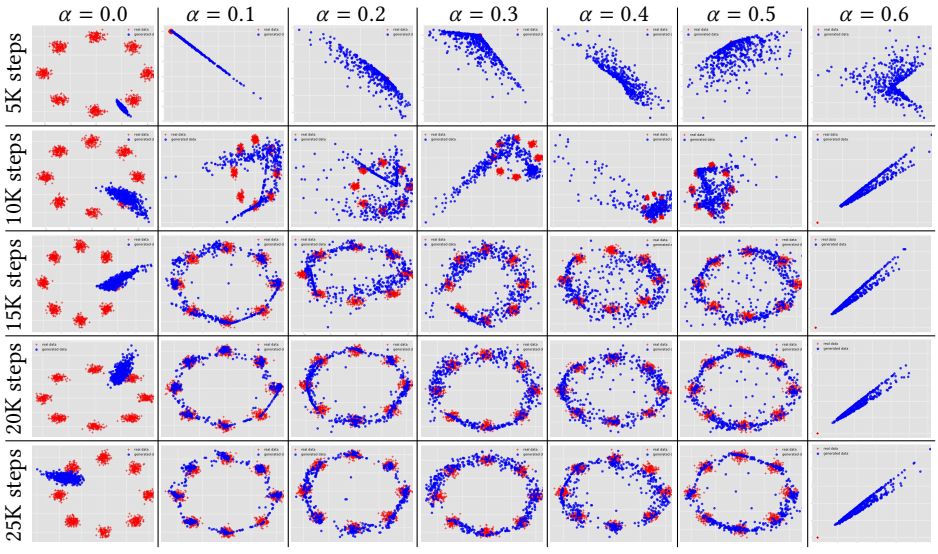


Figure 2.7: Mode coverage on a toy dataset using constant α values. Real data is presented in red while generated data is presented in blue.

We observe that the original GANs formulation ($\alpha = 0$) shows clear signs of mode collapse by only generating samples around a given data mode throughout training. After the initial training steps, $0.1 \leq \alpha \leq 0.5$ shows good coverage of all data modes, successfully mitigating mode collapse. However, considering the initial training stages, we observe that G primarily focuses on sample diversity, without much consideration being given to sample quality. This is even more predominant with bigger α values, *i.e.* $\alpha \geq 0.6$. These results suggest that, while constant α values successfully stimulate sample diversity in the generated set, care must be taken to not undermine sample quality, especially from the initial training stages.

Learned diversity parameter α

Considering the poor sample quality and sample diversity trade-off observed in the early training stages when using a constant α, learning an optimal α may then be a better approach. However, when adding α as a parameter of G, we noticed that G simply learns to increase α at a large rate to reduce its loss. Hence, sample quality continued to be neglected with this approach. To fix this, we suggest α to have the following behavioral properties:

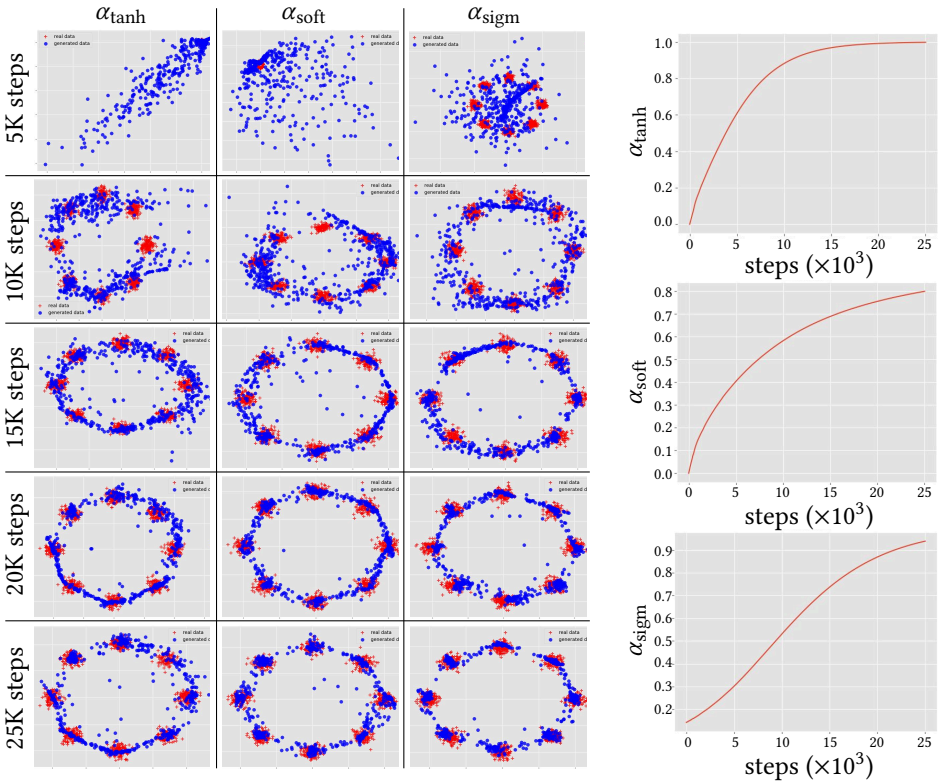
- *Upper boundness*: To avoid microbatch discrimination to overpower the original GANs objective in (2.4), we suggest that α should be upper bounded.
- *Saturated growth*: To prevent the continuous increase of α to lower G's loss, we propose α's growth to saturate over time.
- *Controlled growth*: To balance sample quality and sample diversity in the earlier training stages, we note that α's growth should be performed in a controlled fashion.

To accommodate such properties, we propose to make α a function of a learned dummy parameter β, where $\alpha(\beta) \in [0, 1[$, and let G learn β instead of directly regulating α. We experimented with the following functions which manipulate α's growth differently over time:

$$\alpha(\beta) = \begin{cases} \alpha_{\text{sigm}}(\beta) = \text{Sigmoid}(\beta), \beta \geq \beta_{\text{sigm}} \\ \alpha_{\text{soft}}(\beta) = \text{Softsign}(\beta), \beta \geq \beta_{\text{soft}} \\ \alpha_{\text{tanh}}(\beta) = \text{Tanh}(\beta), \beta \geq \beta_{\text{tanh}} \\ \alpha_{\text{id}}(\beta) = \beta, \end{cases} \tag{2.9}$$

where β_{sigm}, β_{soft}, and β_{tanh} indicate the initialized β value for the respective functions. In our experiments, we use $\beta_{\text{tanh}} = \beta_{\text{soft}} = 0$, and $\beta_{\text{sigm}} = -1.8$ obtained through empirical observations. We reformulate the initial study of directly learning α by using the identify function α_{id}. We tested the different functions in Figure 2.8 by re-using our toy dataset.

We observe that increasing α in a milder fashion (α_{sigm}) promotes a good trade-off between sample quality and sample diversity, especially in the initial training steps. The dominance of sample diversity over sample quality is heavily observed



(a) Mode coverage.

(b) Evolution of different α.

Figure 2.8: Effects of using different α functions on the toy dataset. The generated samples (blue) and real samples (red) are shown in (a). The evolution of α considering each function is presented in (b), according to the number of training steps.

at the beginning of training if using α_{tanh} and, even though less significantly, α_{soft}. This is intuitive since both functions increase α at a steeper rate. However, due to the saturating behavior of all functions, we observe that, given enough training steps, G achieves a good balance of sample quality and sample diversity across the different α.

Considering the objectives of both G and the discriminator ensemble, we may summarize the effects of the different α functions on the training in the following way: in the first training stages, α increases as a measure to reduce G's loss, inducing sample variety. In the mid to late training stages, as α saturates and

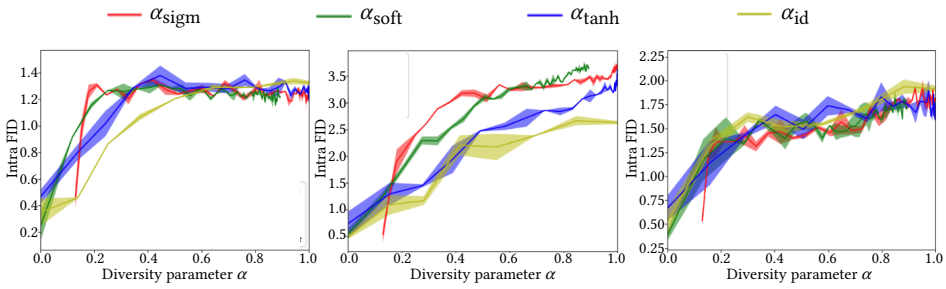
each D better distinguishes real and fake samples, G is stimulated to promote both sample quality and sample diversity in the generated set.

2.3.3 Experimental Results

Following the previous Dropout-GAN experiments in Section 2.2.3, we validated the effects of using different α functions on MNIST [LCB10], CIFAR-10 [Kri09], and cropped CelebA [Liu+15] datasets. Moreover, to quantitatively evaluate the effects of the number of discriminators and different α, we also quantitatively assessed the generated sets by using different variations of FID [Heu+17].

Intra FID

We start out by assessing the Intra FID, which was previously introduced to measure sample variety over time. Note that we do not use the real sample set for this measurement. Instead, we compare two sets of 10K randomly generated samples at the end of every thousand iterations. Hence, higher Intra FID suggests a change in the generated sets over time, which we assume to correlate with G's ability to promote sample diversity. Figure 2.9 presents the evolution of Intra FID over different α values.



(a) MNIST. **(b)** CIFAR-10. **(c)** CelebA.

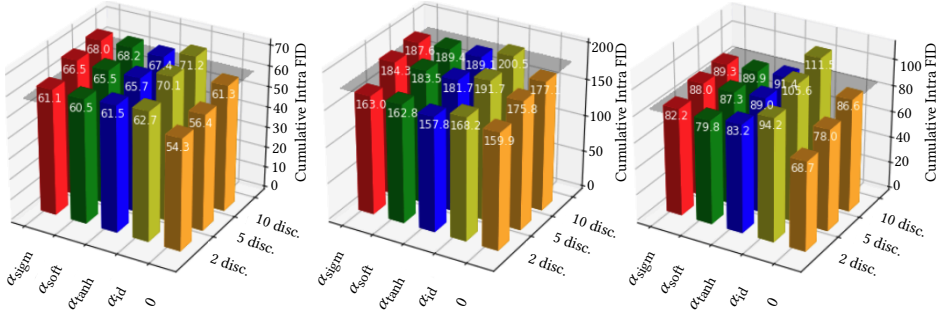
Figure 2.9: Intra FID as α progresses on the different datasets. Higher values suggest higher sample diversity in the generated set over time.

We observe that as α values progress, Intra FID increases at a fast rate, especially at lower α values which relate to earlier training stages. Higher α values maintain the sample diversity in the generated set, represented by the stagnation

(MNIST) or increase (CIFAR-10 and CelebA) of Intra FID. We further see that different α functions have different effects on the generated set as measured by the Intra FID, especially on MNIST and CIFAR-10. However, as α increases and training progresses, Intra FID tends to converge similarly across the different functions, specifically on MNIST and CelebA.

Cumulative Intra FID

Similar to our previous study for Dropout-GAN, we also accumulated all Intra FID values to assess sample diversity over time. Higher values should then reflect a higher variety in the generated set across different iterations. The cumulative Intra FID results using different α and K are presented in Figure 2.10.



(a) MNIST. **(b)** CIFAR-10. **(c)** CelebA.

Figure 2.10: Cumulative Intra FID of different microbatchGAN configurations with a different number of discriminators and α functions. We further report baseline results using original GANs as a grey plane. Higher values suggest more sample diversity over time.

We observe that using a bigger discriminator ensemble results in higher cumulative Intra FID across the different datasets and α functions. Not applying microbatch discrimination, *i.e.* $\alpha = 0$, leads to lower cumulative Intra FID suggesting the efficacy of our approach. Using α_{sigm}, α_{soft}, and α_{tanh} leads to similar results among the different datasets. Even though α_{id} presents higher values compared to the other functions, the generated set lacks sample quality, as previously discussed in Section 2.3.2.

Mean and minimum FID

We further calculated the mean and minimum FID obtained during 50K training iterations to measure the similarity between the real and generated sets. We used 10K samples from each of the sets to calculate the FID at every thousandth iteration. Results are presented in Table 2.5.

Table 2.5: Mean and minimum FID of different microbatchGAN configurations as well as the original GANs baseline. Lower values suggest higher similarity between the real and generated sets. Bold values represent the best mean and minimum FID for each dataset, while underline values represent the best α for each K.

microbatchGAN		MNIST		CIFAR-10		CelebA	
K	α	Mean FID	Min FID	Mean FID	Min FID	Mean FID	Min FID
1	-	50.9 ± 9.7	22.7 ± 0.7	125.5 ± 1.5	84.8 ± 1.6	77.3 ± 1.7	38.5 ± 1.1
2	α_{sigm}	37.6 ± 1.1	23.5 ± 3.0	111.9 ± 0.1	90.8 ± 0.6	76.3 ± 0.6	53.0 ± 2.6
2	α_{soft}	41.9 ± 1.2	24.6 ± 0.0	110.2 ± 0.9	90.6 ± 1.2	74.7 ± 2.9	49.5 ± 0.1
2	α_{tanh}	43.9 ± 0.8	27.2 ± 0.5	115.3 ± 0.5	91.3 ± 0.4	87.1 ± 2.4	54.7 ± 0.8
2	α_{id}	89.1 ± 2.2	53.6 ± 2.9	168.1 ± 2.0	113.2 ± 2.2	206.1 ± 3.5	113.6 ± 5.2
5	α_{sigm}	**34.7 ± 0.3**	20.1 ± 0.1	**103.9 ± 1.8**	81.4 ± 1.1	**66.5 ± 0.6**	40.4 ± 3.1
5	α_{soft}	37.2 ± 0.3	19.4 ± 0.1	106.4 ± 0.8	82.5 ± 1.2	69.1 ± 0.3	42.0 ± 2.0
5	α_{tanh}	39.4 ± 1.1	20.0 ± 0.1	107.2 ± 0.8	80.8 ± 0.6	70.3 ± 1.3	42.8 ± 0.5
5	α_{id}	61.2 ± 0.3	37.3 ± 0.2	127.9 ± 0.4	97.5 ± 2.8	135.9 ± 1.1	77.5 ± 2.0
10	α_{sigm}	38.9 ± 3.0	18.0 ± 0.1	110.2 ± 1.7	79.0 ± 0.7	68.4 ± 0.1	34.8 ± 1.2
10	α_{soft}	36.2 ± 0.9	**17.1 ± 0.2**	110.8 ± 0.4	79.2 ± 0.5	67.8 ± 2.6	**34.5 ± 0.2**
10	α_{tanh}	37.4 ± 1.2	17.4 ± 0.2	112.8 ± 1.7	**77.7 ± 0.6**	71.0 ± 1.4	34.5 ± 0.3
10	α_{id}	48.7 ± 0.9	28.7 ± 0.1	117.0 ± 0.2	87.1 ± 1.0	91.4 ± 0.2	45.4 ± 0.1

We observe that using a bigger discriminator ensemble as well as α_{tanh}, α_{soft}, or α_{sigm} leads to a better mean and minimum FID. We note that the higher FID observed with α_{id} indicate the lack of sample quality in the generated samples, highlighting the need for the proposed properties in α (Section 2.3.2) for the success of microbatch discrimination. The better mean and minimum FID observed with $K = 5$ and $K = 10$, respectively, may be explained by the training instability introduced by using a bigger discriminator set in the earlier training iterations.

2.3.4 Method Comparisons

We now compare different configurations of microbatchGAN to existing methods, including our previously proposed Dropout-GAN.

Comparison to Dropout-GAN on CIFAR-10

We start by comparing microbatchGAN to different GANs variations, namely original GANs, modified loss GANs, LSGAN, and DRAGAN on CIFAR-10. We also compared against the multi-adversarial dropout versions of these methods previously reported in Section 2.2.4, using the same architectural and training settings. For a fair comparison, we also use $K = 2$, similarly to the adversarial dropout variants. Comparison results are presented in Table 2.6.

Table 2.6: Method comparison in terms of minimum FID on CIFAR-10.

	CIFAR-10
GANs [Goo+14]	70.23
mod-GANs [Goo+14]	79.58
LSGAN [Mao+17]	83.66
DRAGAN [Kod+17]	80.57
GANs ($K = 2$)	74.07
mod-GANs ($K = 2$)	71.96
LSGAN ($K = 2$)	73.33
DRAGAN ($K = 2$)	75.83
Dropout-GANs ($K = 2, d = 0.5$)	66.82
Dropout-mod-GANs ($K = 2, d = 0.5$)	67.57
Dropout-LSGAN ($K = 2, d = 0.5$)	69.37
Dropout-DRAGAN ($K = 2, d = 0.5$)	66.90
microbatchGAN ($K = 2, \alpha = \alpha_{\text{sigm}}$)	66.93
microbatchGAN ($K = 2, \alpha = \alpha_{\text{soft}}$)	**65.54**
microbatchGAN ($K = 2, \alpha = \alpha_{\text{tanh}}$)	65.84

We observe that all microbatchGAN configurations improve the minimum FID of all compared methods. Even though all α functions show good performance, α_{soft} presents the best result.

Inception Scores on CIFAR-10, STL-10, and ImageNet

Considering a broader comparison against existing methods, we evaluated microbatchGAN on CIFAR-10, STL-10, ImageNet using Inception Score. We used $K = 2$, for a more honest comparison to the 1-discriminator methods, and followed the architecture and training procedure described in Section 2.2.4. Results are presented in Table 2.7.

Table 2.7: Inception scores of different unsupervised methods on CIFAR-10, STL-10, and ImageNet. The original GANs baseline, which shares identical architectural and training settings as microbatchGAN, is underlined.

	CIFAR-10	STL-10	ImageNet
Real data	11.24	26.08	25.78
WGAN [ACB17]	3.82	-	-
MIX+WGAN [Aro+17]	4.04	-	-
ALI [Dum+17]	5.34	-	-
BEGAN [BSM17]	5.62	-	-
MAGAN [Wan+17]	5.67	-	-
GMAN ($K = 2$) [DGM17]	5.87	-	-
GANs [Goo+14]	5.92	6.78	7.04
Dropout-GAN ($K = 2, d = 0.5$)	5.98	-	-
GMAN ($K = 5$) [DGM17]	6.00	-	-
Dropout-GAN ($K = 5, d = 0.5$)	6.05	-	-
DCGAN [RMC16]	6.40	7.54	7.89
Improved-GAN [Sal+16]	6.86	-	-
D2GAN [Ngu+17]	7.15	7.98	8.25
DFM [WB17]	7.72	8.51	9.18
MGAN [Hoa+18]	8.33	9.22	9.32
microbatchGAN ($K = 2, \alpha = \alpha_{\text{sigm}}$)	6.77	7.23	7.32
microbatchGAN ($K = 2, \alpha = \alpha_{\text{soft}}$)	6.66	7.19	7.40
microbatchGAN ($K = 2, \alpha = \alpha_{\text{tanh}}$)	6.61	7.07	7.40

We note that the compared methods, except for GMAN, Dropout-GAN, and the GANs baseline, were trained with different architectural designs and training settings. Namely, we notice the strong effects of bigger architectures on the increase of IS, especially in DCGAN. DFM and MGAN make use of additional

autoencoders or classifiers, with the latter framework making use of 10 generators in their reported results. We further acknowledge D2GAN's great results, which uses two discriminators with different objectives. However, we find that this may compromise the extendibility of their approach.

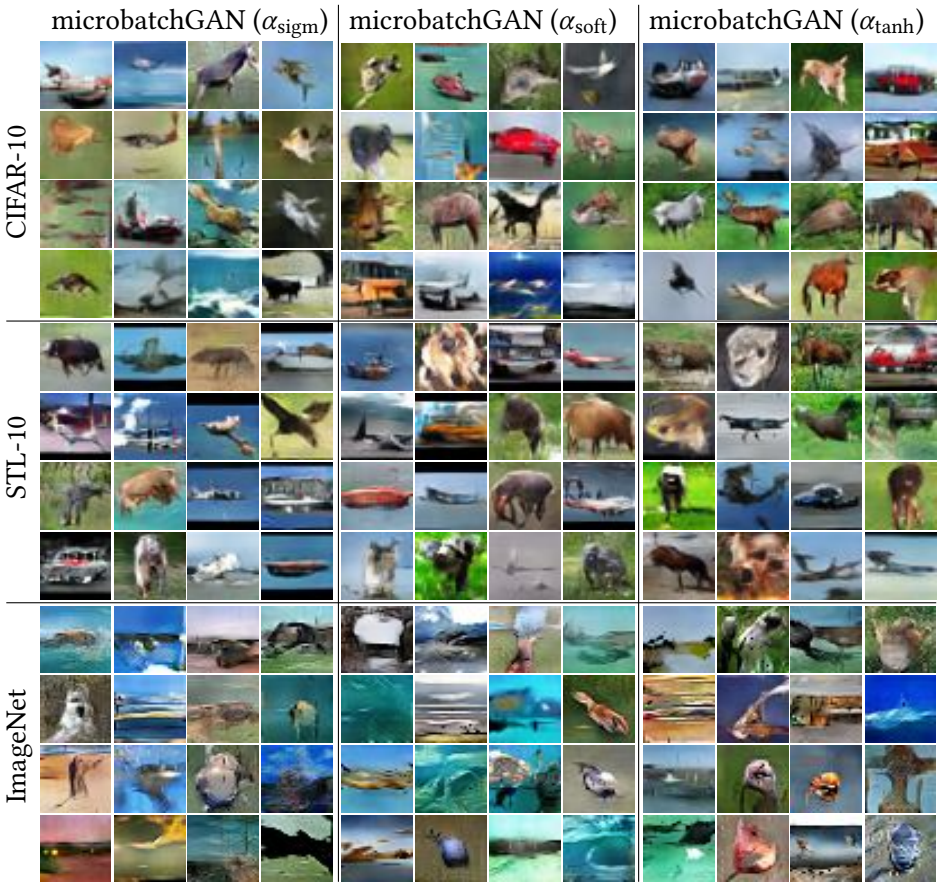


Figure 2.11: Generated samples from different microbatchGAN configurations ($K = 2$ and varying α) trained on CIFAR-10, STL-10, and ImageNet.

Hence, for a direct comparison, we should take a look at the aforementioned method exceptions. Specifically, we observe an increase in IS of approximately 15%, 7%, and 5% on CIFAR-10, STL-10, and ImageNet, respectively, of our approach compared to the original GANs baseline. Moreover, considering CIFAR-10,

all microbatchGAN configurations outperform the best GMAN and Dropout-GAN variants while using fewer discriminators, *i.e.* 2 instead of 5. Examples of generated samples are showcased in Figure 2.11. We observe sample variety and sample quality across the different datasets, suggesting the efficacy of multi-adversarial microbatch discrimination.

2.4 Concluding Remarks

In this Chapter, we proposed two novel multi-adversarial frameworks, Dropout-GAN and microbatchGAN, that help mitigate the mode collapse problem inherent to GANs. To this end, our frameworks leverage a multi-adversarial setting differently to ultimately stimulate sample diversity while maintaining sample quality in the generated set. Through empirical and theoretical discussions, we have shown the efficacy of the proposed frameworks on multiple GANs formulations and datasets. We note that Dropout-GAN has also been shown to improve sample diversity in the text domain in one of our follow-up publications [Sau+20].

A drawback of the proposed methods, however, is in their multi-adversarial nature. More specifically, both methods rely on the addition of discriminators to the original GANs framework, leading to an increase in the memory and computational costs. However, this may be attenuated by applying complementary compression techniques to the proposed frameworks both at training and inference time. We discuss and propose new mechanisms to compress neural networks in the next Section.

3 Compression of Neural Networks

This Chapter is based on two of our refereed papers: Monte Carlo Quantization [MVK19] and Monte Carlo Gradient Quantization [MVK20]. We note that some of the results presented for Monte Carlo Quantization are based on an extended arXiv version of the paper presented at NeurIPS 2019 Workshop on Energy Efficient Machine Learning and Cognitive Computing. For readers unfamiliar with compression of neural networks, we refer to Sze et al. [Sze+20].

In this Chapter, we introduce our novel compression methods to reduce the costs of pre-trained models after training as well as the communication costs during training in distributed settings. Reducing neural network costs by making them more efficient is an important endeavor considering their wide adoption in a broad range of real-world applications. Lowering the footprint of existing and future models may directly translate to a reduction in energy consumption, which not only benefits cloud applications but also opens the possibility of bringing such applications to edge devices. Such compact models may then be store and executed on-chip to improve inference latency and address certain privacy concerns.

We start by providing an overview of existing compression methods in Section 3.1. Then, we present our post-training compression method for weights and activations, Monte Carlo Quantization (MCQ), in Section 3.2. We then extend the previous method to gradient compression, *i.e.* Monte Carlo Gradient Quantization (MCGQ), in Section 3.3. Finally, we summarize our contributions and possible applications of the proposed methods in Section 3.4.

3.1 Related Work

We first discuss existing compression methods that focus mostly on weight and activation compression in Section 3.1.1. Then, we shift our focus to gradient compression methods in Section 3.1.2.

3.1.1 Compression of Weights and Activations

Pruning and quantization techniques may be applied to the weights and activations of neural networks to reduce their memory and computational costs. While pruning removes certain weights or activations that may be redundant [HMD16; LDS90; Moc+18], quantization reduces the precision required to represent weights and activations [Hub+16; Ras+16; Zho+16]. Quantization techniques may also induce sparsity which further enables efficient hardware implementations [Lin+16; VNM17]. We discuss popular quantization methods below.

The reduction of weight precision to binary representations was introduced in BinaryConnect [CBD15a] and further extended to activations by XNOR-Net [Ras+16] and BNN [Hub+16]. On the other hand, TWN [LZL16] proposed to increase the model expressiveness by using ternary weight representations instead. TTQ [Zhu+17] further improved such expressiveness by learning a positive and a negative weight scale during training. Stochastic parameterization was proposed by LR-Net [SLF18] to binarize and ternarize weights while a restriction to powers of two and zero on the weights was presented in INQ [Zho+17]. The categorization of weights in different groups with different scaling factors was presented in FGQ [Mel+17] to minimize the element-wise distance of the original and quantized weights. Hardware-aware quantization was also proposed through reinforcement learning based on a specific hardware's response [Wan+19a]. Compressed networks may also be jointly trained alongside quantizers [Zha+18a] or weights may be directly encoded by Bloomier filters [Rea+18].

The main drawback of the previous methods is the reliance on the additional training or fine-tuning of the compressed neural networks to achieve competitive performance relative to the full-precision counterparts. This may introduce unnecessary costs which may, in turn, have significant financial and environmental implications [SGM19]. On the other hand, our proposed method, Monte Carlo Quantization (MCQ), was designed to appropriately approximate the original weight and activation distributions without requiring retraining of the compressed model. To this end, we use Monte Carlo methods and importance sampling techniques, with the resulting network's complexity being proportional to the number of samples taken during the sampling process. We address other post-training compression techniques that do not rely on pruning or quantization, such as clipping [BNS19] and splitting [Zha+19a], in Chapter 5.

3.1.2 Compression of Gradients

Some of the previous techniques may also be applied to compress gradients on top of the weights and activations, during the backward pass [CBD15b; Gup+15; Zho+16]. Specifically, differentiable quantization schemes were proposed to enable training with discrete gradients [Lou+19] as well as training losses [Wu+18]. On the other hand, some methods solely perform gradient compression to reduce the communication costs in distributed settings where several workers need to synchronize their gradients. We will provide an overview of such methods below.

Focusing on the most extreme case of gradient quantization, 1-bit gradients were presented in 1-bit SGD [Sei+14] and signSGD [Ber+18], with an additional variant using also the sign of momentum being proposed by the latter method. Ternary gradient quantization was proposed by TernGrad [Wen+17] and QSGD [Ali+17], with the latter proposing a family of schemes accommodating 2, 4, and 8 bits. Storing information relative to accumulated errors in memory was proposed by Mem-SGD [SCJ18], with Ef-signSGD [Kar+19] introducing step-wise error feedback to improve the previous 1-bit gradient methods.

Regarding gradient pruning approaches, an even greater compression scheme may be achieved at high sparsity rates using encoded communication. A simple yet effective way of promoting gradient sparsity is to rely on thresholds [AH17; Lin+18a; Str15]. However, optimal threshold levels are likely network-dependent and may be hard to find in practice. To ease the relevance of the chosen threshold, Deep Gradient Compression (DGC) [Lin+18a] proposed to transmit only the accumulated gradients above a certain value. The pruned accumulated gradients would then eventually be sent as training progresses such that no information is lost. Instead of thresholding, one may perform fraction-based pruning instead [Dry+16]. Moreover, AdaComp [Che+18] proposed a dynamic pruning approach based on local gradient activity.

As suggested by previous work, high-level compression of gradients may be performed by applying quantization and pruning techniques. However, existing approaches mostly rely on either one or the other and do not leverage both techniques. To this end, we propose to use a modified version of the previously proposed method, MCQ, to also compress gradients during training. Our new method, called Monte Carlo Gradient Quantization (MCGQ), combines both sparsity and quantization to achieve high gradient compression rates, which may be controlled by the sampling amount.

3.2 Monte Carlo Quantization

We propose to use Monte Carlo methods and importance sampling to compress a pre-trained neural network without retraining. More specifically, we prune and quantize weights and activations by creating sparse, low-bit-width integer representations that approximate the full-precision weights and activations. Due to the sampling nature of our approach, called Monte Carlo Quantization (MCQ), the precision, sparsity, and complexity of our approach depends on the performed sampling amount. Our method is linear in space and time in the number of weights and activations and the resulting compressed models achieve minimal to no performance loss compared to their full-precision, dense counterparts.

We start by discussing the relation between neural networks and Monte Carlo methods in Section 3.2.1 and present MCQ as well as important implementation details in Section 3.2.2. Then, we present empirical evidence of MCQ's efficacy in Section 3.2.3 and compare against existing methods that do require additional training on a variety of models and datasets in Section 3.2.4.

3.2.1 Neural Networks and Monte Carlo methods

Monte Carlo methods and random sampling techniques have been widely applied to neural networks on several fronts. As popular examples we have: network initialization with random weights, network optimization through stochastic variants of gradient descent [RM51], random network regularization schemes such as Dropout [Sri+14] or DropConnect [Wan+13], random data augmentation and shuffling during training, random generative network inputs with GANs [Goo+14]. Particularly, stochastic rounding [Gup+15] has been proposed to optimize training, which differs from MCQ's focus on optimizing inference.

On a separate note, the ReLU [NH10] activation function has been widely used in many state-of-the-art networks since it was proposed. Considering our compression use case, we are particularly interested in exploiting the equivariance property of ReLU. We will be discussing how to leverage such property to normalize and, ultimately, quantize neural networks next.

Network normalization

The equivariance property of ReLU allows for an arbitrary scale and re-scale of weights and activations without affecting the output of a neural network.

Specifically, let us consider the weights $w_{l-1,i,j}$ going from the i-th neuron out of N_{l-1} neurons of layer $l-1$ to the j-th neuron out of N_l neurons in layer l, with $i \in \{0, \dots, N_{l-1}-1\}$ and $j \in \{0, \dots, N_l - 1\}$. Representing the j-th activation in the l-th layer as $a_{l,j}$ and a scaling factor $f \in \mathbb{R}^+$, we have $a_{l,j}$ equal to:

$$max\left\{0, \sum_{i=0}^{N_{l-1}-1} w_{l-1,i,j} a_{l-1,i} + b_{l-1,j}\right\} = f \cdot max\left\{0, \frac{\sum_{i=0}^{N_{l-1}-1} w_{l-1,i,j} a_{l-1,i} + b_{l-1,j}}{f}\right\}. \tag{3.1}$$

To enable the treatment of the weights of a given neuron j in layer l as a probability distributions over all its connections, one may normalize the weights using the following (non-zero) scaling factor:

$$f = \|\mathbf{w}_{l-1,j}\|_1 = \sum_{i=0}^{N_{l-1}-1} |w_{l-1,i,j}|. \tag{3.2}$$

A similar procedure may be applied to normalize the activations $a_{l,j}$ of layer l. Assuming an exclusive use of ReLU on all hidden layers, one may propagate the different scaling factors throughout the entire network. This ultimately enables the usage of integer weights and activations, without the expensive computational cost of re-scaling to full-precision representations at every layer.

Network quantization

Our approach relies on leveraging a normalized network to simulate discrete probability densities by constructing a probability density function (PDF) and sample from the corresponding cumulative distribution function (CDF). The number of hits per weight or activation after this importance sampling process is then their respective integer representation. To ease explanations, we focus solely on weight quantization for now. However, as previously mentioned, the following processes may also be used to quantize activations at inference time.

Given n weights w_k, with $k \in \{0, \dots, n-1\}$ and assuming $\sum_{k=0}^{n-1} |w_k| = \|w\|_1 = 1$ without loss of generality, let us define a unit interval partition $P_m := \sum_{k=1}^{m} |w_k|$:

$$0 = P_0 \xrightarrow{|w_1|} P_1 \xrightarrow{|w_2|} P_2 \cdots P_{n-2} \xrightarrow{|w_{n-1}|} P_{n-1} = 1. \tag{3.3}$$

We may then approximate the original weight distribution by using N uniformly distributed samples $x_i \in [0, 1)$:

$$\sum_{j=0}^{n-1} w_j a_j \approx \frac{1}{N} \sum_{i=0}^{N-1} \underbrace{\text{sign}(w_{j_i})}_{\in\{-1,0,1\}} \times a_{j_i}, \tag{3.4}$$

where $j_i \in \{0, \ldots, n-1\}$ is uniquely determined by $P_{j_i-1} \leq x_i < P_{j_i}$.

We improve this importance sampling process by using *jittered equidistant sampling* with a random offset to better approximate the weight approximation. In particular, given a random variable $\xi \in [0, 1)$, we generate N uniformly distributed samples $x_i \in [0, 1)$ such that $x_i = \dfrac{i + \xi}{N}$, where $i \in \{0, \ldots, N-1\}$.

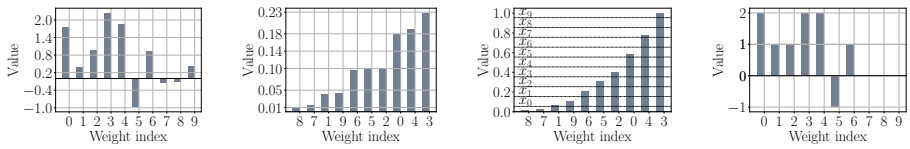
3.2.2 Implementation Details

Our method, MCQ, follows the previous network normalization and network quantization principles to compress a pre-trained neural network. Specifically, we apply the following steps to quantize the $N_{l,w}$ weights of a given layer l:

(1) Create a PDF, such that $\sum_{k=0}^{N_{l,w}-1} |w_k| = 1$.

(2) Perform importance sampling based on weight magnitudes by sampling from the corresponding CDF.

(3) Represent each weight by its hit count at the end of the sampling process.



(a) FP weights. **(b)** Sorted PDF. **(c)** Sampling on CDF. **(d)** Integer weights.

Figure 3.1: Starting from full-precision, FP, weights (a), we create a PDF of the sorted absolute values (b) and uniformly sample from the corresponding CDF (c). The sampling process produces quantized integer network weights based on the number of hits per weight (d). Note that since weights 7, 8, and 9 were not hit, they may be pruned.

Algorithm 3 Monte Carlo Quantization.

Input: Pre-trained full-precision network, L trainable layers, K ratio of samples to be sampled per weight
Output: Quantized network with integer weights
for K=0 to L-1 **do**
  $unsorted_{idxs} \leftarrow argsort(W_K)$
  $W_{sorted} \leftarrow sort(W_K)$
  $W_{abs} \leftarrow abs(W_{sorted})$
  $W_{PDF} \leftarrow \frac{W_{abs}}{\|W_K\|_1}$
  $W_{CDF} \leftarrow \sum_{i=1}^{|W_{PDF}|} W_{PDF_i}$
  $N \leftarrow ceil(|W_K| * K)$
  $start_{idx} \leftarrow 0$
  $\xi \leftarrow random(0, 1)$
  $W'_K \leftarrow [0] \times |W_K|$
  `// Start sampling`
  for i=0 to $N - 1$ **do**
    $x_i \leftarrow \frac{i + \xi}{N}$
    $hit_{idx} \leftarrow argmax(W_{CDF}[start_{idx} :] \geq x_i) + start_{idx}$
    $start_{idx} \leftarrow hit_{idx}$
    $unsorted_{idx} \leftarrow unsorted_{idxs}[hit_{idx}]$
    `// Update counter`
    if $W_K[unsorted_{idx}] > 0$ **then**
      $W'_K[unsorted_{idx}]$++
    else
      $W'_K[unsorted_{idx}]$--
    end if
  end for
  `// Update to integer weights`
  $W_K \leftarrow W'_K$
end for

Figure 3.1 illustrates our method's steps for a layer with 10 weights and 10 samples. A more detailed overview of our method is presented in Algorithm 3. We refer the reader to NumPy's documentation [Har+20] for additional information about the used functions.

Layer normalization

In Section 3.2.1, we introduced normalization as a neuron-wise procedure. In practice, we observed that this required big sample size to achieve a good approximation, especially if the number of incoming weights is limited. Instead, we propose to perform such normalization in a layer-wise manner. This represents a broader sampling approach, which allows the redistribution of hit counts from low-importance weights from a given neuron to high-importance weights from another neuron. As previously mentioned, importance is measured by the weight magnitude in our use case.

Overall, we observed that the layer-wise approach promotes a better overall approximation of the original weight distribution. In this layer-wise variant, our scaling factor f is the 1-norm of all weights from a given layer $l-1$ to the subsequent layer l. Hence, each normalized weight is now a probability with respect to all connections in a given layer, instead of a given neuron. Moreover, this results in the storage of only one floating-point scale per layer, instead of per neuron. Since we do not quantize the gradients in our experiments, the efficient low-precision integer multiply-accumulate (MAC) operations in each layer are re-scaled by $\frac{f}{N}$ before adding the biases.

Importance sampling

Each iteration of our importance sampling process results in a ternary representation for each weight, representing a positive or negative hit, depending on the weight's sign, or no-hit. We use this information to count the total number of hits across all weights at the end of sampling. Since each weight may be hit multiple times across all iterations, the final representation for each weight is likely to not be ternary, depending on the weight's magnitude. As previously mentioned in Section 3.2.1, we avoid the binary search cost otherwise induced with fully random sampling by using a jittered equidistant sampling strategy. Hence, since we only need to iterate through the CDF once, our sampling algorithm is linear in space and time in the number of weights (or activations) at inference time.

We use $N = K \times N_{values}$ samples, where $K \in \mathbb{R}^+$ is a hyper-parameter (not related to the number of discriminators from here on) and N_{values} is the number of values, *i.e.* weights or activations, to sample from. Hence, K directly affects the quality of the approximation, presenting a trade-off between precision and approximation quality, which directly correlates to performance. In other words, while bigger K values promote a better approximation at increased sampling costs, smaller K values sacrifice approximation but make the sampling process more computationally efficient. Even though a different K may be used for different layers, depending on the number of weights or activations, we used the same K for all layers in our experiments.

To promote better approximation, sorting mechanisms applied to Monte Carlo schemes have been proposed in the past [LEc+18; LLT08]. In our use case, sorting the full-precision distributions before creating the PDF results in grouping smaller magnitude values together. In combination with the proposed uniform sampling strategy, this is likely to result in such values being sampled less often, which may result in higher sparsity and a better approximation of the higher-magnitude values. In practice, we observed that sorting the full-precision values before starting the sampling process helped reduce the performance loss of quantizing smaller layers.

Layer quantization

The bit-width of a given layer l is related to the number of bits $B_{W_l} \in \mathbb{N}$ required to represent the highest integer value in its $N_{l,w}$ quantized weights, $Q(w_{l,i})$, including its sign:

$$B_{W_l} = 2 + \left\lfloor \log_2 \left(\max_{0 \le i \le N_{l,w}-1} |Q(w_{l,i})| \right) \right\rfloor. \tag{3.5}$$

We note that, alternatively, separating positive and negative weights into different sets would remove the additional bit used to represent the sign. However, we do not consider this when calculating the bit-widths of each layer.

Online quantization

The previous discussions may be also applied to quantize activations online, *i.e.* at inference time. This differs from weight quantization, which is performed offline, *i.e.* after training but before inference. Hence, considering a layer l, its

$N_{l,a}$ activations are treated as a probability distribution over layer l's output features, such that $\sum_{j=0}^{N_{l,a}-1} |a_{l,j}| = 1$. Since weights and activations are sampled independently, a different K may be used for each of the processes. However, we used the same K to quantize both the weights and activations in our experiments. Moreover, the bit-widths B_{A_l} for the quantized activations $Q(a_{l,j})$, with $j \in \{0, \ldots, N_{l,a} - 1\}$, may be calculated by adapting (3.5), discarding the additional bit sign if ReLU is employed throughout the entire network.

3.2.3 Experimental results

We will first study the effects of using varying amounts of sampling on the performance of the quantized models. We used several pre-trained, full-precision models trained on CIFAR-10, SVHN [Net+11], and ImageNet. Since we use the same K on all layers, some layers may be quantized to different bit-widths, depending on the original weight and activation distributions. Hence, we simply report the average bit-widths across all quantized layers of each network. Moreover, we also indicate the average sparsity percentage of the quantized weights and activations.

Quantizing the first layer of pre-trained models has been shown to increase the performance gap between the compressed networks and their full-precision counterparts [HMD16; LZL16; Zho+16]. With this in mind, we present results with and without quantizing the initial layer. We further note that we do not quantize batch normalization layers since their transformations may be incorporated into the network weights at inference time [Wu+18]. We describe the architectural and training details of the baseline pre-trained models next.

CIFAR-10

We trained VGG-7 [LD15], VGG-14, and ResNet-20 [He+16] baseline models on CIFAR-10 using a popular training regime [1]. More specifically, all models were trained for 300 epochs using the Adam optimizer [KB15] with a decreasing learning rate schedule and weight decay. The ResNet-20 baseline's architecture follows the original residual network configuration [He+16], with 64, 128, and 256 filters in the residual blocks. The quantized model performances over different sampling amounts are presented in Figure 3.2.

1 https://github.com/bearpaw/pytorch-classification

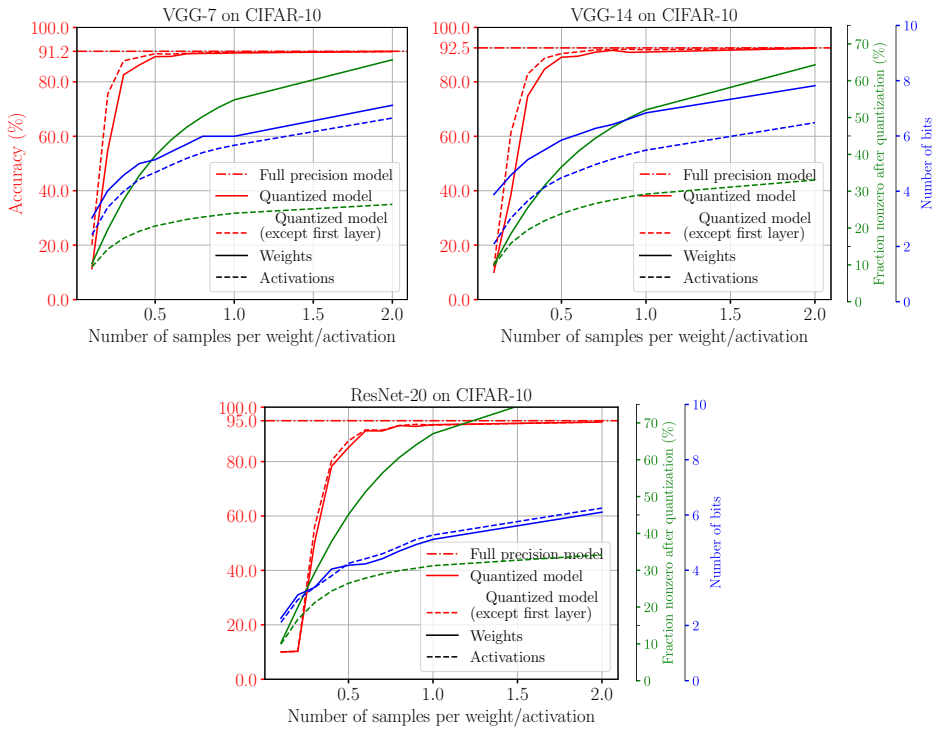


Figure 3.2: Quantized weights and activations on CIFAR-10 models.

We observe that all compressed models quickly reach their respective baseline performance at low bit-widths and high sparsity levels. More specifically, the different quantized models reach baseline accuracy at around $K = 0.5$ while pruning approximately 50% of the weights and approximately 70 to 80% of the activations.

SVHN

We further evaluated using different sampling amounts to compress popular SVHN models previously studied in related methods. Specifically, the baseline VGG-7 model presented by Courbariaux et al. [CBD15a] and Hubara et al. [Hub+16] was trained for 164 epochs using Adam optimizer and a decaying learning rate with weight decay. Moreover, using the previous training set-

tings for 200 epochs, we trained four additional baselines presented by Zhou et al. [Zho+16]: Model A represents a popular SVHN architectural setting [2] and Model D adopts Model A's architecture with a 87.5% reduction in the number of channels. Compression results for the different models are presented in Figure 3.3.

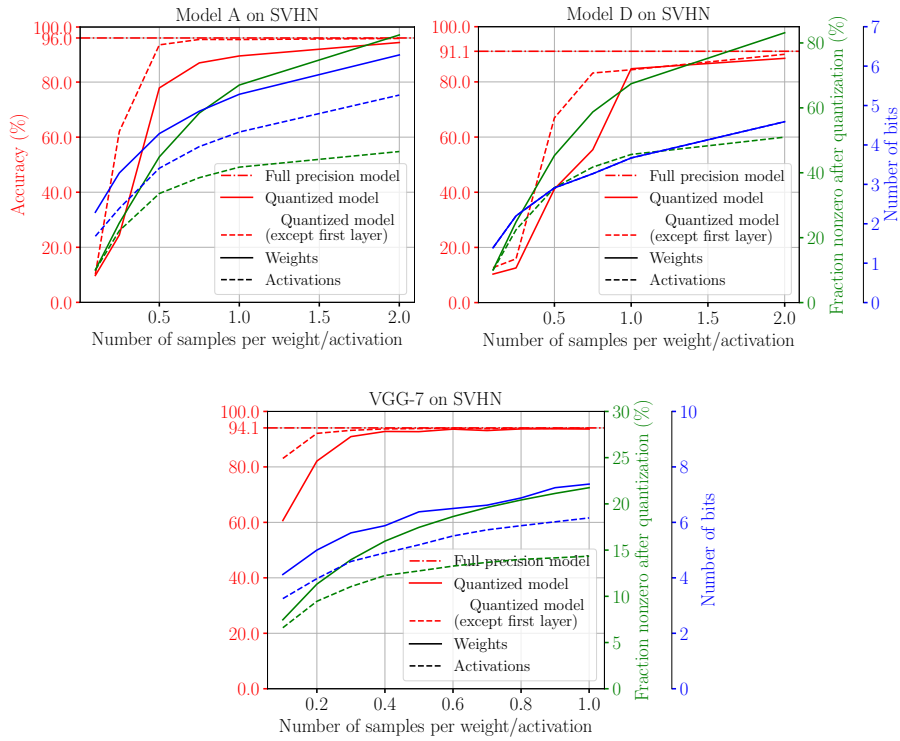


Figure 3.3: Quantized weights and activations on different SVHN models.

Compared to CIFAR-10, we observe that fewer samples per weight or activation are required to reach close to baseline performance. This is likely due to the bigger architectures used, which may increase redundancy and allow for a better approximation of more relevant values according to their magnitude. Moreover, the smaller variant (Model D) requires a higher K to achieve base-

2 https://github.com/aaron-xichen/pytorch-playground

line performance, reflecting the effects of model size in the performance of the compressed models.

ImageNet

We also experimented on ImageNet by using several pre-trained models from Pytorch [Pas+19]'s model zoo [3]. Namely, we applied MCQ on AlexNet, ResNet-18, and ResNet-50. We refer to their documentation for training details of the different models. Figure 3.4 presents the compression results using different sampling amounts.

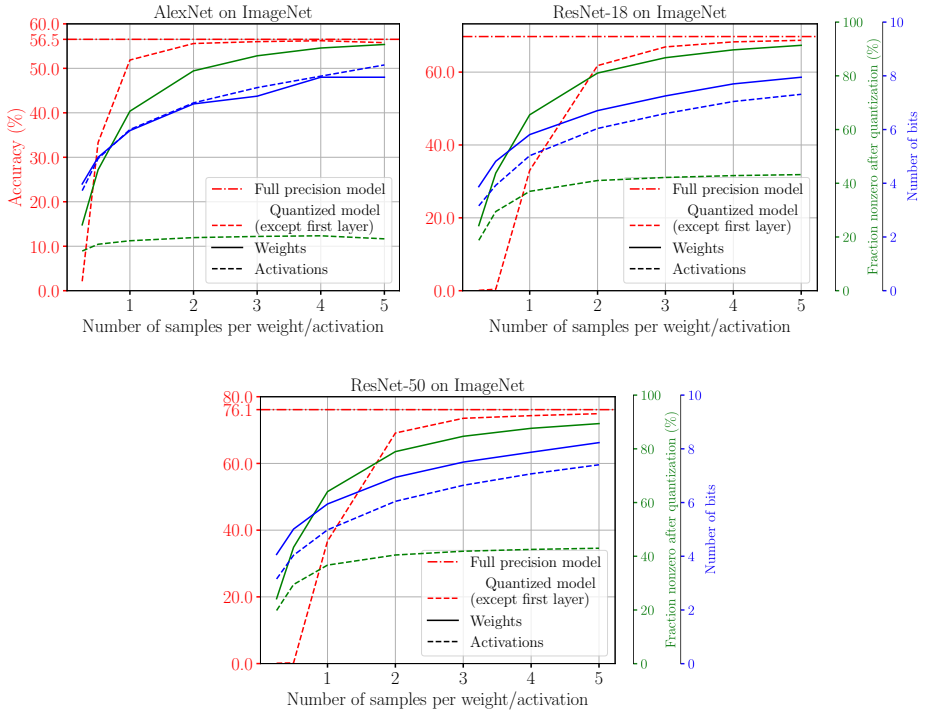


Figure 3.4: Quantized weights and activations on ImageNet models.

Compared to both CIFAR-10 and SVHN, we observe that compressed ImageNet models require additional sampling to reduce performance loss. This may be

3 https://pytorch.org/serve/model_zoo.html

attributed to the more challenging task at the end. The additional sampling results in a higher bit-width and less sparsity on weights and activations when compared to the other datasets. We note that we applied no sorting on ImageNet since sorting showed to be more beneficial when using a lower sampling amount.

3.2.4 Method Comparisons

We now proceed to evaluate MCQ against existing compression methods on the previous datasets. For our compressed models, we indicate the average bit-widths across all quantized layers: "8w-32a" means that the average weight bit-width is 8 bits and the average activation bit-width across all layers is 32 bits, for example. To describe special instances where some layers are not quantized by some of the compared works, we make use of footnotes [4,5,6]. For a fair comparison, we report the accuracy difference between each quantized model and its respective full-precision baseline.

CIFAR-10

We compare against several baselines using the same models on CIFAR-10. To achieve this, we report the baseline results for BNN [Hub+16] and XNOR-Net [Ras+16] presented in BC [CBD15a], since they make use of the same model. Moreover, since we report the performance difference instead of the absolute performance of the quantized models, we present the results of the baselines of BWN [Ras+16] reported in TWN [LZL16] since their original work does not mention the performance of their original baseline. The accuracy differences used for all the other compared methods were the ones reported in their respective works. Importantly, we note that all compared methods use additional training of the compressed models, unlike MCQ. Results are presented in Table 3.1.

We observe that MCQ achieves competitive results, even outperforming some methods that use additional training. Moreover, quantizing weights leads to higher performance loss related to the full-precision (FP) baseline than activations, leading to a decrease of approximately 1.0% accuracy, in the worst case. Quantizing activations on top of weights does not show a negative impact for the VGG-based models but reduces ResNet-20's accuracy by approximately 1.0%.

4 Not quantizing weights in the first layer.
5 Not quantizing weights in the last layer.
6 Using higher precision (8w-8a) for the first layer.

Table 3.1: Comparison results of different compression methods on CIFAR-10. We quantize weights (w), activations (a), or both (w+a), using $K = 1$.

Method	VGG-7	VGG-14	ResNet-20
FP baseline (32w-32a)	91.23	92.49	95.02
Δ MCQ (w)	-0.48 (6.1w-32a)	-1.04 (6.7w-32a)	-0.84 (5.1w-32a)
	+0.04[4] (6.1w-32a)	-0.50[4] (6.8w-32a)	-0.54[4] (5.1w-32a)
Δ MCQ (a)	-0.12[4] (32w-5.68a)	-0.06[4] (32w-5.51a)	-0.28[4] (32w-6.3a)
Δ MCQ (w + a)	-0.58 (6.1w-5.6a)	-1.08 (6.6w-5.3a)	-1.77 (5.1w-5.3a)
	-0.13[4] (6.1w-5.6a)	-0.54[4] (6.8w-5.5a)	-1.21[4] (5.1w-5.3a)
Δ TTQ [Zhu+17] (2w-32a)	-	-	-0.64[4]
Δ dLAC [VNM17] (2w-32a)	-	-3.0 / -1.4[4]	-
Δ TWNs [LZL16] (2w-32a)	-0.06	-	-
Δ BC [CBD15a] (1w-32a)	+0.74	-	-
Δ BNN [Hub+16] (1w-1a)	+0.49[4]	-	-
Δ BWN [Ras+16] (1w-32a)	-0.36 / +0.76[4]	-	-
Δ XNOR-Net [Ras+16] (1w-1a)	+0.47[4]	-	-
Δ RQ [Lou+19] (8w-8a)	+0.25	-	-
Δ LR-net [SLF18] (2w-32a)	-0.11[5]	-	-

Overall, accuracies tend to drop by approximately 0.5% when quantizing the first layer across the different models.

SVHN

We perform a similar comparison study on SVHN against 1-bit weight compression methods. Results are presented in Table 3.2.

Table 3.2: Comparison results of different 1-bit weight compression methods on SVHN. We quantize weights (w), activations (a), or both (w+a), using $K = 1$.

Method	VGG-7	Model A	Model D
FP baseline (32w-32a)	94.06	96.01	91.08
Δ MCQ (w)	-0.30 (7.3w-32a) / -0.02[4] (7.0w-32a)	-0.20[4] (5.1w-32a)	-2.17[4] (4.1w-32a)
Δ MCQ (a)	-0.04 (32w-7.15a)	+0.01[4] (32w-5.28a)	-0.11[4] (32w-4.58a)
Δ MCQ (w + a)	-0.32 (7.2w-6.0a) / -0.06[4] (7.0w-5.5a)	-0.40[4] (5.1w-4.2a)	-3.72[4] (4.1w-3.7a)
Δ DoReFa [Zho+16] (1w-1a)	-	-0.4[4,5]	-10.9[4,5]
Δ BC [CBD15a] (1w-32a)	+0.14	-	-
Δ BNN [Hub+16] (1w-1a)	-0.09[4]	-	-

We observe a minimal performance loss relative to the FP baselines, especially

on bigger models (VGG-7 and Model A). As previously discussed in Section 3.2.3, accuracy is more affected on smaller models (Model D). However, we note that we only use around 4 bits per weight and activations on Model D, so additional sampling may be performed to improve the approximation while still retaining a low bit-width. Overall, quantizing only the activations shows a neglecting effect on performance.

ImageNet

We conclude our experimental studies by performing additional comparisons on ImageNet. We note that the accuracy differences for DoReFa [Zho+16], BWN [Ras+16], TWN [LZL16] were calculated using the results reported by TTQ [Zhu+17]. Comparison results are presented in Table 3.3.

Table 3.3: Comparison results of different compression methods on ImageNet. We quantize weights (w), activations (a), or both (w+a), using $K = 5$.

Method	AlexNet	ResNet-18	ResNet-50
FP baseline (32w-32a)	56.52	69.76	76.13
Δ MCQ (w)	-0.99 (8.00w-32a)	-0.72 (8.00w-32a)	-0.73 (8.28w-32a)
	-0.68[4] (8.00w-32a)	-0.63[4] (8.00w-32a)	-0.20[4] (8.28w-32a)
Δ MCQ (a)	+0.02[4] (32w-8.36a)	-0.58[4] (32w-7.36a)	-0.76[4] (32w-7.45a)
Δ MCQ (w + a)	-1.05 (7.88w-8.46a)	-1.13 (8.00w-7.35a)	-1.64 (8.26w-7.43a)
	-0.75[4] (8.00w-7.2a)	-1.03[4] (8.00w-7.36a)	-1.21[4] (8.28w-7.45a)
Δ FGQ [Mel+17] (2w-8a)	-7.79[6]	-	-4.29
Δ TTQ [Zhu+17] (2w-32a)	+0.3[4,5]	-3.0[4,5]	-
Δ TWNs [LZL16] (2w-32a)	-2.7[4,5]	-4.3[4,5]	-
Δ BWN [Ras+16] (1w-32a)	+0.2	-8.5[4,5]	-
Δ XNOR-Net [Ras+16] (1w-1a)	-12.4	-18.1[4,5]	-
Δ DoReFa [Zho+16] (1w-32a)	-3.3[4,5]	-	-
Δ INQ [Zho+17] (5w-32a)	-0.15	-0.71	-1.59
Δ RQ [Lou+19] (8w-8a)	-	+0.43	-
Δ LR-net [SLF18] (2w-32a)	-	-6.07[4]	-

We observe that quantizing weights to 8-bits results in an accuracy drop of less than 1% on all models. Quantizing both weights and activations results in an additional 0.6% accuracy drop, in the worst case. Quantizing only the activations seems to affect the performance of the residual models but not AlexNet. Overall, MCQ shows competitive results against existing methods that require additional training of the compressed variants to achieve close to baseline performance.

3.3 Monte Carlo Gradient Quantization

We now shift our focus to in-training compression. To this end, we propose to apply an extended version of MCQ to quantize gradients during training. This new approach, called Monte Carlo Gradient Quantization (MCGQ), mainly tackles the communication cost required to synchronize multiple workers in distributed settings. Such communication creates a relevant overhead, especially as neural network architectures grow larger and become more complex. Hence, our novel method improves the training of large networks at scale by compressing and encoding the gradient information exchanged by different workers.

We detail how the previous method, MCQ, may be applied to gradient compression in Section 3.3.1. Moreover, we propose important modifications to MCQ that enable extreme gradient compression schemes in Section 3.3.2. We assess different configurations of MCGQ in Section 3.3.3, and compare against existing gradient compression methods on several tasks, models, and optimizers in Section 3.3.4.

3.3.1 Learning with Quantized Gradients

We start by discussing how to apply gradient quantization and pruning schemes to train neural networks using stochastic gradient descent (SGD) [RM51]. Considering a loss function $f : \mathbb{R}^d \to \mathbb{R}$, SGD minimizes f with relation to a parameter x at iteration t by using a learning rate γ to update x as follows: $x_{t+1} := x_t - \gamma g_{x_t}$, where g_{x_t} is the stochastic gradient such that $\mathbb{E}[g_{x_t}] = \nabla_{x_t} f$.

MCQ may be directly applied to gradients by first computing their floating-point values for all layers. Then, layer by layer, we may create an integer distribution that approximates the original full-precision distribution. By normalizing all gradients of a given layer l by their l_1-norm, $\|g_l\|_1 = \sum_{k=0}^{n-1} |g_{l,k}| = 1$, we may represent the normalized gradients as a PDF. Then, we may construct the respective CDF and define a unit interval partition $P_m := \sum_{k=0}^{m-1} |g_{l,k}|$:

$$0 = \overbrace{P_0 \quad P_1 \qquad P_2 \qquad\qquad P_{m-2}\ P_{m-1}}^{|g_{l,0}| \quad |g_{l,1}| \qquad\qquad |g_{l,n-1}|} = 1. \tag{3.6}$$

Using a similar importance sampling procedure as in MCQ, we may approximate the full-precision gradients with integer representations. Specifically, using jittered equidistant sampling with N uniformly distributed samples x_i in $[0, 1)$,

as described in Section 3.2.1, each gradient $g_{l,k}$ may be represented by its hit counts during the importance sampling process. Namely, considering a sample x_i, $g_{l,k_i} \in \{-1, 0, 1\}$ is set to either 0, if x_i is not in the interval of $g_{l,k}$, or ± 1, if $P_k \leq x_i < P_{k+1}$, depending on the original gradient $g_{l,k}$'s sign. After N sampling iterations, $g_{l,k}$ may then be approximated by:

$$g_{l,k} \approx \frac{1}{N} \sum_{i=0}^{N-1} g_{l,k_i}. \tag{3.7}$$

A simplified adaptation of MCQ's sampling procedure (Algorithm 3) to gradient quantization is presented in Algorithm 4. We briefly note that the function "cumsum" represents a cumulative sum; additional function details may be found in NumPy's documentation [Har+20].

Algorithm 4 Monte Carlo Gradient Quantization.

Input: gradients of layer l g_l, sampling amount K
Output: integer gradients $g_{l_{int}}$
$\xi \leftarrow \text{random}(0, 1)$
$N \leftarrow \lceil len(g_l) \times K \rceil$
$g_{l_{int}} \leftarrow [0] \times len(g_l)$
$start_{idx} \leftarrow 0$
$g_{l_{PDF}} \leftarrow \frac{|g_l|}{\|g_l\|_1}$
$g_{l_{CDF}} \leftarrow \text{cumsum}(g_{l_{PDF}})$
`// perform importance sampling`
for $i = 0, \ldots, N-1$ **do**
  $x_i \leftarrow \frac{\xi + i}{N}$
  `// find hit index`
  $hit_{idx} \leftarrow \text{argmax}(g_{l_{CDF}}[start_{idx}:] | g_{l_{CDF}} \geq x_i)$
  $start_{idx} \leftarrow hit_{idx}$
  `// count according to the original sign`
  $g_{l_{int}}[hit_{idx}] \leftarrow g_{l_{int}}[hit_{idx}] + sign(g_l[hit_{idx}])$
end for

Similarly to MCQ, we describe the number of samples as N, which may be controlled by adjusting K. Quantized gradients of a given layer l are described

as $g_{l_{int}}$. We point special attention to the monotonically increasing sampling from the CDF by keeping track of the index of the previously hit gradient during the last sampling iteration. Hence, similarly to the quantization of weights and activations with MCQ, quantizing gradients only requires a single pass over the CDF and is linear on the number of gradients to quantize: $\mathcal{O}(len(g_l))$. Contrarily to MCQ, we do not sort the full-precision values before constructing the PDF, since we found that this noise reduction step was not relevant to gradients and the respective convergence of the trained neural networks in practice [Nee+15].

We summarize the previous discussions in Algorithm 5. We note that the re-scaling of gradients is not necessary if using a scale-invariant optimizer, such as Adam [KB15] or Lazy Adam, which further allows for efficient sparse gradient updates. On the other hand, if using standard SGD, which is sensitive to gradient scale, the re-scaling of the quantized gradients $g_{l_{int}}$ by $\frac{\|g_l\|_1}{len(g_l)\times K}$ is necessary before performing parameter updates.

Algorithm 5 Gradient quantization.

Input: learning rate δ, n parameters of layer l x_l, sampling amount K
$g_l \leftarrow [0] \times n$
`// compute gradients`
for $i = 0, \ldots, n-1$ **do**
  $g_l[i] \leftarrow \text{StochasticGradient}(x_{l_i})$
end for
`// quantize and prune`
$g_{l_{int}} \leftarrow \text{MCGQ}(g_l, K)$
`// update parameters`
$x_l \leftarrow x_l - \delta g_{l_{int}}$

So far, we have only considered traditional neural network training in a single node. Before we discuss distributed training scenarios, we would like to note that MCGQ may also be used to reduce the computational cost of parameter updates in a single-worker environment by skipping parameter updates with sparse gradients.

3.3.2 Implementation Details

We will now go over several variants of MCGQ, which present new modifications to the original MCQ algorithm presented in Section 3.2. Namely, we introduce dynamic sampling, by learning an optimal K for each layer. Moreover, we propose a novel variant of MCGQ that samples proportionally to the magnitude of the accumulated gradients, instead of the gradient values calculated at each training iteration. Finally, we present important discussions regarding distributed training as well as the proposed encoded communication scheme.

Dynamic sampling

One drawback of MCQ is in K's grid-search for the sampling amounts that achieve a good performance and compression trade-off. Moreover, since different layers are likely to require different sparsity and quantization levels, such search may be even more costly if performed at each layer. To this end, we propose to learn an optimal K for each layer during training. Ideally, each layer learns a low sampling amount that approximates its floating-point gradient distribution up to a sufficient degree. We update K using the first Wasserstein distance [Val74] between the quantized and the full-precision gradient distributions, which has been shown to be correlated with quantization errors [Kre11].

In our use case, the first Wasserstein distance calculates the minimum cost required to transform a floating-point probability distribution $\mathbb{P}_f$ into a quantized probability distribution $\mathbb{P}_q$, using a transformation cost function γ. Specifically, the joint distribution of the two random variables X and Y represents the mass transfer space, with $\Pi(\mathbb{P}_f, \mathbb{P}_q)$ being the set of all X and Y joint distributions:

$$\text{Wasserstein}(\mathbb{P}_f, \mathbb{P}_q) = \inf_{\gamma \in \Pi(\mathbb{P}_f, \mathbb{P}_q)} \mathbb{E}_{(X,Y)\sim\gamma}[||X - Y||]. \tag{3.8}$$

Our dynamic sampling procedure is presented in Algorithm 6. At a given iteration, we first compute the first Wasserstein distance between the full-precision and integer PDFs. Then, we calculate the difference between the such distance and the distance computed in the previous iteration. Finally, considering a learning rate δ_K, we update the K to be used in the next iteration accordingly.

Algorithm 6 Dynamic sampling.

Input: learning rate δ_K, layer l's full-precision and quantized gradients g_l and $g_{l_{int}}$, respectively, and the Wasserstein distance d' and a sampling amount K from a previous iteration

`// compute distance between PDFs`

$d \leftarrow \text{Wasserstein}(\frac{g_l}{\|g_l\|_1}, \frac{g_{l_{int}}}{\|g_{l_{int}}\|_1})$

`// calculate distance difference`

$\Delta_d = d' - d$

`// update K`

$K \leftarrow K + \delta_K \times \Delta_d$

Local gradient accumulation

Sampling proportionally to the gradient magnitude at a given iteration, as proposed so far, may result in certain parameters with consistently small gradients rarely being updated throughout training. In such instances, gradients that are not hit during the importance sampling process will be lost which may compromise training convergence later on. To mitigate this information loss, we propose a variation of MCGQ that samples proportionally to the accumulated gradients instead. In a given iteration, the magnitude of non-sampled gradients is stored independently in a cumulative fashion. On the other hand, the stored information of sampled gradients is reset. Overall, this increases the likelihood of small gradients eventually being sampled in future training iterations.

The usage of accumulated gradients was first proposed by Deep Gradient Compression or DGC [Lin+18a], with small gradient parameters being updated at a lower frequency, equivalently to having a dynamic batch size per parameter. In MCGQ, sampling according to the accumulated gradients enables fewer sampling amounts, ultimately promoting higher sparsity and lower bit-width in the quantized gradients per iteration.

Our accumulated gradient version of MCGQ is presented in Algorithm 7. At a given iteration, after computing all gradients using stochastic descent, we accumulate them with their respective stored values from earlier iterations. Then, we quantize and prune the accumulated gradient information using MCGQ and update the parameters appropriately. (Note that, as previously discussed, we omit gradient re-scaling by assuming a scale-insensitive optimizer.) Finally, we

reset the accumulated gradient values of the updated parameters for the next iteration.

Algorithm 7 Gradient quantization with accumulated gradients.

Input: learning rate δ, n parameters of layer l x_l, accumulated gradients of layer l ag_l, sampling amount K
// compute gradients
for $i = 0, \ldots, n-1$ **do**
  $g_l[i] \leftarrow \text{StochasticGradient}(x_{l_i})$
end for
// accumulate gradients
$ag_l \leftarrow ag_l + g_l$
// quantize and prune
$g_{l_{int}} \leftarrow \text{MCGQ}(ag_l, K)$
// update parameters
$x_l \leftarrow x_l - \delta g_{l_{int}}$
// reset used accumulated gradients
$ag_l[g_{l_{int}} \neq 0] \leftarrow 0$

Distributed settings

So far, we have described MCGQ from the perspective of a single worker. However, MCGQ may be simply applied to distributed settings among several workers. More specifically, after each worker independently performs the gradient quantization steps proposed so far, the quantized gradients may be exchanged and merged across all workers by using an all-reduce operation [GLS99]. Then, each worker updates its locally stored parameter copies using the total gradient average across workers. In the accumulated gradients variant, each worker may accumulate its gradients independently. If dynamic sampling is in use, the different sampling amounts per layer among workers may also be synchronized to ensure correctness.

On each worker, MCGQ offers practical benefits by enabling the leverage of quantized, sparse gradient representations. However, considering large distributed settings, communication is more likely to be a bigger bottleneck than compute. To alleviate this, we describe our proposed communication compression scheme below.

Communication compression scheme

Existing gradient pruning methods propose to employ different compression schemes to leverage gradient sparsity [Ali+17; Lin+18a]. With MCGQ, since we promote both quantization and pruning, we propose to use run-length encoding (RLE) to compress continuous sequences of zero-valued gradients and transmit only the number of bits necessary to transfer the maximum quantized gradient value for the non-zero gradients.

Assuming the MCGQ variant without accumulated gradients and considering a vector of n quantized gradients of layer l, $g_{l_{int}} = \{g_{l_{int_0}}, g_{l_{int_1}}, \ldots, g_{l_{int_{n-1}}}\}$, each worker sends the following information scheme, adopting (3.5):

1. Number of bits required to represent the maximum non-zero gradient:

$$B_g = 1 + \left\lfloor \log_2 \left(\max_{0 \leq i \leq n-1} |g_{l_{int_i}}| \right) + 1 \right\rfloor. \tag{3.9}$$

2. Number of bits required to represent the longest sequence of $c_0 \in \mathbb{N}$ zero-valued gradients:

$$B_{RLE} = \lfloor \log_2(c_0) + 1 \rfloor. \tag{3.10}$$

3. Use B_g bits to represent the non-zero gradient values as well as the first gradient in each zero-valued gradient sequence.

4. Use B_{RLE} bits to represent the length of each zero-valued gradient sequence.

Figure 3.5 illustrates an example of the proposed scheme, considering $g_{l_{int}} = [2, -1, 0, 0, 0, 3, 0, 1]$ and respective $B_g = 3$, $c_0 = 3$, and $B_{RLE} = 2$.

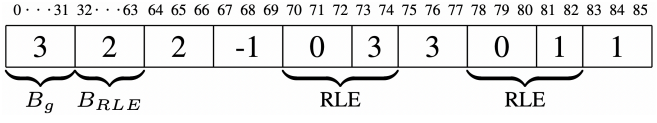


Figure 3.5: Example of our communication compression scheme. In total, 86 bits are used to represent $g_{l_{int}}$, instead of 256 bits ($len(g_{l_{int}}) \times 32$).

We note that the previous scheme may be extended to the accumulated gradients variant. Moreover, if a scale-sensitive optimizer is used in training, the gradient norm $\|g_l\|_1$ for each layer l may be transmitted as well, *i.e.* a 32-bit float, to re-scale the quantized gradients to their original range before performing parameter updates in each worker.

3.3.3 Experimental Results

We now evaluate different variants of MCGQ on different models and tasks, namely logistic regression, image classification, and language modeling.

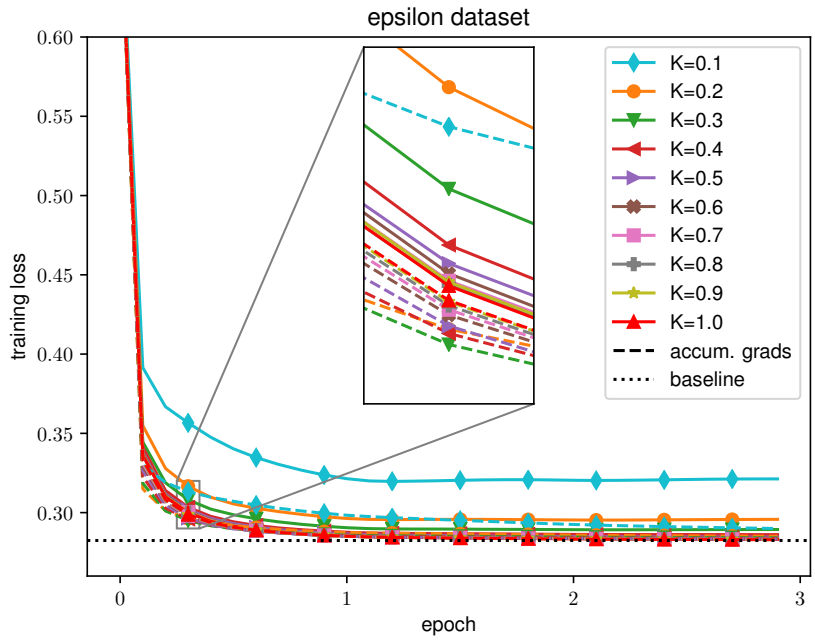


Figure 3.6: Training losses of different MCGQ variations on the epsilon dataset using different sampling rates and accumulated gradients.

Logistic regression

We followed the logistic regression experiments described in Mem-SGD [SCJ18] using the epsilon dataset [Son+08] which contains 400k 2K-dimensional samples. We refer to the original paper for information regarding the objective function and training protocols. As a baseline, we use the results of LogisticSGD from scikit-learn [Ped+11] reported in the aforementioned work. Results using different MCGQ variants and sampling amounts are presented in Figure 3.6.

Overall, we observe that a bigger K often leads to a faster convergence over time, independently of using accumulated gradients or not. However, the gradient information retained by accumulating the gradients across several iterations allows for using smaller sampling amounts, leading to a lower variance between different K.

Image classification

We use dynamic sampling in both the original and accumulated gradients variants to train a ResNet-110 on CIFAR-10. We followed the training settings publicly available[7] using the Lazy Adam optimizer, which was designed to handle sparse gradient updates. The baseline model trained with full-precision gradients reached an accuracy of 91.22%. Comparison results between both MCGQ variants and the baseline are presented in Figure 3.7.

We observe that both MCGQ variants share a similar behavior as the baseline model in terms of training loss. However, both variants reach a better test performance than the 32-bit gradient baseline.

Language modeling

We also investigate using quantized gradients in the text domain by first evaluating language modeling on the Penn Treebank corpus (PTB) [MSM93] dataset. We use a 2-LSTM model [8] with the training details proposed by DGC [Lin+18a], to which we will further compare our method to in Section 3.3.4. For the next experiments on language modeling, we use our proposed communication scheme to assess the compression levels in possible distributed settings. We note that we only simulate distributed scenarios by increasing the batch size of a single

7 https://github.com/bearpaw/pytorch-classification
8 https://github.com/salesforce/awd-lstm-lm

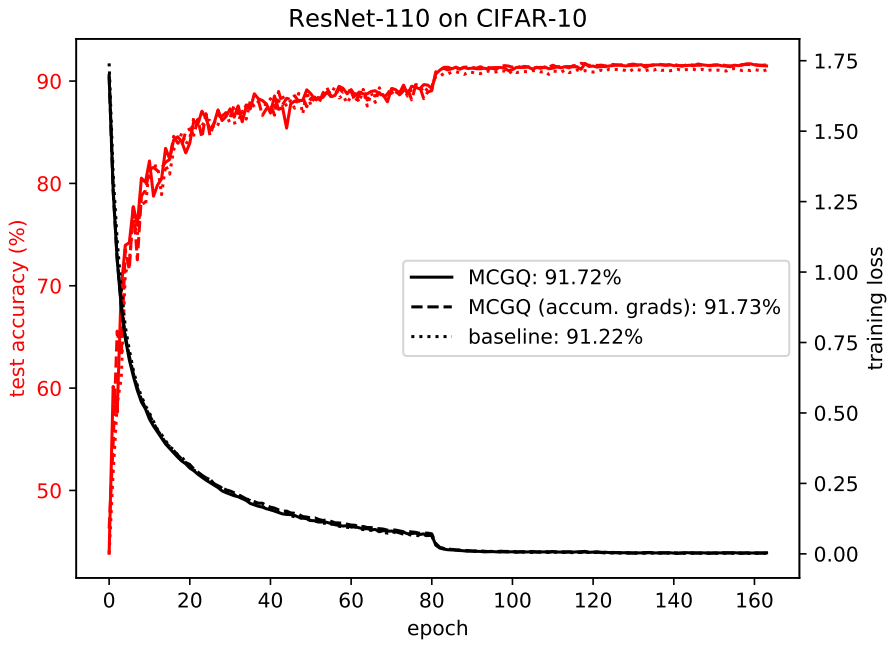


Figure 3.7: Different ResNet-110 accuracies (red lines) and training losses (black lines) using full-precision as well as quantized gradients on CIFAR-10.

worker accordingly. The training losses of different MCGQ compression rates using accumulated gradients, as well as the respective full-precision gradient baseline, are presented in Figure 3.8.

Using our proposed RLE-based compression scheme presented in Section 3.3.2, at 469× fewer communication exchanged compared to the full-precision gradient baseline, the MCGQ model consistently presents lower training losses than the baseline. Further compressing up to 1200×, our model shows comparable training loss progression to the baseline model.

Furthermore, we assess gradient compression on character-level language modeling using Char-RNN trained on Shakespeare excerpts [9]. We use the same 2-layer LSTM and training settings as AdaComp [Che+18] since we directly

9 https://github.com/karpathy/char-rnn

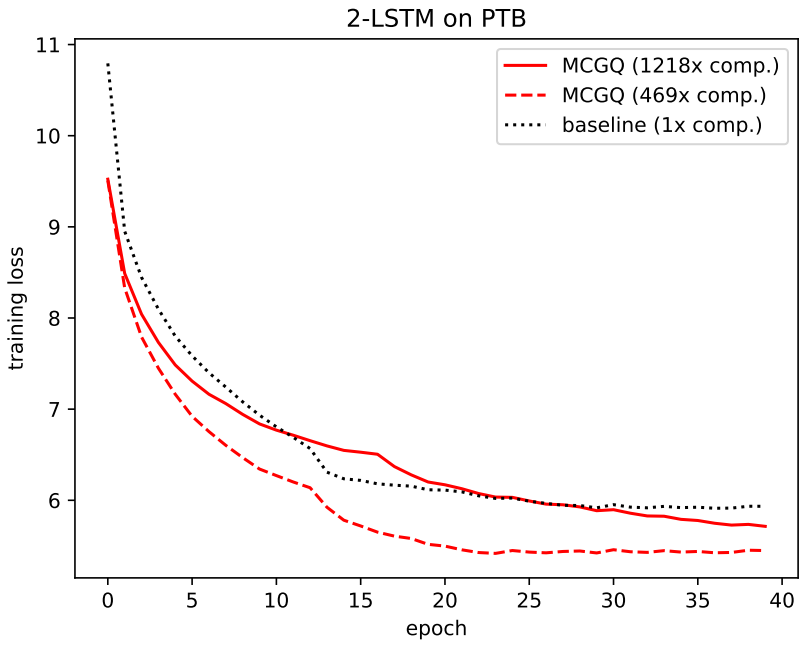


Figure 3.8: Training losses of two MCGQ models with different compression rates using accumulated gradients and different sampling amounts on PTB.

compare MCGQ to AdaComp in Section 3.3.4. The training losses of different models with compressed and uncompressed gradients are presented in Figure 3.9.

Once again, we observe that the different MCGQ models are able to successfully learn even at high compression rates. Moreover, the different models show a similar loss behavior as the baseline model throughout the entirety of the training.

3.3.4 Method Comparisons

We now compare MCGQ against methods that either quantize or prune gradients on the previously described tasks.

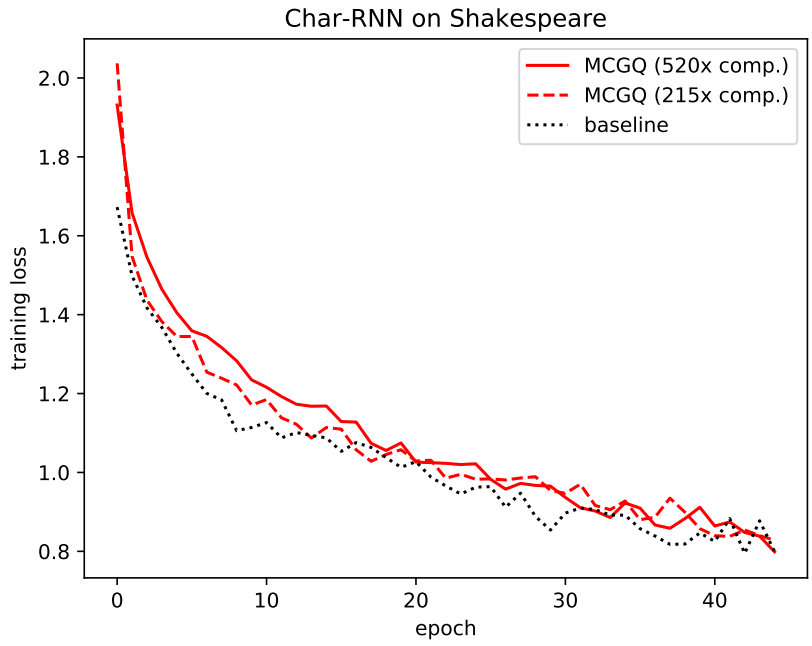


Figure 3.9: Training losses of two MCGQ models with different compression rates using accumulated gradients and different sampling amounts on Shakespeare.

Logistic regression

We first compared MCGQ to Mem-SGD [SCJ18], QSGD [Ali+17], and 32-bit SGD. Comparison results on the epsilon dataset using MCGQ with $K = 1.0$ are shown in Figure 3.10.

We observe that both MCGQ variants show faster convergence than all gradient quantization methods as well as the full-precision 32-bit SGD. Despite both variants showing similar behaviors, accumulating gradients in MCGQ presents slightly faster convergence in the initial training phase.

Image classification

Using the previously described ResNet-110 experiments on CIFAR-10 in Section 3.3.3, we now compare MCGQ with and without accumulated gradients

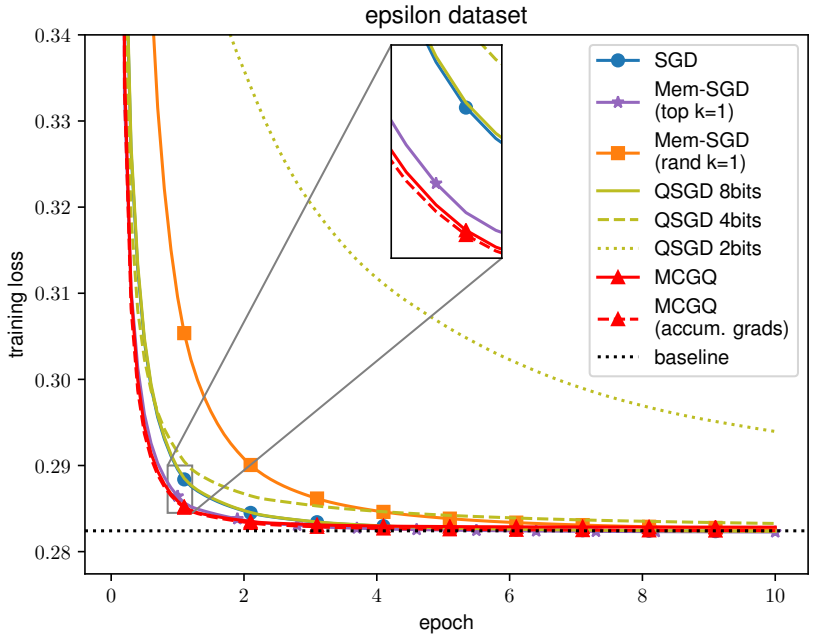


Figure 3.10: Comparison results on the epsilon dataset. Both of our MCGQ variants (represented as red lines) show faster convergence than the compared gradient quantization methods, in terms of training loss.

against QSGD as well as popular gradient pruning methods: Gradient Dropping [AH17] and DGC [Lin+18a]. We note that we use dynamic sampling in both of our variants and initialize K equally among all layers. We report the accuracy differences between the compressed gradients models and their respective baseline models presented in the different works. Similarly to MCQ, we present the average maximum number of bits required to represent the maximum gradient values of all layers, including their sign, at each iteration. Comparison results are presented in Figure 3.11.

We observe that from 3 bits or higher, both MCGQ variants outperform QSGD, with accumulating gradients showing better performance, especially at lower bit-widths. Particularly, we reach the performance of the full-precision gradient baseline at 3 bits instead of 4 bits, as reported by QSGD as well as

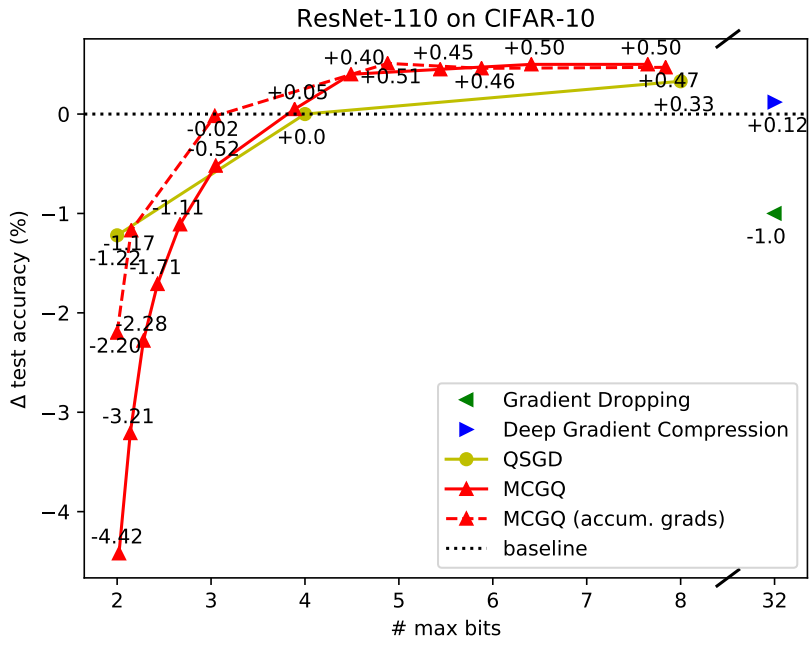


Figure 3.11: Comparison results using ResNet-110 on CIFAR-10. Considering the accuracy difference to the respective 32-bit baseline, both MCGQ variants reach full-precision performance by 4-bit gradient compression.

our variant with no gradient accumulation. Considering higher bit-widths, *i.e.* bigger sampling amounts, both MCGQ variants outperform the baseline in a similar fashion, improving accuracy by approximately 0.5%, outperforming the compared methods, including QSGD which has a 0.33% accuracy increase at 8 bits. We note that since DGC and Gradient Dropping are gradient pruning methods, the bit-width of the non-zero gradients is still represented in full-precision, despite being highly sparse. We will be comparing overall sparsity and compression levels later on with our language modeling experiments.

To continue our comparison with gradient quantization methods, we also experiment with ResNet-18 [He+16] on CIFAR-100, following the experiments [10] presented by ef-signSGD [Kar+19] and reporting the average over 3 training

10 https://github.com/epfml/error-feedback-SGD

runs. Specifically, we compared MCGQ to the following 1-bit gradient compression methods: signSGD [Ber+18], signum [Ber+18], and ef-signSGD [Kar+19]. Since we did not observe any significant benefits of having a dynamic K with this variant, we only report results using the accumulated gradients variant without dynamic sampling from now on. We compress gradient communication by approximately 32×, similarly to the compared binary methods, by using the communication scheme proposed in Section 3.3.2. Comparison results are presented in Table 3.4.

Table 3.4: Comparison results on CIFAR-100 using the best learning rate of the compared methods.

ResNet-18 on CIFAR-100			
Method	Test accuracy (%)	Compression rate	LR
SGD (baseline)	74.02	1×	$5.6e^{-2}$
MCGQ	**74.89**	**≈ 32×**	$\mathbf{5.6e^{-2}}$
ef-signSGD [Kar+19]	74.43	32×	$5.6e^{-2}$
signSGD [Ber+18]	73.14	32×	$5.6e^{-2}$
signum [Ber+18]	72.20	32×	$3.2e^{-4}$

We report the models with the best learning rate (LR) for signSGD and ef-signSGD and apply the same LR schedule to SGD by reducing the LR by 10 at epochs 100 and 150. We observe that MCGQ outperforms the compared methods at the same compression rates and under identical training settings. We present the training progress of the different models in Figure 3.12.

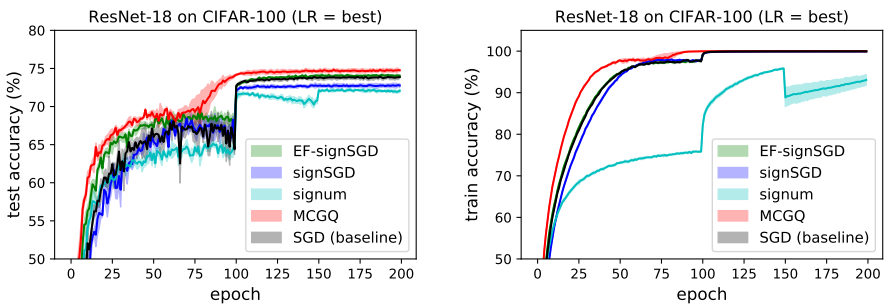


Figure 3.12: Test and train accuracies of different gradient quantization methods on CIFAR-100 using the best learning rates of the compared methods.

We observe that MCGQ showcases the fastest convergence in terms of test and train accuracy among the compared methods, including the 32-bit gradient baseline. While ef-signSGD matches the baseline's progress, the other methods show overall poorer convergence, especially in terms of test accuracy.

Ideally, the different methods would not require the adjustment of training hyper-parameters, such as the learning rate, relative to the 32-bit baseline. Hence, we also compared the different methods using the best learning rate reported for the SGD baseline with momentum (SGDm) presented by Karimireddy et al. [Kar+19]. Comparison results are presented in Table 3.5.

Table 3.5: Comparison results on CIFAR-100 using SGDm's best learning rate.

ResNet-18 on CIFAR-100			
Method	Test accuracy (%)	Compression rate	LR
SGDm	75.20	1×	$1.0e^{-2}$
SGD (baseline)	69.75	1×	$1.0e^{-2}$
MCGQ	**72.57**	**≈ 32×**	$\mathbf{1.0e^{-2}}$
ef-signSGD [Kar+19]	69.69	32×	$1.0e^{-2}$
signSGD [Ber+18]	67.13	32×	$1.0e^{-2}$
signum [Ber+18]	58.90	32×	$1.0e^{-2}$

We observe that, out of all gradient quantization methods, MCGQ presents the closest accuracy to the 32-bit SGD baseline with momentum. Moreover, we note that we also significantly outperform the 32-bit SGD baseline without momentum. These findings suggest the superior applicability of our approach in being integrated into existing training frameworks. We report the training progress of the different methods in Figure 3.13.

We observe MCGQ converges faster in the mid-training iterations out of all compared methods. However, at the end of the training, SGDm presents the best performance in terms of test accuracy. Once again, we observe a poorer performance of the signSGD and signum methods through training.

Language modeling

We conclude our comparisons by analyzing highly sparse gradient pruning methods using the language model experiments described in Section 3.3.3. We start out by comparing MCGQ with DGC [Lin+18a] on PTB in Table 3.6.

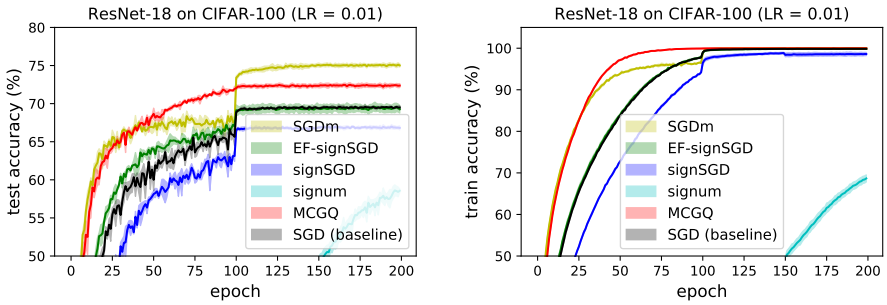


Figure 3.13: Test and train accuracies of different gradient quantization methods on CIFAR-100 using SGDm's best learning rate.

Table 3.6: Compression results of two MCGQ variants with different sampling amounts and Deep Gradient Compression (DGC) on PTB.

2-LSTM on PTB			
Method	Perplexity ↓	Gradient size	Compression rate
Baseline	82.03	194.69 MB	1×
Δ MCGQ	**-0.21**	**0.16 MB**	**1218×**
Δ MCGQ	**-4.30**	**0.42 MB**	**469×**
Δ DGC [Lin+18a]	-0.06	0.42 MB	462×

We observe that, at similar compression rates of 469× and 462× for MCGQ and DGC, respectively, our method improves perplexity by a greater degree relative to the respective full-precision gradient baseline. At approximately 1200× compression rates, MCGQ still manages to outperform the relative perplexity performance when compared to DGC. We note that DGC's baseline model achieves a higher perplexity than our baseline model.

Moreover, we also compare MCGQ to AdaComp [Che+18] on Shakespeare text. Comparison results, in terms of cross-entropy validation loss, are presented in Table 3.7.

We observe that, at more than double the compression rate, MCGQ reaches a similar performance as AdaComp. If compressed to similar compression rates, MCGQ outperforms AdaComp while reducing the validation loss of the baseline model.

Table 3.7: Compression results of two MCGQ variants with different sampling amounts and AdaComp on Shakespeare.

Char-RNN on Shakespeare			
Method	Val. loss ↓	Gradient size	Compression rate
Baseline	1.578	13.28 MB	1×
Δ MCGQ	**+0.020**	**0.03 MB**	**520×**
Δ MCGQ	**-0.016**	**0.06 MB**	**215×**
Δ AdaComp [Che+18]	+0.020	0.07 MB	200×

3.4 Concluding Remarks

In this Chapter, we showed that randomized importance sampling techniques are an effective technique to convert floating-point values to integer values by quantization. Specifically, we proposed two novel Monte Carlo and importance sampling methods to quantize weights and activations in a post-training regime, *i.e.* MCQ, as well as gradients during training, *i.e.* MCGQ. The proposed methods combine quantization and pruning to successfully compress different neural networks on multiple datasets and tasks with minimal to no performance loss compared to the full-precision counterparts. Moreover, the trade-off between performance and compression (both in terms of sparsity and bit-width) may be controlled by adjusting the number of samples.

Since both methods require enough sampling to enable a good approximation, they may require higher bit-width than existing quantization approaches, especially MCQ. To reduce the required precision, one may split the maximum quantized values using Outlier Channel Splitting (OCS) [Zha+19a], which may be seen as orthogonal work. More so, combining existing compression methods on top of MCQ, such as clipping [BNS19], also presents a good complement to ultimately reduce precision without compromising the approximation of the original distribution and without requiring any additional training. Moreover, one may also improve the importance sampling process by incorporating Taylor expansions [Mol+17] in both of our methods. In future work, it would be interesting to evaluate the scalability of MCGQ in a real distributed setting by performing training across multiple workers.

In the context of GANs, MCQ may be used to reduce the memory and computational cost of a pre-trained generator, which will be further discussed in

Chapter 5. Moreover, MCGQ may be used to reduce the communication costs of training GANs in a distributed training, which is an important step considering the increasing trend in the number of models by multi-adversarial frameworks, such as the proposed Dropout-GAN and microbatchGAN, and architectural sizes.

4 Evaluation of Generated Samples

This Chapter is based on two of our refereed papers: Fuzzy Topology Impact (FTI) [MNM21] and Mark-Evaluate [MM20]. The results presented for FTI are an extended version of some of the work done by Julian Niedermeier for his Master's thesis, which I mentored. This preliminary version of FTI [NMM20] was presented at AAAI 2020 workshop on Evaluating Evaluation of AI Systems. For readers unfamiliar with the evaluation of generative models, we refer to Theis et al. [TOB16].

Evaluation of a generated set is an important topic of research given the recent popularity of generative models, mostly fueled by the introduction of GANs. A proper and thorough evaluation of a generative model opens doors for improving the generation process to meet the demands of specific real-world applications.

In this Chapter, we discuss the challenging task of evaluating generated samples, specifically from different GANs as well as language models. We start by introducing existing evaluation metrics as well as their current drawbacks in Section 4.1. Then, we propose one novel metric called Fuzzy Topology Impact (FTI) in Section 4.2, and a new family of metrics called Mark-Evaluate in Section 4.3. Finally, we provide some closing thoughts on the proposed work as well as future directions in Section 4.4.

4.1 Related Work

Existing evaluation methods may be broadly categorized into three main groups: analysis of likelihoods [TOB16] and probability distributions [Gre+12; Heu+17], topological analysis of manifolds [KO18; Kyn+19; Saj+18], and classifier-based methods [GKV17; Sal+16; SSA18]. A description of popular metrics follows.

Single-valued metrics

We now provide an overview of two evaluation metrics that have been previously referenced in this thesis: Inception score (IS) [Sal+16] and Fréchet Inception

Distance (FID) [Heu+17]. Since proposed, such metrics have been widely popular in evaluating GANs by making different use of an Inception-V3 [Sze+16] model pre-trained on ImageNet.

Namely, IS proposes to measure the Kullback-Leibler Divergence [KL51] between the conditional probability distribution of a generated sample belonging to an ImageNet class and the marginal distribution across the classes assigned to all generated samples. Both distributions are constructed from Inception-V3's output distribution. The intuition is that, given the broad classes of ImageNet, a pre-trained model on that data should assign a high probability to one class for each generated sample, depending on its quality. Moreover, the differently assigned classes should be close to uniformly distributed, depending on the sample variety in the generated set.

On the other hand, FID uses the embeddings of Inception-V3's last pooling layer to represent both real and generated images in a feature space. Hence, contrary to IS, FID uses the real data to assess a generated set. As the name suggests, FID measures the similarity between the real and the generated image embeddings by using the Fréchet Distance. Note that high and low values represent a better generated set for IS and FID, respectively. One of the practical benefits of FID is to be more accurate than IS in assessing sample diversity, namely when only one identical sample is generated for each class.

Note that neither high IS or low FID, or vice-versa, necessarily suggest a good or bad sample quality or a good or bad sample diversity. This lack of clarity of single-valued metrics may be unsatisfactory in scenarios where independent insights regarding either sample quality or sample diversity in the generated set are important, as it will be discussed throughout this Chapter.

Double-valued metrics

Several double-valued metrics have been recently proposed to individually measure sample quality and sample diversity. Similar to FID, the following metrics first use the embeddings of different pre-trained models to position real and generated samples in a shared feature space. Then, precision (sample quality) and recall (sample diversity) may be measured using the different strategies described below.

Precision and recall for distributions (PRD) was initially proposed by Sajjadi et al. [Saj+18]. They use *k-means* [Mac67] to approximate the supports of the real and generated sets using the centroids from the different clusters. The generated

set is then assessed in terms of relative probability densities. More specifically, precision measures the probability of a generated sample being inside the support of the real distribution. On the other hand, the opposite procedure is used to measure recall by calculating the probability of a real sample being inside the support of the generated distribution.

A follow-up method called improved precision and recall (IMPAR) was introduced by Kynkäänniemi et al. [Kyn+19] to simplify the manifold approximation process by using *k-nearest neighbors* (KNN) instead of k-means. More specifically, the real and generated manifolds are constructed by several hyperspheres that connect each sample to the k-closest neighbor of its respective set. Then, precision is measured by the ratio of generated samples in the real manifold, and recall is measured by the ratio of real samples in the generated manifold. Due to its simplicity, we also employ such manifold approximation using KNN in our proposed metrics in this Chapter: FTI and Mark-Evaluate.

Topology-based metrics have also been recently proposed to assess language generation [Zha+19b; Zha+20a] and are starting to supersede popular single-valued text metrics such as BLEU [Pap+02], ROUGE [Lin04] and METEOR [BL05]. We note that, even though FID was initially proposed by Heusel et al. [Heu+17] with image generation in mind, Semeniuta et al. [SSG18] proposed to extend FID to the text domain by simply replacing the pre-trained image model, Inception-V3, with a text model, InferSent [Con+17]. In this Chapter, we also extend both PRD and IMPAR to the text domain by using sentence embeddings [RG19] or contextualized word embeddings [Dev+19]. Such representations have recently been shown to improve language assessment in a variety of language tasks [MBC19; Rei+20; SDP20; Zha+19b; Zha+20a].

In this Chapter, we also study unsupervised conditional language metrics, namely BERTScore [Zha+20a] and MoverScore [Zha+19b], which show a high correlation to human evaluation on conditional tasks, such as machine translation and text summarization. BERTScore returns a double-valued metric to assess a generated sentence against a reference sentence by measuring the required transport between semantically similar words from the different sentences. MoverScore returns a single-valued metric representing the minimum transport between an entire generated sentence and a reference sentence by leveraging contextual representations. We now proceed to discuss our proposed metrics, FTI and Mark-Evaluate, in Sections 4.2 and 4.3, respectively.

4.2 Fuzzy Topology Impact

We now introduce our first double-valued metric, Fuzzy Topology Impact (FTI), which uses topological representations and Fuzzy Logic to assess a generated set in a fine-grained manner. In this Chapter, we use FTI to assess only image generation, whereas our other family of metrics, Mark-Evaluate, is used to evaluate language generation in Section 4.3. However, we note that since both methods rely on embeddings of pre-trained models, we may apply each metric to different data domains.

We describe FTI in Section 4.2.1 and important implementation details in Section 4.2.2. We evaluate different configurations of FTI in Section 4.2.3 and show empirical evidence of the benefits of having a finer-granularity compared to existing methods in Section 4.2.4.

4.2.1 Topological Representations and Fuzzy Logic

FTI leverages the topological representations from Uniform Manifold Approximation and Projection (UMAP) [MHM18], which is a popular method for dimension reduction that leverages Riemannian geometry and Fuzzy Logic. In our use case, we use such representations to assess a generated set where each sample is a feature representation from a pre-trained model. Namely, UMAP is used to retrieve a directed, weighted graph using each sample as a node, with the connections being established by using KNN and the weights determined by using Riemannian geometry principles and fuzzy logic. In the end, such graph provides not only an overview of nearby samples but also how near they are relative to one another. In other words, the weight of each connection presents the probability of its existence in the original feature space.

To assess sample quality and diversity, we start by constructing a real data graph for the real samples and a generated data graph for the generated samples. Then, we analyze how similar the two graphs are by measuring the impact that samples from one graph have on the other. More specifically, we define impact as the drop in the average probability of the existence of each connection on each original graph after inserting new samples or nodes from the other graph. Hence, quality relates to the impact that generated samples have on the real data graph, whereas diversity represents the impact that real samples have on the generated data graph. Even though this process is inspired by notions of precision and

recall of existing topology-based metrics, our method *Fuzzy Topology Impact* (FTI) enables a more sensitive and finer-grained assessment of a generated set.

Topological representations

We will provide a brief overview of UMAP's steps for the construction of each graph as well as our proposed modifications for the adoption of such graphs to the evaluation use case. In UMAP, each graph may be seen as an approximation of the data manifold. Namely, by assuming samples to be uniformly distributed across the manifold, UMAP defines a notion of local distance of each sample by normalizing all sample distances on its neighborhood by the k-th neighbor's distance, or by a scaling factor σ. Such assumption enables the usage of Riemannian geometry and Fuzzy Logic to formally prove the efficacy of UMAP's weighted, directed graph in terms of manifold approximation [MHM18].

Let us represent each graph $G = (X, E)$ by a set of nodes or feature representations $X = \{x_1, \ldots, x_N\}$ and a set of $N \times k$ directed edges $E \subseteq \{(x_i, x_{i_j}) \mid j \in \{1, \ldots, k\}, i \in \{1, \ldots, N\}\}$, where $x_{i_j} \in X$ are the k-nearest neighbors of x_i under the euclidean distance d. The weight or probability of existence $p_{x_i, x_{i_j}} \in [0, 1]$ of a directed edge $(x_i, x_{i_j}) \in E$ is:

$$p_{x_i, x_{i_j}} = \exp\left(\frac{-d(x_i, x_{i_j})}{\sigma_i}\right), \tag{4.1}$$

where $\exp(\cdot)$ is the natural exponential function and $\sigma_i \in \mathbb{R}^+$ is the scaling factor of x_i such that the weights of its outgoing connections are standardized by:

$$\sum_{j=1}^{k} p_{x_i, x_{i_j}} = \log_2(k), \tag{4.2}$$

with $\log_2(k)$ being originally proposed by McInnes et al. [MHM18] based on a series of empirical studies. Since each sample has a corresponding scaling factor, the local connectivity in each neighborhood set is still preserved despite the global normalization of all weights in the graph.

We note that the described graph differs from the final graph retrieved by UMAP due to the different use cases: while we aim at performing sample assessment, UMAP's goal is to reduce sample dimensionality. Namely, McInnes et al. [MHM18] further extends the previous graph to an undirected graph by combining the weights of edges with opposite directions. On the other hand, we

simply use the described directed graph. By retaining the probability of existence of both incoming and outgoing connections of each sample, we aim at inducing higher outlier awareness in the overall assessment of a generated set.

Impact evaluation

We measure the impact that an evaluation set X' has on a reference set X by first calculating the average probability of existence of the edges of graph G, constructed only from samples in X:

$$\overline{P_G} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{k} p_{x_i, x_{i_j}}}{N \times k}. \tag{4.3}$$

We note that, given (4.2), $\overline{P_G}$ is constant. However, we are only interested in calculating the impact in the final graph. More specifically, we insert a new sample $x'_i \in X'$ to G and update the weights of all original edges accordingly:

$$p^{x'_i}_{x_i, x_{i_j}} = \begin{cases} 0, & \text{if } j = k \wedge d(x_i, x_{i_k}) > d(x_i, x'_i) \\ \exp\left(\dfrac{-d(x_i, x_{i_j})}{\sigma'_i}\right), & \text{if } j \neq k \wedge d(x_i, x_{i_k}) > d(x_i, x'_i) \\ p_{x_i, x_{i_j}}, & \text{otherwise.} \end{cases} \tag{4.4}$$

Namely, if x'_i is not a k-nearest neighbor of any original sample, no changes are performed in G's weights. However, if x'_i is in the neighborhood of a given original sample x_i, we remove the original edge from x_i to its previous k-th nearest neighbor and update the rest of the outgoing weights of x_i following (4.1). Moreover, we update the original σ_i by introducing σ'_i, where σ'_i satisfies:

$$\sum_{j=1}^{k-1} \left(\exp\left(\frac{-d(x_i, x_{i_j})}{\sigma'_i}\right)\right) + \exp\left(\frac{-d(x_i, x'_i)}{\sigma'_i}\right) = \log_2(k). \tag{4.5}$$

The updated average probability of existence of the original G's weights after inserting x'_i may then be described as:

$$\overline{P_{G, x'_i}} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{k} p^{x'_i}_{x_i, x_{i_j}}}{N \times k}. \tag{4.6}$$

Considering a set of evaluation feature representations $X' = \{x'_1, \ldots, x'_{N'}\}$, the overall impact (FTI) may be defined as the average drop in $\overline{P_G}$, calculated using the original graph G with a set of feature representations X and k nearest neighbors:

$$FTI(X, X', k) = \frac{\sum_{i=1}^{N'} \left(\overline{P_G} - \overline{P_{G,x'_i}} \right)}{N'}. \tag{4.7}$$

A pseudo-code for the proposed method is presented in Algorithm 8. Assuming the euclidean distances between each reference sample and its k-nearest neighbors are provided, we iterate through the reference samples that have a given evaluation sample x'_i in their neighborhood and update the original graph probabilities accordingly. During this process, we keep track of the total impact induced by all evaluation samples. We note that we use the function SmoothDistApprox$(\cdot)$, originally proposed by McInnes et al. [MHM18], to find each updated scaling factor σ'_i through a binary search.

Quality and diversity

So far, we have introduced FTI in a broader sense by calculating the drop in the average probability of existence that an evaluation set X' has on a reference set X. However, we would like to assess a generated set in terms of sample quality and sample diversity separately. To this end, we calculate sample quality by setting X to be the set of real samples, X_r, and X' to be the set of generated samples, X_g. Conversely, to measure sample diversity, we set X to X_g and X' to X_r. Hence, the assessment of quality and diversity of a generated set X_g is proposed as:

$$\text{quality} = \text{FTI}(X_r, X_g, k) \qquad \text{diversity} = \text{FTI}(X_g, X_r, k). \tag{4.8}$$

We note that this assessment concept was first introduced by Sajjadi et al. [Saj+18] and adopted by Kynkäänniemi et al. [Kyn+19] in their proposed double-valued metrics based on precision and recall. In FTI, however, we relate sample quality to the impact that, on average, a generated sample has on the real data graph. On the other hand, sample diversity is measured by the impact that, on average, a real sample has on the generated data graph.

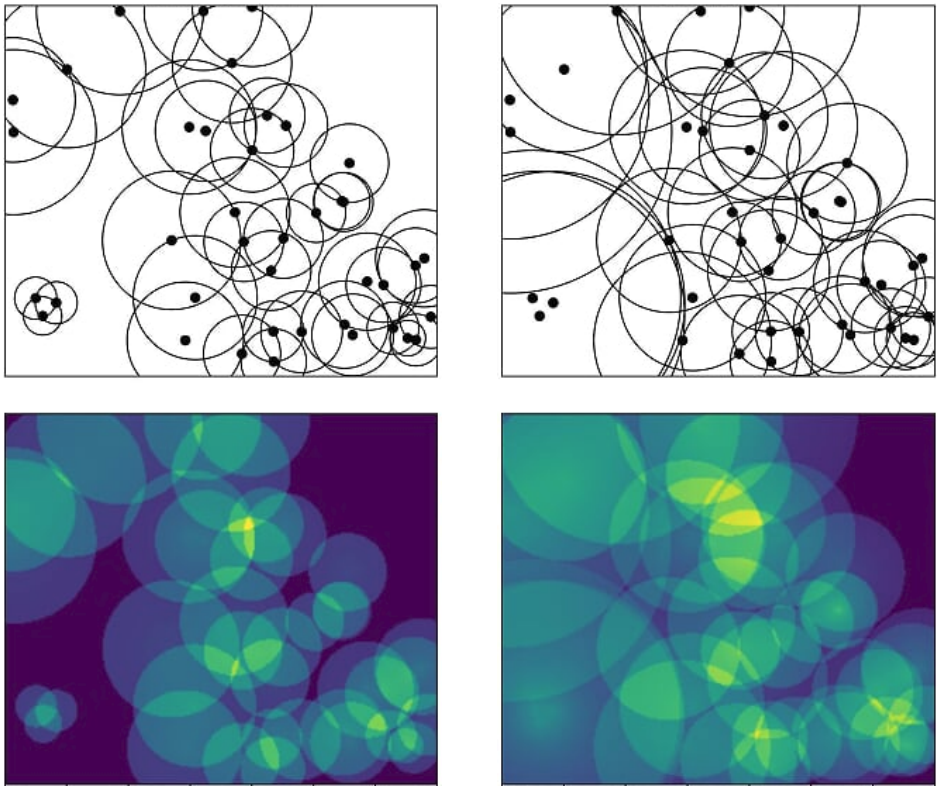
Algorithm 8 Fuzzy Topology Impact.

Input: Original graph G constructed from a reference set X using k neighbors, evaluation set X', dictionary *dist* with the euclidean distances between each reference sample and is k nearest neighbors

impact $\leftarrow 0$
for each $x'_i \in X'$ **do**
  $p^X \leftarrow 0$
  $p^{X'} \leftarrow 0$
  count $\leftarrow 0$
  for each $x_i \in X$ **do**
    if $d(x_i, x'_i) < d(x_i, x_{i_k})$ **then**
      count $\leftarrow$ count + 1
      $p^X \leftarrow p^X + p_{x_i, x_{i_k}}$
      del dists$[(x_i, x_{i_k})]$
      $p^{x'_i}_{x_i, x_{i_k}} \leftarrow 0$
      dists$[(x_i, x'_i)] \leftarrow d(x_i, x'_i)$
      $\sigma'_i \leftarrow$ SmoothDistApprox(dists, k)
      for $j = 1, \ldots, k-1$ **do**
        $p^X \leftarrow p^X + p_{x_i, x_{i_j}}$
        $p^{x'_i}_{x_i, x_{i_j}} \leftarrow \exp\left(\frac{-d(x_i, x_{i_j})}{\sigma'_i}\right)$
        $p^{X'} \leftarrow p^{X'} + p^{x'_i}_{x_i, x_{i_j}}$
      end for
    end if
  end for
  impact $\leftarrow$ impact $+ p^X - p^{X'}$
end for
return $\frac{\text{impact}}{N'}$

4.2.2 Implementation Details

The only adjustable parameter of FTI is the number of neighbors, k. Hence, we now focus on the effects of different values for k in the assessment of a generated set.



(a) $k = 2$.

(b) $k = 3$.

Figure 4.1: Impact assignment given 40 random original points using (a) $k = 2$ and 3 (b) $k = 3$. The top row shows the hyperspheres from each sample to its k-th nearest neighbor, while the bottom row shows the different degrees of impact assigned to each hypersphere. Warmer colors represent high impact, cooler colors indicate low impact, with the darkest color indicating no impact.

Number of neighbors

The number of neighbors directly affects the approximation of the data manifold performed by KNN. Intuitively, a larger k will broaden the estimated manifold whereas a smaller k will shrink such estimation. The resolution of the manifold estimation leads to the assignment of different degrees of impact over the data space by FTI. A visual representation of such effects is presented in Figure 4.1.

We observe that a higher k increases the chances of a new (random) sample receiving a higher impact than a smaller k. Moreover, we would like to note the importance of using probability weights (FTI) instead of binary weights (IMPAR), which may otherwise result in a disproportionate (large) coverage of the data space. (Check Figure 4.1 (b) for a visual example of such coverage.) Hence, the finer-grained assessment introduced by our approach is especially important if using a higher k or if dealing with sparse data.

4.2.3 Experimental Results

We now evaluate the effects of using different values for k on assessing realistic images generated from pre-trained StyleGAN2 [Kar+20b] models on LSUN-Church [Yu+15], LSUN-Horse [Yu+15], and FFHQ [KLA19]. We simulate different levels of sample diversity by using different truncation ψ, namely $\psi = 0.5$ and $\psi = 1.0$, which translate to more and less diversity, respectively [BDS19; KD18; KLA19; Kyn+19].

Truncation in StyleGAN2

Results using different values for k and truncation levels are presented in Figure 4.2. We rescaled both the generated and real images to 256x256 for faster computation.

We observe that all variants assess the generated sets correctly, showing an increase of sample diversity while maintaining sample quality as ψ increases. Moreover, smaller k values tend to show a higher FTI ratio than higher k values, relative to the respective results at $\psi = 0.5$. This suggests that the higher impact spread across the data space when using more neighbors, as previously mentioned in Section 4.2.2, results in the assignment of a higher impact to generated samples with $\psi = 0.5$ in the real data graph. We observed similar conclusions on identical experiments performed with StyleGAN2's predecessor, StyleGAN [KLA19].

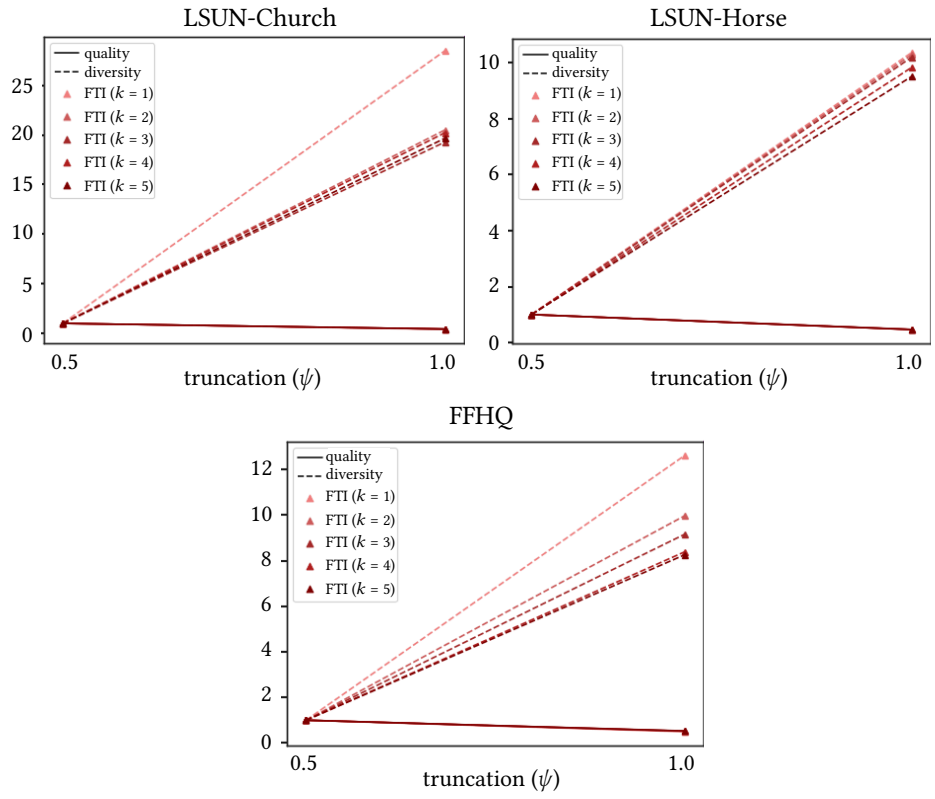


Figure 4.2: Truncation results on StyleGAN2 using different values for k on LSUN-Church, LSUN-Horse and FFHQ, normalized by their respective values at $\psi = 0.5$.

4.2.4 Method Comparisons

We now compare FTI to existing single-valued (IS [Sal+16] and FID [Heu+17]) and double-valued metrics (PRD [Saj+18] and IMPAR [Kyn+19]) on several experiments. We use the feature representations of Inception-V3 to represent image samples. Due to the different range of values retrieved by FTI and the compared double-valued metrics, we report the different results in terms of ratio for an easier comparison analysis. We use $k = 3$ across all experiments to match the number of neighbors originally proposed by IMPAR.

We start by extending the previous StyleGAN2 truncation experiment in Section 4.2.3 to the compared methods. Moreover, we continue a synthetic

analysis of sample quality and sample diversity on Fashion-MNIST [XRV17], CIFAR-10, and CIFAR-100 [Kri09] by simulating noise sensitivity as well as mode addition and invention. For these experiments, we use the original training set as the real set and several modifications of the testing set as the generated set.

Truncation in StyleGAN2

Comparison results between FTI and compared methods is presented in Figure 4.3.

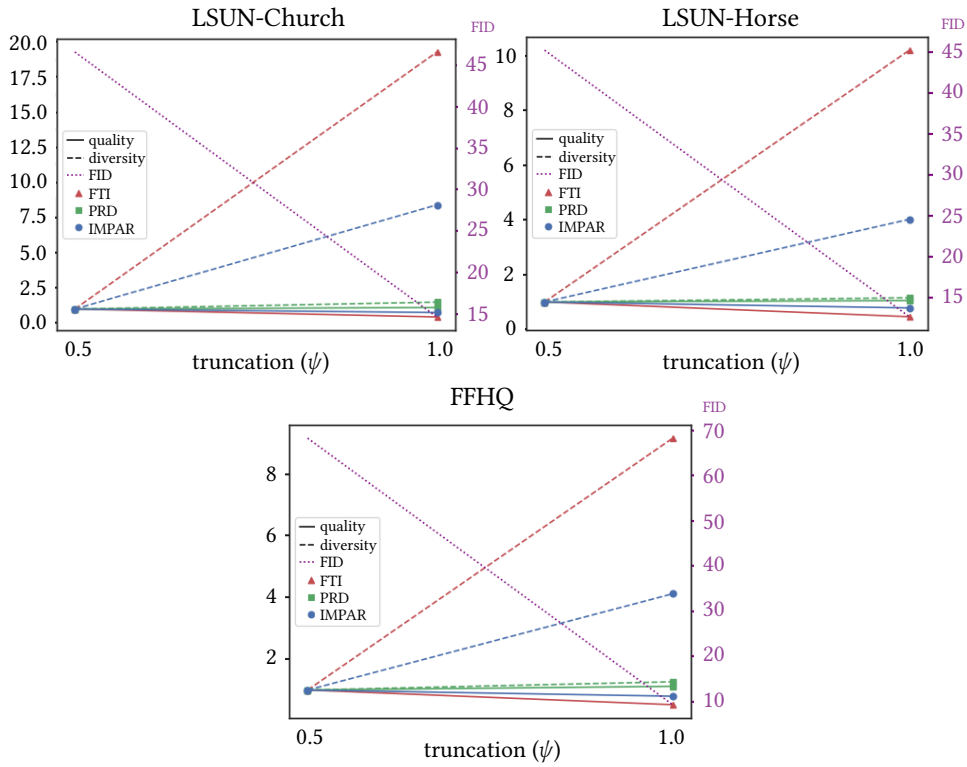


Figure 4.3: Truncation results on StyleGAN2 using the different methods on LSUN-Church, LSUN-Horse and FFHQ, normalized by their respective values at $\psi = 0.5$.

We observe that FTI shows a clearer expected behavior when increasing ψ than the compared methods by maintaining its quality assessment and increasing

its diversity assessment. Specifically, even though less pronounced, IMPAR also shows similar behavior to FTI, while PRD fails to show a substantial change in its assessment. FID improves as ψ increases, suggesting a bias towards sample diversity in its single-valued assessment which we further discuss in Chapter 5.

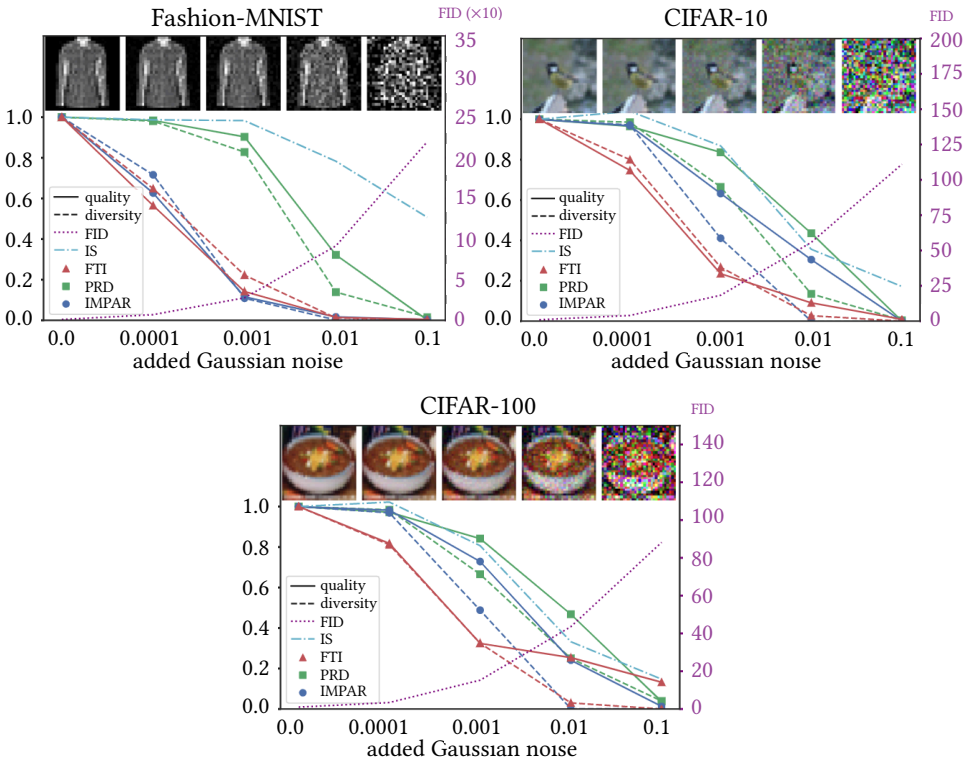


Figure 4.4: Method comparisons with added Gaussian noise on Fashion-MNIST, CIFAR-10 and CIFAR-100 test images, normalized by the respective metric values with no noise.

Noise sensitivity

To simulate different degrees of sample quality, we incrementally added Gaussian noise to the test set images. Note that, since we are distorting the test image samples in the feature space, sample diversity relative to the real images is also likely to be negatively affected. Hence, both assessments of sample quality and

sample diversity are expected to decrease, especially at higher noise distortions. Comparison results are shown in Figure 4.4.

We observe that, in general, all metrics show high sensitivity to noise, presenting a deteriorated assessment at visually negligible noise amounts. (We note that such noise changes may be amplified at the feature representation level though.) More concretely, IS shows less noise sensitivity on Fashion-MNIST compared to CIFAR-10 and CIFAR-100, similarly to PRD, unexpectedly showing a better assessment at low noise amounts on the latter datasets. On the other hand, FID shows a consistent sensitivity to noise on the different datasets. Once again, both FTI and IMPAR show a similar assessment behavior, even though our method shows a higher sensitivity to noise given its finer-grained nature.

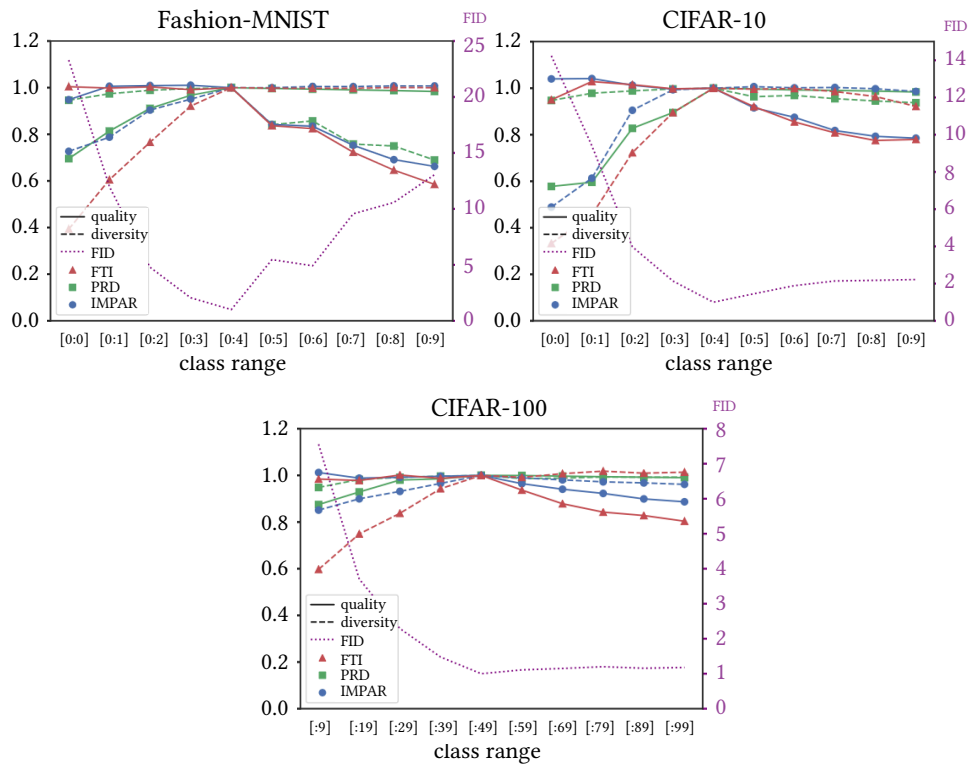


Figure 4.5: Method comparisons for mode addition and invention on Fashion-MNIST, CIFAR-10, and CIFAR-100, normalized by the respective values at half of the classes.

Mode addition and invention

To simulate a drop in sample quality and an increase in sample diversity, and vice-versa, we follow the mode addition and invention experiments proposed by Sajjadi et al. [Saj+18]. Specifically, we drop half the classes of the real image set, which remains constant throughout the entire experiment, and progressively add classes to the generated set, *i.e.* the test set in our use case. We start by adding classes present in the real set, one by one, to stimulate mode addition. Then, we proceed to add the rest of the classes to simulate mode invention. Comparison results are presented in Figure 4.5.

We observe that FTI exhibits the expected behavior across all datasets: during the mode addition phase, quality remains constant while diversity increases, whereas diversity remains constant and quality decreases during the mode invention phase. PRD fails to detect mode invention on CIFAR-10 and CIFAR-100 and shows contradictory behavior on its quality and diversity assessments. Moreover, IMPAR is less sensitive than FTI across the board while showing ambiguous diversity in the mid-class ranges of the mode addition phase on CIFAR-10. Even though FID shows sensitivity to mode addition, it fails to significantly detect mode invention on CIFAR-10 and CIFAR-100, reinforcing the importance of a more thorough assessment by using double-valued metrics.

4.3 Mark-Evaluate

We now shift our focus from assessing image generation to evaluating text generation. To this end, we provide a thorough study regarding applying existing image evaluation metrics, namely FID [Heu+17], PRD [Saj+18] and IMPAR [Kyn+19], to challenging language assessment tasks. Moreover, we present a new family of both single-valued and double-valued metrics that we call Mark-Evaluate. Our metrics leverage different population estimation methods widely used in ecology to estimate the size of animal populations in the wild.

We introduce our proposed metrics in Section 4.3.1 as well as important implementation details in Section 4.3.2. The efficacy of the proposed metrics is showcased in Section 4.3.3 and a comparison to existing metrics in synthetic experiments as well as human evaluation correlation is presented in Section 4.3.4.

4.3.1 Population Estimation Methods

Population estimation methods have been widely employed over the last decades to study the evolution of species in ecology [Kre89]. Existing methods may be separated into two categories focusing on either open populations or closed populations. In our use case, we use several closed population methods which assume the true population size to remain constant throughout the estimation process. More specifically, we use the final population estimation of the different methods to assess a generated set consisting of either contextualized word [Dev+19] or sentence [RG19] embeddings, depending on the task, given the corresponding real set.

Our family of metrics, Mark-Evaluate (ME), consists of two single-valued metrics and one double-valued metric. Concretely, we use two mark-recapture methods which leverage a single marking and recapture process ($\text{ME}_{\text{Petersen}}$) or multiple markings and recaptures ($\text{ME}_{\text{Schnabel}}$) using the Petersen [Ric75] and Schnabel [Sch38] estimators, respectively. Moreover, we use a maximum-likelihood method based on the program CAPTURE [Oti+78], that relies on multiple markings and captures ($\text{ME}_{\text{CAPTURE}}$) for the final population size estimation. The capture mechanism used by the different estimators is performed using capture volumes, which are calculated by the hyperspheres to the k-closest neighbors on each set, following IMPAR [Kyn+19] and FTI. A visual comparison of the different methods using $k = 1$ and considering a reference set S and an evaluation set S' is presented in Figure 4.6.

Considering the reference (S) and evaluation (S') sets, the true population size (P) is $P = |S| + |S'|$, since we are only dealing with closed populations. Hence, given an estimated population size ($\widehat{P}$), the accuracy loss (A) of the estimation may be measured as:

$$A(P, \widehat{P}) = \max\left(\left|\frac{\widehat{P} - P}{P}\right|, 1\right), \tag{4.9}$$

with $A(P, \widehat{P}) \approx 0$ representing a good estimation and $A(P, \widehat{P}) \approx 1$ indicating a poor estimation of the population size. In general, a higher number of captured samples will improve the estimation of all methods since more information is provided regarding the data space. In our use case, this means that, if evaluation samples tend to be within the hyperspheres of reference samples, and vice-versa, each metric will give a high assessment score to the generated set.

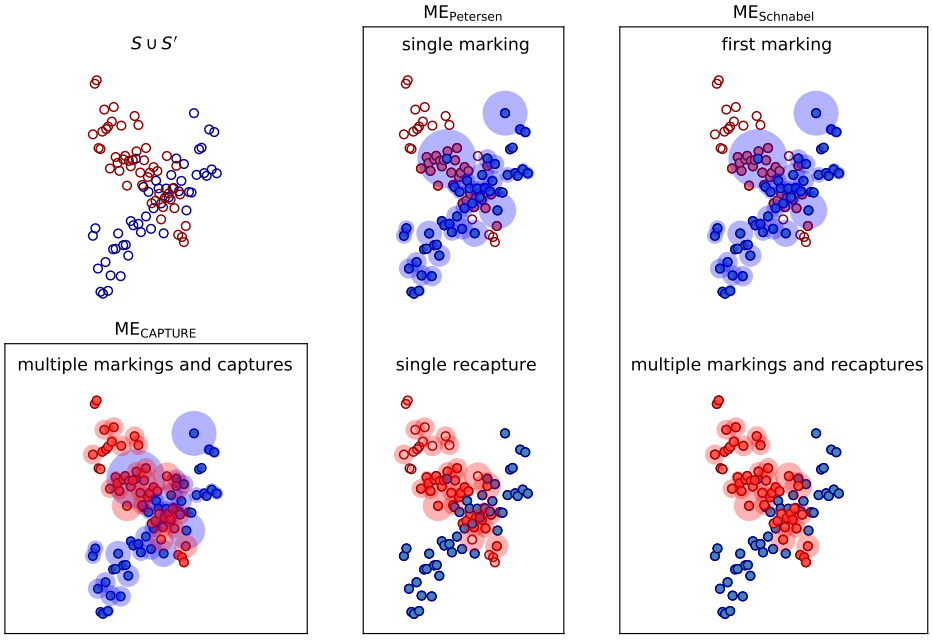


Figure 4.6: Our evaluation metrics are based on population estimation methods. Reference ($s \in S$) and evaluation ($s' \in S'$) samples are represented as blue and red circles, respectively. Filled circles represent marked samples. In $ME_{Petersen}$ and $ME_{Schnabel}$, we first capture and mark all samples inside any hypersphere of s. Then, in $ME_{Petersen}$, we count the number of marked samples (recaptures) inside any hypersphere of s'. In $ME_{Schnabel}$, we perform a similar process iteratively, marking and recapturing samples inside the hypersphere of each s'. This results in all samples being marked in the end. Alternately, $ME_{CAPTURE}$ captures and marks samples inside any hypersphere of s and s'.

4.3.2 Implementation Details

In this Section, we discuss how each estimator calculates its population size estimate $\widehat{P}$. For that purpose, we adapt the formulations of Kynkäänniemi et al. [Kyn+19] and define a function f, which returns a binary representing whether a sample $s' \in S'$ is inside the capture volume, *i.e.* hypersphere, of some sample $s \in S$:

$$f(s', S) = \begin{cases} 1, & \text{if } ||s' - s||_2 \leq ||s - \text{NN}_k(s, S)[-1]||_2 \text{ for at least one } s \in S \\ 0, & \text{otherwise,} \end{cases} \tag{4.10}$$

where $\text{NN}_k(s, S)$ is a function that returns a set with the k-nearest neighbors of $s \in S$, including itself. The returned set is ordered by ascending Euclidean distances, with $\text{NN}_k(s, S)[-1]$ representing the k-th nearest neighbor of s.

Petersen estimator

As previously presented in Figure 4.6, the Petersen estimator [Ric75] uses a single marking and a single recapture step. More specifically, it simply considers the ratio of marked samples (M) in the population size (P) to be the same as the ratio of recaptured samples (R) in the captured samples (C), in the marking and recapture steps, respectively. Recalling S and S' as the sets of reference and evaluation samples, respectively, the Petersen's population size estimate, $\widehat{P}_{\text{Petersen}}$, is:

$$\widehat{P}_{\text{Petersen}}(S, S') = \frac{C(S, S')M(S, S')}{R(S, S')}. \tag{4.11}$$

In our use case, the marking step consists of marking all samples inside the hypersphere of any s. Hence, the number of marked samples (M) is calculated by:

$$M(S, S') = |S| + \sum_{s' \in S'} f(s', S). \tag{4.12}$$

Conversely, the recapture step first consists of capturing all samples inside the hypersphere of any s'. Hence, the number of captured samples (C) is measured by:

$$C(S, S') = |S'| + \sum_{s \in S} f(s, S'). \tag{4.13}$$

Then, it requires counting the number of recaptured samples, *i.e.* the captured samples that were previously marked:

$$R(S, S') = \sum_{s' \in S'} f(s', S) + \sum_{s \in S} f(s, S'). \tag{4.14}$$

Schnabel estimator

The extension of the previous method to multiple markings and recaptures was proposed by Schnabel [Sch38]. Considering T consecutive Petersen estimates and the reference (S) and evaluation (S') sets, the population size estimate of the Schnabel estimator, $\widehat{P}_{\text{Schnabel}}$, is given by:

$$\widehat{P}_{\text{Schnabel}}(S, S') = \frac{C_T(S, S') M_T(S, S')}{R_T(S, S')}. \tag{4.15}$$

Throughout the estimation process, at a given iteration $t \in \{1, \ldots, T\}$, the set of marked samples may be recursively defined as:

$$M(t, S, S') = \begin{cases} S \cup \{s' \in S' | f(s', S) = 1\}, & \text{if } t = 1, \\ S \cup S', & \text{if } t = T, \\ M(1, S, S') \cup \left(\bigcup_{i=1}^{t-1} \text{NN}_k(s'_i, S') \right), & \text{otherwise,} \end{cases} \tag{4.16}$$

with the set of evaluation samples S' being defined as $\{s'_1, \ldots, s'_{|S'|}\}$. At $t = 1$, the marking step is identical to the single step of $\text{ME}_{\text{Petersen}}$. At subsequent iterations, newly captured (not marked) samples are marked. At the end of the estimation process, all samples are marked since we iterate through all samples. Hence, at $t = T$, the number of marked samples (M_T) is calculated by:

$$M_T(S, S') = |M(T, S, S')|. \tag{4.17}$$

After all recapture steps, the total number of captured samples consists of: (1) the number of all evaluation samples, (2) their k-th nearest evaluation samples, and (3) the number of reference samples inside the hyperspheres of each evalu-

ation sample $s' \in S'$. Hence, at $t = T$, the number of captured samples (C_T) is measured by:

$$C_T = (k+1) \times |S'| + \sum_{s' \in S'} \sum_{s \in S} f(s, \mathrm{NN}_k(s', S')). \tag{4.18}$$

Since all reference samples are marked in the first marking step, the total number of recaptured samples consists of: (1) the number of reference samples inside the hyperspheres of each evaluation sample s', and (2) the number of captured k-th nearest evaluation samples already marked. Hence, at $t = T$, the number of recaptured samples (R_T) is computed by:

$$R_T(S, S') = \sum_{i=1}^{|S'|} \sum_{j=1}^{|S|} \Big(f(s_j, \mathrm{NN}_k(s'_i, S')) + |M(i, S, S') \cap \mathrm{NN}_k(s'_i, S')| \Big). \tag{4.19}$$

Program CAPTURE

One may also use the *model null* from program CAPTURE [Oti+78] to estimate the population size that maximizes the log-likelihood given a total number of marked (M_T) and captured (C_{total}) samples over $T = |S| + |S'|$ trials, with S and S' representing the reference and evaluation sets, respectively. More specifically, the final population estimate ($\widehat{P}_{\text{CAPTURE}}$) is found by iterating through several provisional population estimates ($P_{\text{CAPTURE}} \in \mathbb{N}_{\geq M}$):

$$\widehat{P}_{\text{CAPTURE}}(S, S') = \underset{P_{\text{CAPTURE}}}{\operatorname{argmax}} \, \widehat{L}_n(P_{\text{CAPTURE}}; S, S'), \tag{4.20}$$

with a given P_{CAPTURE} having the following log-likelihood, as presented in Krebs [Kre89]:

$$\begin{aligned} L_n(P_{\text{CAPTURE}}; S, S') = \\ \ln\left(\frac{P_{\text{CAPTURE}}!}{(P_{\text{CAPTURE}} - M_T(S, S'))!} \right) + C_{\text{total}}(S, S') \times ln\Big(C_{\text{total}}(S, S')\Big) \\ + \Big(T \times P_{\text{CAPTURE}} - C_{\text{total}}(S, S')\Big) \times \ln\Big(T \times P_{\text{CAPTURE}} - C_{\text{total}}(S, S')\Big) \\ - (T \times P_{\text{CAPTURE}}) \ln(T \times P_{\text{CAPTURE}}), \end{aligned} \tag{4.21}$$

where "!" represents the factorial operation.

Since we iterate through all samples, the total number of captures consists of: (1) the number of samples in S and S', (2) their respective neighbors, and (3) the number of samples from each set inside the hyperspheres of any sample from the other set. Hence, the total number of captures (C_{total}), is calculated as:

$$\begin{aligned} C_{\text{total}}(S, S') = &\sum_{s \in S} \sum_{s' \in S'} \Big(f(s', \text{NN}_k(s, S)) + |\text{NN}_k(s, S)| \Big) \\ &+ \sum_{s' \in S'} \sum_{s \in S} \Big(f(s, \text{NN}_k(s', S')) + |\text{NN}_k(s', S')| \Big), \end{aligned} \tag{4.22}$$

with f defined in (4.10) and NN_k returning the set of k-nearest neighbors of a given sample, as introduced earlier in this Section.

Family of metrics

We combine the described estimations in one family of metrics: Mark-Evaluate (ME). We use the accuracy loss (4.9) of each population estimation to compute the final score of their corresponding metric, namely $\text{ME}_{\text{Petersen}}$, $\text{ME}_{\text{Schnabel}}$, or $\text{ME}_{\text{CAPTURE}}$. Each score is then calculated by:

$$\text{ME}_{\{\text{Petersen, Schnabel, CAPTURE}\}}(S, S') = 1 - A\Big(P, \widehat{P}_{\{\text{Petersen, Schnabel, CAPTURE}\}}(S, S')\Big). \tag{4.23}$$

We note that $\text{ME}_{\text{Schnable}}$ may be used as a double-valued metric due to its iterative nature, resulting in a different population estimate depending on which sets are used as the reference and evaluation sets. Hence, given a real set S_r and a generated set S_g, sample quality and sample diversity may be measured by $\text{ME}_{\text{Schnable}}(S_r, S_e)$ and $\text{ME}_{\text{Schnable}}(S_e, S_r)$, respectively. On the other hand, since the population estimations of both $\widehat{P}_{\text{Petersen}}$ and $\widehat{P}_{\text{CAPTURE}}$ are independent of the set assignments, we have $\text{ME}_{\text{Petersen}}(S_r, S_e) = \text{ME}_{\text{Petersen}}(S_e, S_r)$ and $\text{ME}_{\text{CAPTURE}}(S_r, S_e) = \text{ME}_{\text{CAPTURE}}(S_e, S_r)$. Hence, both $\text{ME}_{\text{Petersen}}$ and $\text{ME}_{\text{CAPTURE}}$ may only be used as singe-valued metrics.

4.3.3 Experimental Results

Now, we briefly validate the different metrics on MNLI [WNB18], which contains sentences from different topics. We use 10K training sentences and 10K validation sentences as our reference and generated sets, respectively. Each sentence is represented by embeddings from SBERT [RG19]. Namely, we are interested in analyzing the convergence of the different metrics to their maximum scores when significantly augmenting the capture volumes by increasing k.

Number of neighbors

Results with $k \in \{1, \ldots, 40\}$ are presented in Figure 4.7.

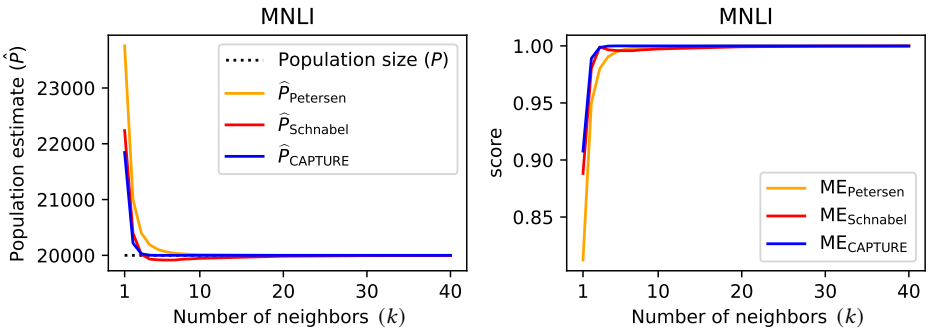


Figure 4.7: Effects of using different values for k, which directly impacts the size of the used hyperspheres. The population size is 20K, since we use 10K reference and 10K evaluation samples. The population estimates and scores of the different metrics are presented in the left and right figures, respectively, converging to the expected values as k increases.

We observe that the increase in the hypersphere volumes by increasing k results in the estimated population size of each metric to converge to the true population size. Such convergence translates to a maximum score of 1 in all metrics. We note that different metrics present different behaviors, especially at low values for k.

4.3.4 Method Comparisons

We now compare Mark-Evaluate to the existing metrics described in Section 4.1. Namely, we evaluate drops in sample diversity and sample quality by syntheti-

cally inducing mode collapse and random word swaps, respectively, as introduced by Semeniuta et al. [SSG18]. Moreover, we assess language generation by evaluating the generated text from several language models. Finally, we evaluate machine translation and text summarization models in terms of correlation to human evaluation.

We start with synthetic experiments on the previously mentioned MNLI dataset [WNB18], which contains more than 400K sentence pairs across five different topics. We note that we discard the pair information since we are only interested in each sentence and its respective topic for our experiments. Hence, the same number of individual sentences from the training and validation sets are treated as our real and generated sets, respectively. Similar to FTI's experiments in Section 4.2.4, we manipulate the validation or generated set to stimulate different levels of sample quality or diversity. Each sentence is represented using SBERT embeddings from a BERT-base model trained on SNLI [Bow+15] and MNLI [11].

Mode collapse

We gradually remove sentences from each topic in the validation (generated) set to simulate different degrees of mode collapse. No modifications were made in the training (real) set, which contains sentences from all five topics. We kept the sizes of the modified validation sets constant, assessing 4K validation and 4K real samples throughout the entire experiment. Comparison results to PRD [Saj+18], IMPAR [Kyn+19] and FID [Heu+17] are presented in Figure 4.8.

We observe that our single-valued metrics, $\mathrm{ME}_{\mathrm{Petersen}}$ and $\mathrm{ME}_{\mathrm{CAPTURE}}$, show higher deterioration as more sentence topics are dropped, as expected. Similar behavior is also observed by FID. Our double-valued metric, $\mathrm{ME}_{\mathrm{Schnabel}}$, shows a drop in sample diversity, as expected, and a generally constant behavior in terms of sample quality until up to four topics dropped. Then, at four dropped topics, there is a sudden quality assessment shift of ≈ 0.5, suggesting high sensitivity. We note that IMPAR shows a similar behavior, masked by the different y-scales, decreasing its quality assessment by ≈ 0.4 under the same scenario. PRD shows contradictory behaviors to what is expected, as previously observed in Section 4.2.4.

11 https://github.com/UKPLab/sentence-transformers

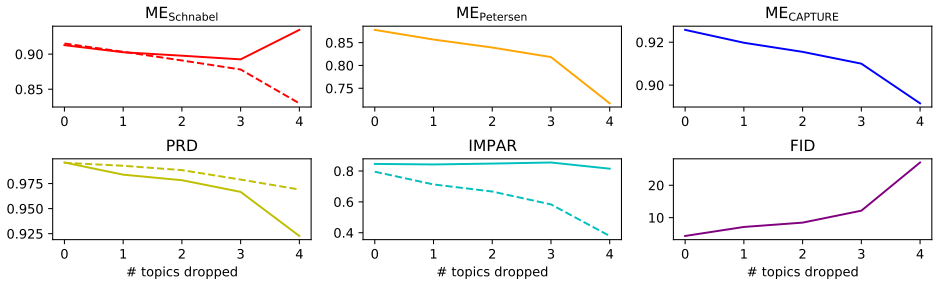


Figure 4.8: Comparison results while dropping sentences from certain topics to simulate mode collapse. For the double-valued metrics, quality and diversity assessments are represented as full lines and dotted lines, respectively.

Word swap

For the simulation of different levels of quality detriment, we randomly swap each word of each validation sentence based on a range of swap probabilities. We use 10K training (real) samples and 10K validation (generated) samples. We note that the training set remains unchanged throughout all steps, similarly to the previous experiment. Comparison results against the same previous methods are presented in Figure 4.9.

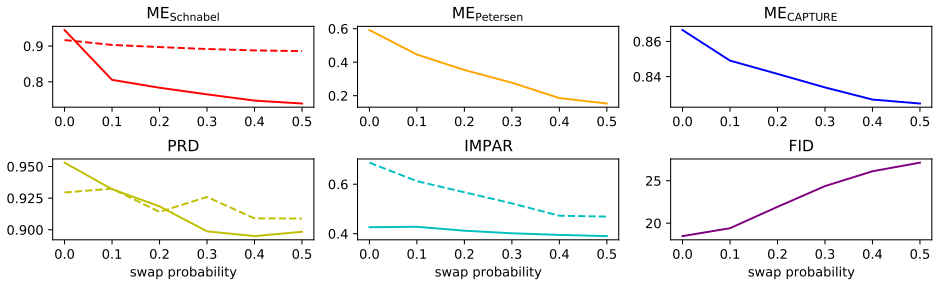


Figure 4.9: Comparison results while swapping words with a certain swap probability. For the double-valued metrics, quality and diversity assessments are represented as full lines and dotted lines, respectively.

We observe that ME$_{Schnabel}$ showcases the expected behavior, with its diversity assessment remaining close to constant and its quality assessment dropping as more words are (likely to be) swapped. On the other hand, PRD and IMPAR's diversity assessments show instability and consistent decay, respectively, which

is not expected. While PRD's quality assessment decays at a fast rate, at least at lower swap probabilities, IMPAR's remains almost constant throughout the entire experiment. All single-valued metrics show the desired behavior, deteriorating as the quality detriment increases. As a side note, we point out the success of using SBERT embeddings to detect quality changes, which has previously been shown to be a challenge using other embedding schemes [SSG18].

Language generation

From now on, we focus on the assessment of generated text from existing models. We start by measuring the correlation to human evaluation of the text models studied by Cífka et al. [Cíf+18]. Namely, we assess a traditional language model as well as different autoencoders: variational, adversarial (regularized or not), and plain. The human assessment was entirely based on the fluency of the generated text; further information may be found in their work [Cíf+18]. Hence, for the assess double-valued metrics, we study the correlation only in terms of quality assessment. On top of the previously compared metrics, we also use the reverse and forward cross-entropy (CE) calculated from a language model pre-trained on English Gigaword [NGV12] presented by Cífka et al. [Cíf+18]. To represent each sentence, we used SBERT embeddings from a BERT-large model pre-trained on SNLI and MNLI due to their superior performance [RG19]. Pearson, Kendall, and Spearman correlations are presented in Table 4.1.

Table 4.1: Absolute human evaluation correlations in terms of fluency of 10 different text generation models. We note that Mark-Evaluate, IMPAR, PRD results are reported based on their best k, obtained before correlations start to drop. The highest overall correlations are underlined, whereas our metrics' correlations that match or outperform all compared methods are represented in bold.

Correlations	Forward CE	Reverse CE	FID	PRD	IMPAR	**$ME_{Schnabel}$**	**$ME_{Petersen}$**	**$ME_{CAPTURE}$**
Pearson	0.606	0.440	0.902	0.830	0.745	**<u>0.917</u>**	0.872	**0.902**
Kendall	0.556	0.333	0.867	0.822	0.778	**<u>0.911</u>**	**<u>0.911</u>**	**0.867**
Spearman	0.697	0.491	0.964	0.939	0.903	**<u>0.976</u>**	**<u>0.976</u>**	**0.964**

We observe that, in all scenarios, at least one of our metrics achieves a higher human evaluation correlation than all compared methods. Even though $ME_{Petersen}$ is slightly outperformed by FID in terms of Pearson correlation,

we note that we still outperform both PRD and IMPAR. The compared cross-entropy metrics, which were commonly used to evaluate text generation in the past [Cíf+18; SSG18; Zha+18b], show the least correlation to human evaluation compared to more recent, embedding-based metrics.

We now focus on the evaluation of conditional language generation, where a generated text may be directly compared to one or more reference texts. Common examples of conditional tasks are machine translation and text summarization, which will be evaluated next. Since we are now interested in analyzing shorter generated scripts, which may only consist of a few words over one sentence, we require a finer-grained representation. Hence, we propose to use contextualized word embeddings from BERT-base fine-tuned on MNLI, following MoverScore [Zha+19b], which is one of our compared metrics.

More specifically, we propose to use several embeddings to represent each word, namely the embeddings of the last five layers of BERT-base. This is motivated by the effectiveness of using latter BERT layers on downstream tasks [Liu+19; Zha+20a]. Moreover, by increasing the overall population size, the estimation of our different methods is likely to improve as a result. A comparison between the proposed and existing representation schemes is illustrated in Figure 4.10.

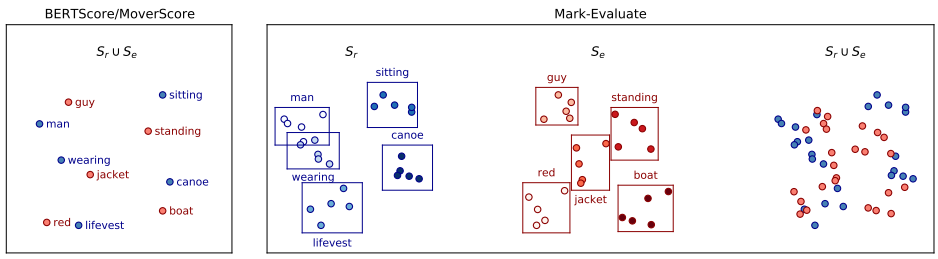


Figure 4.10: Considering a reference (S_r) and evaluation (S_e) word set, we represent each word by the five embeddings from the last five BERT-base layers. Existing methods, such as BERTScore and MoverScore, adopt a single representation for each word by using either the embeddings of a given layer or embedding aggregation schemes over several layers, respectively.

In the following experiments, we also use our proposed embedding representations for PRD and IMPAR. Moreover, all methods use BERT-base embeddings. Due to the likely discrepancy in the number of reference and evaluation words, we only use the sample quality assessment for double-valued metrics. For the

clustering-based metrics (Mark-Evaluate, PRD, and IMPAR), the reported results use the best value for k obtained before the performance started to drop.

Machine translation

We start by assessing machine translation from several systems provided by Bojar et al. [BGK17] in the WMT17 metrics task. We followed MoverScore's implementation [12] and evaluated around 3K Czech (cs), German (de), Russian (ru), Turkish (tr), and Chinese (zh) to English (en) translations. We use the embeddings from the fifth layer of BERT-base for BERTScore, with higher correlations being possibly obtained by grid-searching over all layers. Pearson human evaluation correlations are presented in Table 4.2.

Table 4.2: Pearson correlations to human evaluation on different language pairs of systems from the WMT17 metrics task. The highest correlations of each language pair are underlined, whereas our metrics' correlations that match or outperform all compared methods are represented in bold.

Translations	BERTScore	MoverScore	PRD	IMPAR	**ME$_{\text{Schnabel}}$**	**ME$_{\text{Petersen}}$**	**ME$_{\text{CAPTURE}}$**
cs-en	0.966	0.983	0.992	0.987	0.989	0.988	0.987
de-en	0.859	0.920	0.769	0.934	**0.944**	**0.953**	**0.953**
ru-en	0.868	0.921	0.933	0.896	0.902	0.908	0.908
tr-en	0.938	0.931	0.935	0.959	**0.970**	**0.960**	**0.959**
zh-en	0.894	0.943	0.889	0.933	**0.957**	0.936	0.936
Average	0.905	0.940	0.904	0.942	**0.952**	**0.949**	**0.949**

We observe that at least one of our proposed metrics achieves the highest human evaluation correlation on the majority of language pairs compared to existing metrics. Moreover, when averaging the correlations across all language pairs, all of our metrics outperform the rest. Considering only the compared metrics, PRD achieves the highest correlation, validating the importance of integrating existing metrics in different data domains.

Text summarization

We also evaluated the correlation to human evaluation on text summarization of news articles using the TAC-2009 dataset[13]. Each article, related to 1 out of 10

12 https://github.com/AIPHES/emnlp19-moverscore
13 http://tac.nist.gov/

topics, has 4 reference and 55 generated summaries. The human evaluation is separated into two assessments: the pyramid (pyr.) and the responsiveness (resp.) score. While the first focuses on semantic similarities between the reference and generated summaries, the latter analyzes the overall grammar and content quality of the generated summaries. Kendall, Pearson, and Spearman correlations are presented in Table 4.3.

Table 4.3: Kendall, Pearson, and Spearman correlations to human evaluation at the summary-level on TAC 2009. We evaluate correlations based on two human assessments in the form of the responsiveness and pyramid scores. The highest correlations of each score are underlines, whereas our metrics' correlations that match or outperform all compared methods are represented in bold.

	Correlations	BERTScore	MoverScore	PRD	IMPAR	$\mathbf{ME_{Schnabel}}$	$\mathbf{ME_{Petersen}}$	$\mathbf{ME_{CAPTURE}}$
Resp.	**Kendall**	-	0.482	0.398	0.481	**0.483**	**0.487**	**0.484**
	Pearson	0.739	0.754	0.564	0.743	0.739	0.683	0.747
	Spearman	0.580	0.594	0.501	0.594	**0.595**	**0.598**	**0.596**
Pyr.	**Kendall**	-	0.550	0.444	0.541	0.548	**0.555**	**0.565**
	Pearson	0.823	0.831	0.658	0.804	0.813	0.770	0.808
	Spearman	0.703	0.701	0.588	0.693	0.698	**0.704**	**0.718**

We observe that, in both the pyramid and responsiveness scores, at least one of our metrics shows higher Kendall and Spearman correlations than all compared metrics. Moreover, all of our metrics show a higher Pearson correlation than PRD, whereas IMPAR is consistently outperformed by at least one of our metrics. Overall, MoverScore shows the highest Pearson correlations on both scores. The reported BERTScore correlations were the ones reported by Zhao et al. [Zha+19b].

4.4 Concluding Remarks

In this Chapter, we introduced two new evaluation methods: FTI, which uses topological representations and fuzzy logic to assess image generation, and Mark-Evaluate, a family of metrics based on population estimation methods for assessing text generation. Overall, the proposed metrics provide either single-valued or double-valued assessments, which may be preferred in certain

application scenarios. In the end, the assessment improvement by the proposed work helps shed light on how to improve existing generation processes, like the generator model in GANs, on a variety of tasks.

Ideally, evaluation metrics should be general and applicable to different tasks and data domains. Using the embeddings of a given pre-trained model to represent the real and generated samples is a promising step in this direction. Hence, both FTI and Mark-Evaluate may be applied to different application scenarios by appropriately changing the pre-trained model used to obtain the sample embeddings.

5 Evaluation of Compressed GANs

This Chapter is based on one of our papers regarding evaluating the compression effects of the generator model in GANs [MYM21], which is currently under review at the time of this writing. Overall, this Chapter sheds some light on the connection between the discussed topics throughout this thesis.

The ascending trend of computation and memory demands in GANs reinforces the need for post-training compression of the generator model. Importantly, a thorough analysis of how different compression levels affect the generated set is essential to the efficient employment of GANs in real-world applications. In this Chapter, we compress the weights of popular GANs, particularly the generator model, to low bit-widths using different compression techniques. Then, we study the consequent compression effects by proposing two new, outlier-aware metrics based on locality-sensitive hashing (LSH).

In Section 5.1, we introduce recent efforts on compressing GANs during training. In Section 5.2, we propose our new metrics and analyze the post-training compression effects on popular GANs. Finally, in Section 5.3, we present some concluding thoughts and future directions.

5.1 Related Work

Compression of GANs during training has been recently shown to increase the inherent training instability of GANs, previously discussed in Chapter 2, often resulting in a reduction of sample quality or extended mode collapse [Che+20; Li+20; Shu+19; Wan+19b; YP20]. On top of the training of multiple models in GANs, the high entropy of the generator's input and output may also present an obstacle for the successful compression of G alone [YP20]. Next, we describe existing methods proposed to compress the generator (and also occasionally the discriminator) during training.

Evolutionary algorithms for channel pruning have been applied to compress the generator [Shu+19]. Knowledge distillation [HVD15] may also be used to

compress a student generator to indirectly learn from a larger teacher generator and discriminator [Agu+19]. Namely, an additional student generator may be added to the previous framework to enable higher compression levels of the student generator [Che+20]. Additionally, one may directly learn a student generator from a pre-trained teacher generator using a learned intermediate representation training (LIT) [Kor+19].

On a different line of work, the negative learning effects of compressing both the generator and the discriminator may be reduced using Expectation-Maximization compression algorithms [Wan+19b]. Moreover, Liu et al. [Liu+20] showed that improved training stabilization may also be achieved if quantizing more layers of the discriminator than the generator. Yu et al. [YP20] used self-supervised learning by leveraging a pre-trained generator and discriminator to successfully train a pruned generator. Extending the original GANs objective to account for knowledge distillation, channel pruning, and quantization may also be a viable alternative [Wan+20]. Finally, Li et al. [Li+20] used neural architecture search [EMH19] to find the best-compressed generator candidates in conditional GANs.

All of the previous methods focus on developing new techniques to compressing GANs during training. However, to the best of our knowledge, the study of the efficacy of existing compression methods to compress G after training and without any fine-tuning remains an open question at the time of this writing. The several benefits of post-training compression previously discussed in Chapter 3, namely the low computational cost and no training data restrictions, may also translate to GANs. Moreover, if no additional training is performed to the compressed G, one would not need to worry about the training instability of GANs. We expand on these thoughts for the rest of this Chapter.

5.2 Assessment of the Compression Effects

We introduce locality-sensitive hashing for assessing a generated set in Section 5.2.1 and detail our two novel LSH-based metrics and important implementation details in Section 5.2.2. Moreover, we apply compression techniques to popular GANs and present a preliminary discussion based on qualitative results in Section 5.2.3. Finally, in Section 5.2.4, we compare our LSH-based metrics to existing evaluation methods in the context of compression in GANs.

5.2.1 Locality-Sensitive Hashing

As previously discussed in Chapter 4, reference and evaluation data manifolds may be approximated using KNN, as originally proposed by Kynkäänniemi et al. [Kyn+19] and further adopted by our evaluation methods, FTI and Mark-Evaluate. However, considering the assessment of one evaluation sample, the manifold approximation process has linear complexity on the number of reference samples, since one has to iterate through the entire reference set to obtain an approximation of the reference manifold. Moreover, standard KNN approximations are highly sensitive to outlier samples in both the reference and evaluation sets, which may greatly distort the manifold approximation, negatively affecting the overall assessment of precision and/or recall. This may also be observed with highly sparse data, with the resulting hypersphere to that k-th nearest neighbor covering a large, unpopulated portion of the data space. Dealing with such (outlier) samples is crucial for a proper assessment of the compression effects in GANs since the originally generated set is likely to become distorted at high compression levels.

Alternatives to KNN

One may improve the computational cost of KNN by considering only a subset of samples. (We note that performing sample dimensionality reduction is also a viable option but it is outside the scope of this thesis.) Suitable sample filtering methods depend on the nature of the data, *e.g.* K-D trees [Ben75] are used for low-dimensional, continuous data, whereas inverted lists may be applied to high-dimensional, discrete data. In our use case, our data is both high-dimensional and continuous since each sample consists of the feature representations of a pre-trained VGG-16 model [LD15], as initially proposed by Kynkäänniemi et al. [Kyn+19]. Hence, on top of fitting the nature of our data, locality-sensitive hashing (LSH) is an ideal solution since it also grants the ability of discarding (likely farther) neighbors, which may act as an outlier-prevention mechanism.

LSH-based metrics

To tackle the computational expensiveness and outlier sensitiveness of existing KNN-based metrics, we propose two new metrics based on LSH. At their core, they rely on splitting the data space into different regions by using random hyperplanes. The intuition is that outlier samples are likely isolated in a certain

region, reducing their possible negative assessment effects. Our simplest metric only uses LSH to directly approximate the reference or evaluation manifolds with the regions that contain reference or evaluation samples, respectively. On the other hand, our follow-up metric uses both LSH and KNN to approximate the manifolds but only compares samples inside a certain region. A comparison of the different metrics is illustrated in Figure 5.1.

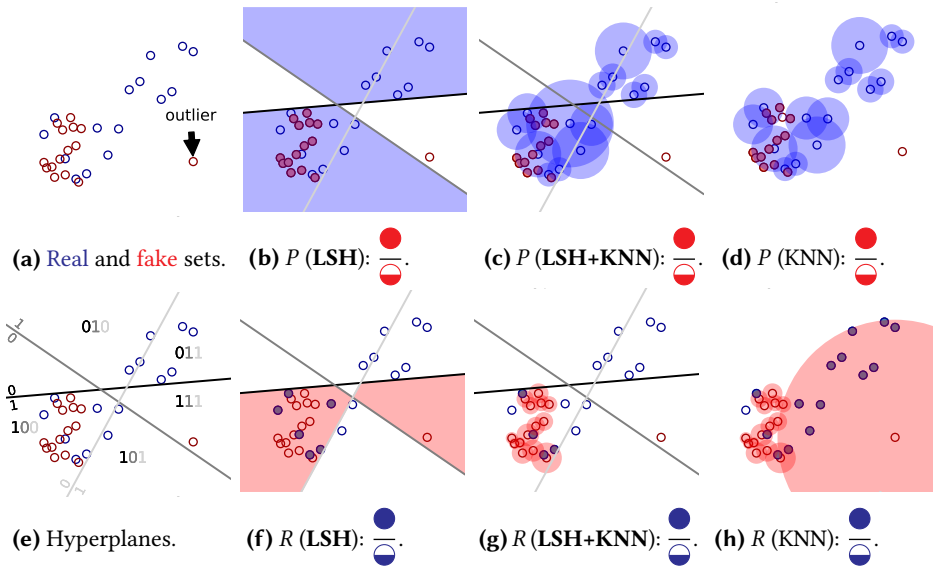


Figure 5.1: Comparison of different metrics in terms of precision (P) and recall (R), calculated using the ratios in the fake and real sets, respectively. Considering a set of real and generated samples (a), we first split the data space into several regions by generating random hyperplanes (e). Each region is represented by in which side it lies regarding each hyperplane, indicated by the binary sequences. Using solely LSH, we consider samples inside the same region to calculate precision and recall (b, f). Using both LSH and KNN, we only consider the hyperspheres to the k-nearest sample ($k = 1$ in this example) within each region to approximate the manifold (c, g). On the other hand, standard KNN approaches, *e.g.* IMPAR [Kyn+19], consider all samples to generate such hyperspheres (d, h), which may have undesirable outlier effects (h). We mitigate this by (likely) inserting outliers in isolated regions (f, g).

Generating identical keys for nearby samples allows us to reduce the search space of standard KNN by only considering samples with the same keys, *i.e.* in the same region. To achieve this, each key is generated in a simple, yet effective,

way by leveraging random projections [SC08]. For efficient storage and access, we map each key and respective samples using a hash table.

5.2.2 Implementation Details

In this Section, we present details of the proposed steps. Namely, we specify how keys are generated and how that information may be used to assess a generated set using the proposed double-valued metrics. Moreover, we present a computational complexity analysis, showcasing the efficiency of our metrics.

Random projections

We follow the random projection procedure presented by Slaney et al. [SC08], and generate H random d-dimensional hyperplanes formed by a random vector $h \sim \mathcal{N}_d(0,1)$ and a random variable $b \sim \mathcal{U}(0,1)$:

$$h_0 x_0 + \ldots + h_{d-1} x_{d-1} + b = 0. \tag{5.1}$$

We use such random hyperplanes to construct a key of a d-dimensional sample $\phi \in \Phi$. More specifically, each bit of ϕ's key represents in which side of a given hyperplane ϕ lies on, obtained according to the sign of the following dot product:

$$\text{hash}_{h,b}(\phi) = \begin{cases} 1, & \text{if } \underbrace{\text{sign}(\phi \cdot h + b)}_{\in\{-1,0,1\}} \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

After iterating across all H hyperplanes, ϕ's key will have H bits:

$$\text{key}_H(\phi) = \text{hash}_{h_0,b_0}(\phi) \ldots \text{hash}_{h_{H-1},b_{H-1}}(\phi), \tag{5.3}$$

where each bit may be computed in parallel by independently generating $h_{0,\ldots,H-1}$ and $b_{0,\ldots,H-1}$. Considering an entire set of samples Φ, a given sample ϕ's key may also be generated in parallel. We store this information in a hash table HT_Φ, where each unique key is mapped to the list of samples in Φ that share that same key.

Precision and recall assessment

Each of our proposed metrics retrieves a double-valued assessment related to precision (sample quality) and recall (sample diversity). We start by discussing our simplest and most efficient metrics, which uses solely LSH without KNN. In this variant, we calculate precision as the probability of a generated sample having at least one real sample in its region. Conversely, we calculate recall as the probability of a real sample having at least one generated sample in its region.

For this purpose, let us first define the function f_{LSH} that returns whether or not a key is in an initialized hash table HT_{Φ} with:

$$f_{\text{LSH}}(\text{key}, \text{HT}_{\Phi}) = \begin{cases} 1, & \text{if key} \in \text{HT}_{\Phi} \\ 0, & \text{otherwise.} \end{cases} \tag{5.4}$$

Given two sample sets Φ_a and Φ_b, we calculate the average probability of a sample $\phi_a \in \Phi_a$ being in the region of at least one sample $\phi_b \in \Phi_b$. More precisely, we simply check if ϕ_a's key exists in the initialized hash table of Φ_b (HT_{Φ_b}):

$$p_{\text{LSH}}(\Phi_a, \Phi_b) = \frac{1}{|\Phi_a|} \sum_{\phi_a \in \Phi_a} f_{\text{LSH}}(\text{key}_H(\phi_a), \text{HT}_{\Phi_b}). \tag{5.5}$$

Using our LSH metric variant, and considering a real set S_r and a generated set S_g, we calculate precision (P_{LSH}) and recall (R_{LSH}) as:

$$P_{\text{LSH}} = p_{\text{LSH}}(S_g, S_r) \qquad R_{\text{LSH}} = p_{\text{LSH}}(S_r, S_g). \tag{5.6}$$

For our other metric, we use KNN on top of LSH to improve the manifold approximation. In this variant, we calculate precision with the probability of a generated sample being in the hypersphere of at least one real sample inside its region. Conversely, recall is calculated with the probability of a real sample being in the hypersphere of at least one generated sample inside its region.

In this variant, we follow the formulations presented by Kynkäänniemi et al. [Kyn+19] to define whether or not a sample ϕ is inside the hypersphere of at least one sample $\phi' \in \Phi$, calculated to its k-nearest neighbor in the set Φ by $\text{NN}_k(\phi', \Phi)$:

$$f_{\text{LSH+KNN}}(\phi, \Phi) = \begin{cases} 1, & \text{if } \|\phi - \phi'\|_2 \leq \|\phi - \text{NN}_k(\phi', \Phi)\|_2 \\ & \text{for at least one } \phi' \in \Phi \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

In the circumstances where there are not enough samples in the set Φ to retrieve the k-th nearest neighbor of ϕ', *i.e.* $k \geq |\Phi|$, we adapt $\text{NN}_k(\phi', \Phi)$ to ensure that the farthest neighbor is retrieved instead.

Considering two sample sets Φ_a and Φ_b, we calculate the average probability of a sample $\phi_a \in \Phi_a$ being in the neighborhood of at least one sample $\phi_b \in \Phi_b$ in ϕ_a's region. More precisely, we retrieve the samples in Φ_b that are in the same region as ϕ_a by accessing $\text{HT}_{\Phi_b}[\text{key}_H(\phi_a)]$. Then, we check if ϕ_a is in the neighborhood of any of such samples by using (5.7):

$$p_{\text{LSH+KNN}}(\Phi_a, \Phi_b) = \frac{1}{|\Phi_a|} \sum_{\phi_a \in \Phi_a} f_{\text{LSH+KNN}}(\phi_a, \text{HT}_{\Phi_b}[\text{key}_H(\phi_a)]). \tag{5.8}$$

Finally, using our metric variant that leverages LSH and KNN, we calculate precision ($P_{\text{LSH+KNN}}$) and recall ($R_{\text{LSH+KNN}}$) as:

$$P_{\text{LSH+KNN}} = p_{\text{LSH+KNN}}(S_g, S_r) \qquad R_{\text{LSH+KNN}} = p_{\text{LSH+KNN}}(S_r, S_g). \tag{5.9}$$

Complexity analysis

We now briefly shed some light on the efficiency of the proposed methods compared to standard KNN approaches, such as IMPAR [Kyn+19]. We are interested in calculating the average computational cost of assessing a d-dimensional evaluation sample against n (d-dimensional) reference samples, using H random hyperplanes for our metrics. Efficient scaling with the growth of the reference set is crucial since a higher n is likely to lead to a better overall assessment. The complexities of the different metrics are presented in Table 5.1.

For both of our metrics, determining the region of a d-dimensional sample costs $d \times H$, originated from performing one dot product per hyperplane, as discussed earlier in this Section. Moreover, since H hyperplanes divide the data space into $\approx 2^H$ regions, we may expect, on average, $n/2^H$ samples in each region. Hence, in our LSH with KNN variant, the comparison cost between an evaluation

Table 5.1: Average computational cost and complexity of the different metrics.

Evaluation method	Computational cost	Complexity in n
LSH (ours)	$d \times H$	$O(1)$
LSH + KNN (ours)	$d \times H + d \times n/2^H$	$O(\log(n))$, if $H \sim \log(n)$
KNN [Kyn+19]	$d \times n$	$O(n)$

sample and the rest of the samples in its region is $d \times n/2^H$. Note that, with $H \sim \log(n)$, the previous comparison cost becomes independent of n. Hence, our LSH-based variants with and without KNN have a constant and logarithmic complexity on the number of reference samples, respectively. On the other hand, existing (standard) KNN metrics have linear complexity since one must iterate through all reference samples. Figure 5.2 shows how the different metrics scale with an increasing number of reference samples on existing hardware.

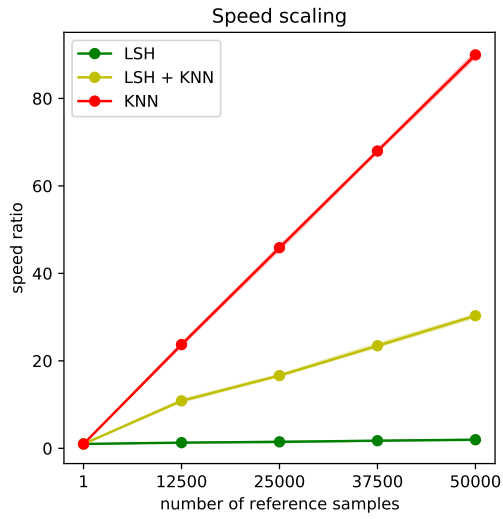


Figure 5.2: Speed scaling analysis of the different evaluation methods using one NVIDIA GeForce GTX 1080 Ti GPU. We report the average over three evaluation runs using 50K evaluation samples and a varying amount of reference samples.

5.2.3 Experimental Results

In this Section, we present qualitative results regarding the effects of compressing the weights of pre-trained generators in popular GANs after training and without any fine-tuning. We study several post-training compression schemes [14] mainly applied to image classification models in the past [MVK19; Zha+19a]:

- Linear quantization **(Q)**, which normalizes by the absolute maximum weight and assumes symmetric dynamic range [Wan+19b; Zha+19a].
- Pruning and quantization **(P + Q)**, which represents our method Monte Carlo Quantization or MCQ. As previously presented in Chapter 3, the sampling nature of MCQ allows for both quantization and pruning by discarding non-hit weights.
- Splitting and quantization **(S + Q)**, which leverages Outlier Channel Splitting (OCS) [Zha+19a] to split and duplicate outlier weights to reduce the dynamic range. (After splitting, we apply linear quantization.)
- Clipping and quantization **(C + Q)**, which uses Analytical Clipping for Integer Quantization (ACIQ) [BNS19] to find an optimal clipping threshold between the continuous and discrete weight distributions. (After clipping, we apply linear quantization.)

These compression schemes rely on different assumptions and steps to quantize the weights of a given layer. The quantized assignments using linear quantization are sensitive to outlier weights due to the usage of the full dynamic range of the continuous weights. On the other hand, by abruptly reducing the dynamic range with clipping, ACIQ is likely to distort outlier weights whereas MCQ relies on importance sampling, which may compromise the weight approximation at low sampling amounts. Finally, OCS increases the number of weights after splitting, which may be unfeasible at a high outlier rate.

We compress pre-trained generator weights of several popular GANs designs:

- PA-GAN [ZK19] progressively augments D's input to prevent overfitting during training.

14 https://github.com/cornell-zhang/dnn-quant-ocs

- zCR-GAN [Zha+20b] augments G's input by adding small-magnitude noise as a latent consistency regularization mechanism.
- SN-GAN [Miy+18] spectrally normalizes D's weights to improve learning.
- SS-GAN [Che+19] leverages self-supervised learning in G and D by introducing auxiliary rotations.
- StyleGAN2 [Kar+20b] builds on a style-based G architecture [KLA19] and proposes new mechanisms for normalization, regularization, and progressive growing of G.
- ADA [Kar+20a] extends StyleGAN2 by adaptively augmenting D to enable learning with limited data.
- BigGAN [BDS19] orthogonally regularizes a conditional G to learn from large-scale data.

We use publicly available pre-trained generators from the previous GANs trained on a variety of data [15,16,17], namely high-quality images of human faces (1024x1024 and 256x256) from Flickr-Faces-HQ (FFHQ) [KLA19] as well as cats, dogs, and wildlife faces (512x512) from Animal Faces-HQ (AFHQ) [Cho+20]. BigGAN was trained on downscaled 128x128 images from ImageNet [Rus+15].

In the following experiments, we compress the vast majority of G's weights ($\approx$ 94 to 96% depending on the architecture and data resolution), except the ones in the last residual block of each network. As discussed in Section 5.2.2, we reduce the computational complexity of our LSH-based metrics when evaluating against n reference samples by using $H = \lfloor \log(n) \rfloor$. Moreover, we follow IMPAR's recommended number of neighbors [Kyn+19] and use $k = 3$ in our LSH and KNN metric variant. We qualitatively analyze the compression effects on the different generated sets next.

Mean images

We start by analyzing the pixel mean across 10K generated images, as presented by Karras et al [Kar+20a]. Results of using different compression techniques on a StyleGAN2 pre-trained on FFHQ are presented in Figure 5.3.

15 https://tfhub.dev/deepmind/biggan-128/2
16 https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada/pretrained/
17 https://nvlabs-fi-cdn.nvidia.com/stylegan2/networks/

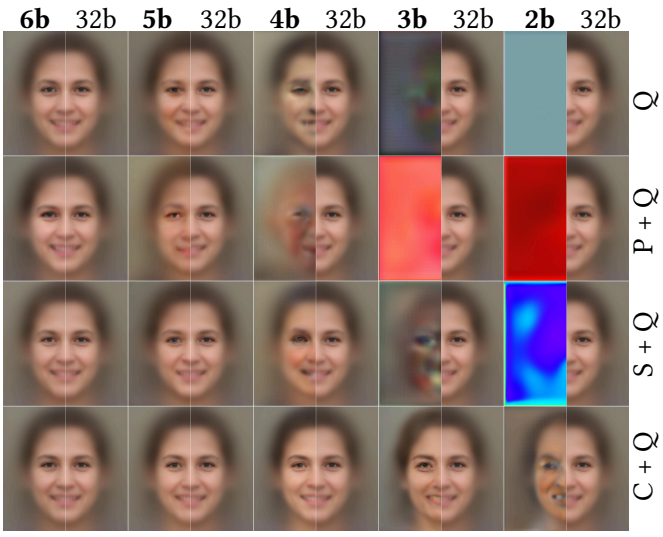


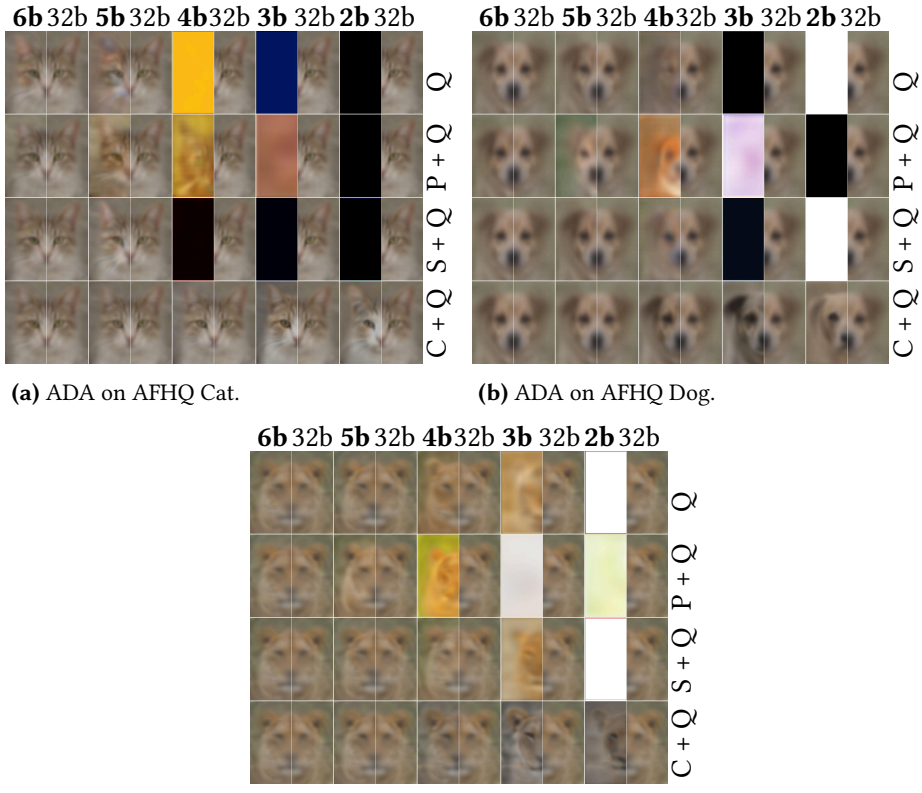
Figure 5.3: Mean 10K generated images from a StyleGAN2 generator pre-trained on FFHQ 1024x1024 and compressed to different bit-widths (from 6 to 2 bits) using different compression techniques (quantization, pruning, splitting and clipping). Mean faces generated by the 32-bit baseline are also presented for comparison.

We observe that compressing G's weights using clipping and linear quantization (C + Q) yields the best results, especially at low bit-widths. Specifically, the resemblance of the mean generated images to a human face, even at 2 bits, suggests that generative models may be less sensitive to outlier weight distortion (inherent to clipping) than discriminative models, particularly image classification models [Zha+19a]. At 5 to 6 bits, all compression techniques show negligible effects in the mean generated images compared to the 32-bit baseline. Similar conclusions apply when compressing a pre-trained ADA model on AFHQ, as observed in Figure 5.4.

We now also extend the previous experiments to analyze how compression affects different GANs that were trained on the same data: FFHQ 256x256. Due to its superior performance, we only report compression results using clipping and linear quantization (C + Q) from here on. The mean generated images at 2-bit compression rates and respective 32-bit baselines are shown in Figure 5.5.

We observe that the different 32-bits baselines show a similar average generated face, supporting previous claims regarding the similar behaviour of different

GANs [Luc+18]. On the other hand, the generated faces at 2-bit compression rates are affected differently across different GANs with each compressed G seemingly retaining the ability to generate different face attributes. Namely, elderly features (Figure 5.5 (e)), longer hair (Figures 5.5 (a) and 5.5 (f)), darker hair (Figures 5.5 (b) and 5.5 (d)), and male characteristics (Figure 5.5 (c)).



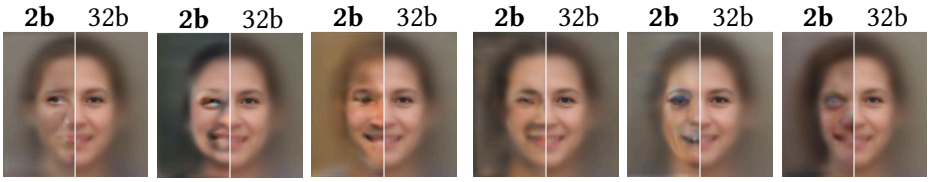
(a) ADA on AFHQ Cat.

(b) ADA on AFHQ Dog.

(c) ADA on AFHQ Wild.

Figure 5.4: Mean 10K generated images from an ADA generator pre-trained on AFHQ 512x512 and compressed to different bit-widths (from 6 to 2 bits) using different compression techniques (quantization, pruning, splitting and clipping). Mean faces generated by the 32-bit baseline are also presented for comparison.

In general, we would like to point out that the overall face structure remains, for the most part, intact both on FFHQ and AFHQ even at higher compression rates. In other words, the compressed generators still manage to generate face-



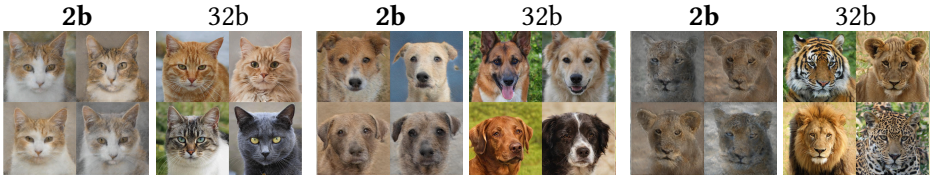
(a) ADA. **(b)** StyleGAN2. **(c)** zCR-GAN. **(d)** SN-GAN. **(e)** PA-GAN. **(f)** SS-GAN.

Figure 5.5: Mean 10K generated images from generators of different GANs pre-trained on FFHQ 256x256 and compressed to 2 bits using clipping and linear quantization (C + Q). Mean faces generated by the different 32-bit baselines are also presented for comparison.

like images, especially if using clipping and linear quantization. This suggests that compression negatively affects the generated set more in terms of sample diversity than sample quality since the average generated face is different from the respective 32-bit baseline.

Generated images

Continuing with the previous discussion, we now visually assess the decrease in sample quality in random sets of generated images from compressed and uncompressed generators. Note that images of both sets are generated using the same seed. A comparison between the generated images from uncompressed and compressed generators (with 32 bits and 2 bits weights, respectively), pre-trained on AFHQ are presented in Figure 5.6.

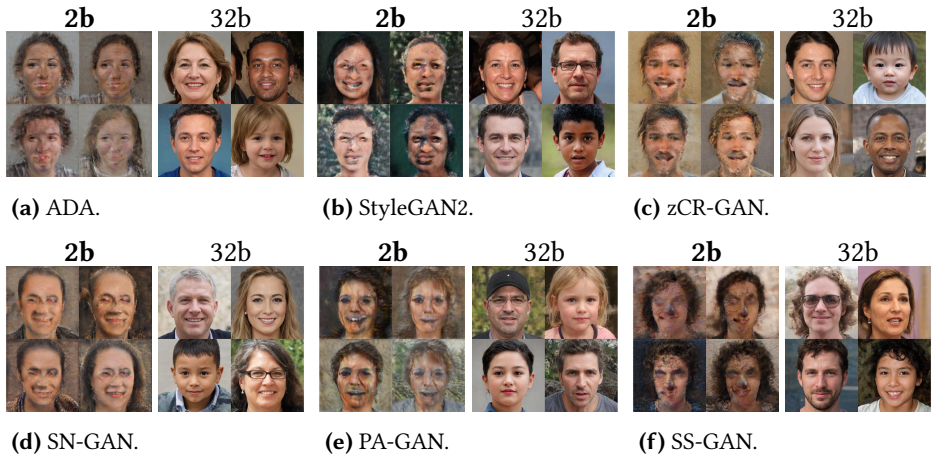


(a) AFHQ Cat. **(b)** AFHQ Dog. **(c)** AFHQ Wild.

Figure 5.6: Generated images from ADA generators on AFHQ 512x512 and compressed to 2 bits using clipping and linear quantization (C + Q). Images generated by the 32-bit baselines are also presented for comparison.

We observe a lack of sample diversity in the generated samples from the compressed generators compared to the respective 32-bit generator baselines.

However, sample quality remains high, with the 2-bit generators being able to generate realistic images of cats, dogs, and wildlife. Similar compression effects are observed in Figure 5.7 with different GANs pre-trained on FFHQ.



(a) ADA. **(b)** StyleGAN2. **(c)** zCR-GAN.

(d) SN-GAN. **(e)** PA-GAN. **(f)** SS-GAN.

Figure 5.7: Generated images from different generators pre-trained on FFHQ 256x256 and compressed to 2 bits using clipping and linear quantization (C + Q). Images generated by the different 32-bit baselines are also presented for comparison.

The generation effects from different compression-levels on BigGAN generators pre-trained on ImageNet are shown in Figure 5.8. We observe that, as compression increases, the depicted objects start becoming increasingly blurry and start adopting identical textures.

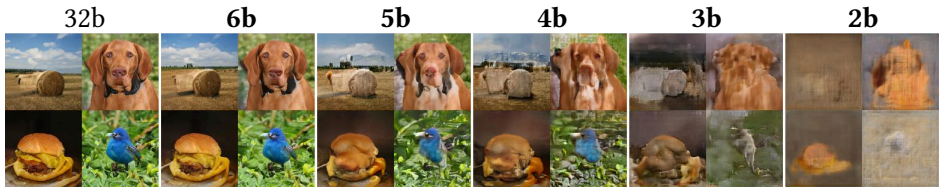


Figure 5.8: Generated images from a BigGAN generator pre-trained on ImageNet 128x128 and compressed to different bit-widths (6 to 2 bits) using clipping and linear quantization (C + Q). Images generated by the 32-bit baseline are also presented for comparison.

5.2.4 Method Comparisons

We now analyze the compression effects quantitatively by using the proposed metrics as well as existing single-valued (FID [Heu+17] and KID [Bińkowski+18]), and double-valued KNN metrics (IMPAR [Kyn+19]). Due to the inherent bias of FID with small reference sets [Kar+20a], we use KID as the compared single-valued metric on AFHQ. However, to assess generated data on the bigger datasets (FFHQ and ImageNet) we compare our metrics against FID. On top of quantitatively assessing several compressed generators, we also study the correlation of single-valued metrics to precision and recall, separately. Finally, we study the Pareto frontiers (regarding precision and recall) of compressed StyleGAN2 generators and existing 32-bit style-based generator baselines.

Quantitative compression effects

Comparison metric results of assessing several low bit-width generators pre-trained on FFHQ and ImageNet are presented in Table 5.2. We observe that, in most cases, generators compressed to low bit-width (3 to 4 bits) achieve a high precision assessment by our proposed metrics as well as IMPAR and KNN. On the other hand, recall decreases with higher compression, suggesting a decrease of sample diversity in the generated sets. However, such insights may not be derived from analyzing only FID, which consistently deteriorates as compression levels increase. The same conclusions apply using KID to assess compressed ADA generators in Table 5.3.

Overall, the quantitative analysis of the different compressed GANs suggests that sample diversity is majorly affected, whereas sample quality is retained, to some extent, even at low bit-width compression. This also correlates with our qualitative analysis presented in Section 5.2.3.

FID and KID correlations to precision and recall

Due to the ineffectiveness of both FID and KID to detect the preservation of sample quality in the compressed generators, we now study their separate correlation to precision and recall. Since both lower FID and KID values represent a better assessment of the generated set, we expect to see negative correlations: as precision or recall increases, FID or KID should decrease, and vice-versa. For the correlation measurements, we calculate the Pearson and Spearman correlations, which evaluate the linear and monotonic relationships of the different metric

Table 5.2: Assessment using FID and different precision (P) and recall (R) metrics on FFHQ (1024x1024 and 256x256) and ImageNet. We used 50K real and generated samples and present the average over 3 evaluation runs for each metric. For our metrics as well as KNN, a higher score is better. For the distance-based metric, FID, a lower score is better.

Network	b	LSH ↑		LSH + KNN ↑		KNN [Kyn+19] ↑		FID [Heu+17] ↓
		P	R	P	R	P	R	
StyleGAN2 (FFHQ-1024)	32	0.949	0.934	0.830	0.766	0.689	0.493	2.8
	4	**0.958**	0.913	**0.881**	0.743	**0.747**	0.298	12.0
	3	0.945	0.847	**0.869**	0.512	0.676	0.038	50.4
	2	0.561	0.400	0.052	0.026	0.091	0.000	157.5
ADA (FFHQ-256)	32	0.929	0.919	0.763	0.784	0.681	0.442	3.8
	4	**0.941**	0.906	**0.811**	0.701	**0.740**	0.285	10.1
	3	**0.969**	0.815	**0.896**	0.446	**0.808**	0.048	40.4
	2	0.751	0.651	0.280	0.135	0.307	0.000	145.9
StyleGAN2 (FFHQ-256)	32	0.933	0.917	0.764	0.772	0.682	0.445	3.7
	4	**0.940**	0.902	**0.813**	0.715	**0.802**	0.220	14.4
	3	**0.965**	0.838	**0.900**	0.595	**0.891**	0.043	40.2
	2	0.730	0.626	0.352	0.229	0.371	0.000	143.4
zCR-GAN (FFHQ-256)	32	0.934	0.918	0.769	0.741	0.680	0.473	3.3
	4	**0.960**	0.893	**0.861**	0.696	**0.766**	0.300	12.6
	3	**0.955**	0.813	**0.876**	0.492	**0.828**	0.043	65.2
	2	0.452	0.544	0.106	0.062	0.189	0.000	157.4
SN-GAN (FFHQ-256)	32	0.932	0.916	0.772	0.693	0.717	0.383	4.5
	4	**0.936**	0.894	**0.819**	**0.694**	**0.744**	0.208	14.1
	3	**0.956**	0.840	**0.866**	0.442	**0.727**	0.032	49.1
	2	0.612	0.634	0.246	0.133	0.206	0.000	140.7
PA-GAN (FFHQ-256)	32	0.932	0.918	0.764	0.693	0.683	0.449	3.8
	4	**0.957**	0.887	**0.863**	0.691	**0.789**	0.260	13.5
	3	**0.969**	0.812	**0.900**	0.454	**0.797**	0.041	50.2
	2	0.425	0.376	0.071	0.027	0.214	0.000	164.8
SS-GAN (FFHQ-256)	32	0.934	0.916	0.772	0.733	0.686	0.428	4.2
	4	**0.945**	0.897	**0.827**	**0.787**	**0.755**	0.254	11.4
	3	**0.967**	0.804	**0.887**	0.427	**0.773**	0.041	44.2
	2	0.324	0.440	0.039	0.092	0.035	0.000	197.2
BigGAN (ImageNet)	32	0.748	0.715	0.505	0.270	0.858	0.149	10.8
	4	**0.816**	0.588	**0.677**	0.240	0.687	0.014	44.1
	3	**0.822**	0.419	**0.699**	0.013	0.649	0.000	119.0
	2	0.639	0.103	**0.505**	0.000	**0.979**	0.000	191.7

Table 5.3: Assessment using KID and different precision (P) and recall (R) metrics on AFHQ (512x512). We used ≈5K real and generated samples and present the average over 10 evaluation runs for each metric. For our metrics as well as KNN, a higher score is better. For the distance-based metric, KID, a lower score is better.

Network	b	LSH ↑		LSH + KNN ↑		KNN [Kyn+19] ↑		KID [Bińkowski+18] ↓
		P	R	P	R	P	R	$\times 10^3$
ADA (AFHQ Cat)	32	0.940	0.917	0.714	0.753	0.765	0.532	0.7
	4	**0.946**	0.901	**0.752**	0.697	**0.841**	0.356	4.6
	3	**0.957**	0.829	**0.773**	0.566	**0.819**	0.140	21.8
	2	0.847	0.534	0.459	0.156	0.295	0.005	55.3
ADA (AFHQ Dog)	32	0.776	0.779	0.484	0.509	0.743	0.605	1.1
	4	0.775	0.762	**0.495**	0.495	**0.820**	0.436	3.4
	3	0.769	0.656	0.468	0.375	**0.809**	0.185	15.2
	2	0.577	0.317	0.153	0.028	0.216	0.010	66.1
ADA (AFHQ Wild)	32	0.963	0.921	0.872	0.739	0.758	0.286	0.4
	4	**0.965**	0.906	**0.880**	0.705	**0.797**	0.156	1.5
	3	**0.977**	0.851	0.869	0.567	0.648	0.034	8.4
	2	0.495	0.645	0.166	0.059	0.058	0.00	72.1

scores, respectively. To increase the number of values and improve the correlation study, we used the scores obtained from all the previous metrics when assessing the previously compressed generators from 6 to 2 bits.

The correlation results between FID and the different double-valued metric assessments of precision and recall on several compressed GANs pre-trained on FFHQ are presented in Figure 5.9. We observe that absolute Pearson correlations are high across the different GANs and double-valued metrics, indicating a linear relation between FID scores and the different precision and recall values. On the other hand, the absolute Spearman correlations are high only for recall assessments, with precision assessments measuring close to no correlation to the FID scores.

Figure 5.10 presents the correlation between KID and the compared double-valued metrics. We observe that the KID's correlations are of a similar nature as the previous FID's correlations, presenting high absolute Pearson correlations for both precision and recall assessments but higher absolute Spearman correlation for recall than precision.

The previous correlation studies suggest that popular single-valued metrics, such as FID and KID, may be biased towards sample diversity and may neglect

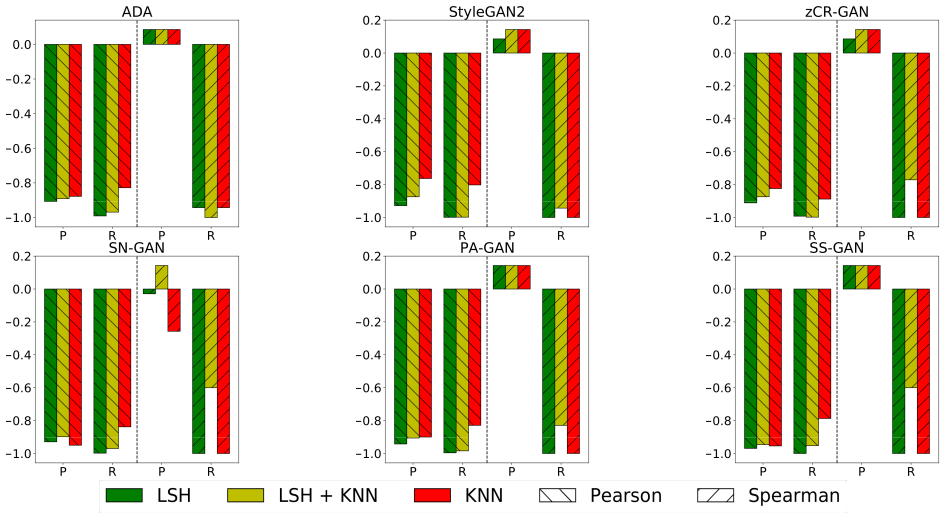
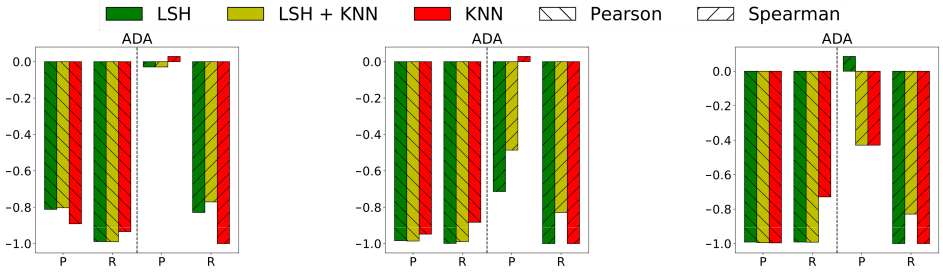


Figure 5.9: Pearson and Spearman correlations between FID and the different precision and recall methods on several compressed generators pre-trained on FFHQ 256x256.

sample quality in the evaluation of generated sets. However, the high sensitivity to sample diversity may empower such metrics to be used to detect mode collapse.



(a) AFHQ Cat. **(b)** AFHQ Dog. **(c)** AFHQ Wild.

Figure 5.10: Pearson and Spearman correlations between KID and the different precision and recall methods on compressed ADA generators pre-trained on AFHQ 512x512.

Pareto frontier

Finally, we study how compression may be used to balance pre-trained generators that may exhibit higher sample diversity but lower sample quality than other generators. To this end, we consider several configurations of style-based generators presented by Karras et al. [Kar+20b] and analyze how a compressed StyleGAN2 generator compares in terms of precision and recall measurements using IMPAR or KNN. The Pareto frontier of the different generators is presented in Figure 5.11.

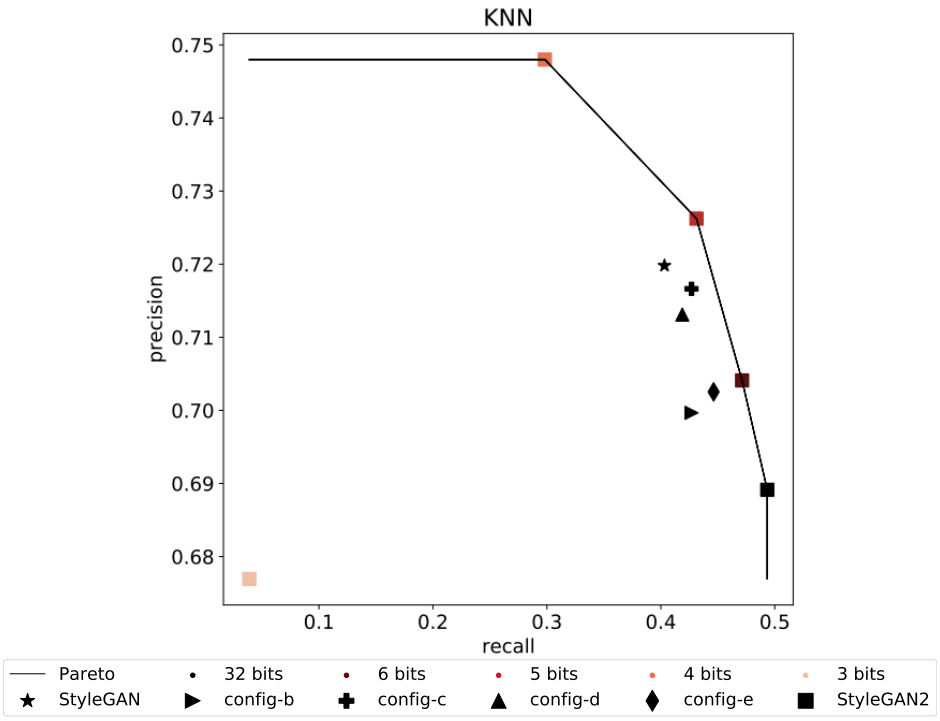


Figure 5.11: Pareto frontier of different style-based generator configurations pre-trained on FFHQ 1024x1024. We present the average over 3 evaluation runs.

The compared variants consist of the original StyleGAN generator [KLA19] with weight demodulation (config-b), lazy regularization (config-c), path length regularization (config-d), new architectural settings (config-e), and larger networks (StyleGAN2). We note that the 32-bit StyleGAN2 generator achieves the

highest recall compared to the other variants at the cost of sacrificed precision. However, we observe that compression may help to balance out the generation process, increasing precision when compressing up to 4 bits. More particularly, the 5-bit StyleGAN2 generator achieves the highest precision at similar recall values of the compared 32-bit generators.

5.3 Concluding Remarks

In this Chapter, we applied existing post-training compression techniques to the generators of several GANs. From all the tested techniques, clipping and linear quantization showed the best performance, allowing the compression of the generator weights to low bit-widths. Overall, we observed that the sample diversity of compressed generators tends to get more negatively affected by high compression levels than sample quality. Hence, post-training compression may be used to efficiently balance existing generators at a low computational cost without requiring any fine-tuning.

For a better assessment of the compression effects, we proposed two new double-valued evaluation metrics based on locality-sensitive hashing. Our new metrics diminish the negative impact that a low number of outlier samples may have on the overall assessment of a generated set. Moreover, we significantly improve the efficiency of the evaluation process compared to existing KNN-based metrics.

We note that performing online compression on the generator activations may also be achieved by the discussed compression techniques. Hence, allying both weight and activation compression presents a promising direction for future work. This may ultimately allow the deployment of compressed GANs in real-world applications by further reducing computational and memory costs at inference time.

6 Conclusion

In this thesis, we propose several diversification, compression, and evaluation methods for generative adversarial networks (GANs). Particularly, we introduce new diversification methods to induce sample diversity by leveraging discriminator ensembles to mitigate mode collapse. To assess a generated set, we proposed several novel single-valued and double-valued metrics that may be used in different application scenarios, such as the evaluation of image generation and both conditional and unconditional language generation. Moreover, we present two compression methods that may be used to reduce the memory, computation, and communication costs of existing GANs. Finally, we propose two efficient evaluation metrics to appropriately assess how the generated set of a compressed generator is affected by different compression levels and techniques.

In Chapter 2, we discuss the importance of a balanced generation process, possessing both sample quality and sample diversity. To this end, we propose to tackle the lack of sample diversity in existing GANs, commonly referred to as mode collapse, using two novel multi-adversarial frameworks: Dropout-GAN and microbatchGAN. Both frameworks leverage discriminator ensembles in different ways to ultimately stimulate sample diversity in the generator during training. In the future, making the different discriminators aware of one another may enhance the overall feedback provided to the generator.

In Chapter 3, we present two novel neural network compression methods: Monte Carlo Quantization and Monte Carlo Gradient Quantization. Such methods may be used to reduce the costs of increasing a framework size, as previously proposed. The first method reduces memory and computation costs by quantizing the weights and activations of pre-trained models, in a post-training fashion and without any additional fine-tuning. On the other hand, the latter method reduces the communication cost between different workers in a distributed training setting by quantizing and encoding synchronized gradients during training. Both approaches use Monte Carlo methods allied with importance sampling to quantize and prune floating-point values to approximated integer represen-

tations. In the future, combining both approaches to compress a model from scratch is worth exploring.

In Chapter 4, we discuss the importance of properly assessing a generated set and introduce two novel evaluation methods: Fuzzy Topology Impact and Mark-Evaluate. The first method retrieves a double-valued assignment in terms of sample quality and sample diversity by using topology representations and fuzzy logic to measure the impact between a real and generated image set. The latter method consists of a family of both single-valued and double-valued metrics by using different population estimation methods to assess a generated text set. Testing the adaptability of both methods on additional data domains is a worthy future direction to promote their general applicability.

In Chapter 5, we study the generator weight compression effects on the generated set by using existing post-training compression methods, including Monte Carlo Quantization. To properly assess such effects, we propose two efficient evaluation metrics that rely on locality-sensitive hashing to minimize the negative effects that outliers may have in the overall assessment. Overall, we observe that existing methods may be applied to compress pre-trained generators, with higher compression levels offering a trade-off between sample quality and sample diversity. Specifically, while sample diversity is highly affected in low compression bit-widths, sample quality is maintained to some degree. Compressing both the weights and activations is a promising next step to further optimize the inference of highly compressed generators.

Even though we mostly focused on GANs advancement throughout this thesis, the proposed methods should be seen as general practices that may be applied to different frameworks, applications, and data domains to ultimately enhance real-world applications. Particularly, the proposed diversification methods may be used to extend other adversarial frameworks, whereas the compression methods may be generally applied to a wide range of neural networks and the evaluation methods applied to different generative frameworks. Hence, considering their general applicability, the presented methods may be seen as different tools to advance existing machine learning and deep learning methods. A more detailed discussion of future work directions that may be worth pursuing follows.

6.1 Future Work

The principles introduced in Dropout-GAN and microbatchGAN may be combined to further increase the sample diversity capability of a generator model in GANs. However, finding a suitable balance between the dropout rate and microbatch discrimination of the separate methods may be required. More specifically, for Dropout-GAN, the dropout rate may have to be adjusted to not compromise the benefits of microbatch discrimination. On the other hand, for microbatchGAN, the properties of the α functions employed may need to be modified to assist training convergence. For example, using a fixed schedule for α that is not learned but is instead decided based on the interaction between the two frameworks during training may be beneficial.

On a similar note, Dropout-GAN and microbatchGAN may also be applied to existing GANs. From the two, Dropout-GAN is more general and, therefore, easier to apply to existing frameworks. For example, by extending D2GAN [Ngu+17] to an even number of dropout discriminators or MGAN [Hoa+18] to multiple discriminators with each MGAN generator being assigned to a different dynamic ensemble of discriminators independently. Additional efforts on microbatchGAN are likely needed to be followed to ensure compatibility with other GAN frameworks. Particularly, special care is required to successfully incorporate microbatch discrimination in existing frameworks that also modify the value function in GANs or rely on additional models, such as auto-encoders [WB17].

A more in-depth look at the benefits of using the proposed Monte Carlo sampling techniques in MCQ or MCGQ compared to simpler approaches, such as deterministic or stochastic rounding [Gup+15], are important to be studied in different scenarios. In principle, the ability to both prune and quantize neural networks by our proposed methods promotes efficiency and allows better control over sparsity levels. For example, when using 4 samples, one will at most hit 4 different weights, while the rest may be pruned. Moreover, by leveraging random offsets, the importance sampling process of MCQ and MCGQ allows the iteration over different random offsets, which is likely to result in better compressed neural networks in the end. Nevertheless, using the aforementioned simpler techniques may be beneficial in certain applications. Particularly, comparing MCQ to stochastic rounding applied to pre-trained models at inference time represents interesting future work.

The process of evaluating evaluation metrics usually relies on correlation

to human evaluation. Even though such benchmarks are widely used in the natural language processing community, there is a shortage of image generation benchmarks in computer vision. However, additional efforts are also needed to further improve the existing benchmarks for text generation. Specifically, having human evaluation data on assessing language in different aspects other than fluency, such as adequacy in conditional language generation, is important to broaden the application scenarios of existing natural generation models. Overall, increasing both the number and effectiveness of benchmarks available is a crucial step to properly assess, improve, and develop novel evaluation metrics both for image and text.

Applying the diversification, compression, and evaluation methods proposed in this thesis to novel generative models is a promising direction to follow. Particularly, generative flow networks (GFlowNets) [Ben+21] were recently proposed and show great promise to be a likely successor of GANs due to their increased sample efficiency and ability to find prominent data modes. We are looking forward to seeing how the techniques proposed throughout this thesis may help shape or improve the next-generation generative models.

Bibliography

[AB17] Martin Arjovsky and Léon Bottou. **Towards principled methods for training generative adversarial networks**. In: *International Conference on Learning Representations*. 2017 (cited in pages 4, 12).

[Aba+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. **Tensorflow: Large-scale machine learning on heterogeneous distributed systems**. *arXiv preprint arXiv:1603.04467* (2016) (cited in page 15).

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. **Wasserstein generative adversarial networks**. In: *International Conference on Machine Learning*. 2017 (cited in pages 4, 12, 38).

[Agu+19] Angeline Aguinaldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. **Compressing GANs using knowledge distillation**. *arXiv preprint arXiv:1902.00159* (2019) (cited in page 108).

[AH17] Alham Fikri Aji and Kenneth Heafield. **Sparse communication for distributed gradient descent**. In: *Conference on Empirical Methods in Natural Language Processing*. 2017 (cited in pages 43, 69).

[Ali+17] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. **QSGD: Communication-efficient SGD via gradient quantization and encoding**. In: *Advances in Neural Information Processing Systems*. 2017 (cited in pages 5, 43, 63, 68).

[Aro+17] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. **Generalization and equilibrium in generative adversarial nets (GANs)**. In: *International Conference on Machine Learning*. 2017 (cited in page 38).

[BDS19] Andrew Brock, Jeff Donahue, and Karen Simonyan. **Large scale GAN training for high fidelity natural image synthesis**. In: *International Conference on Learning Representations*. 2019 (cited in pages 86, 116).

[Ben+21] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. **Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation**. In: *Advances in Neural Information Processing Systems*. 2021 (cited in page 130).

[Ben75] Jon Louis Bentley. **Multidimensional binary search trees used for associative searching**. *Communications of the ACM* (1975) (cited in page 109).

[Ber+18] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. **signSGD: Compressed optimisation for non-convex problems**. In: *International Conference on Machine Learning*. 2018 (cited in pages 5, 43, 71, 72).

[BGK17] Ondřej Bojar, Yvette Graham, and Amir Kamran. **Results of the WMT17 metrics shared task**. In: *Conference on Machine Translation*. 2017 (cited in page 103).

[Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. **Demystifying MMD GANs**. In: *International Conference on Learning Representations*. 2018 (cited in pages 5, 121, 123).

[BL05] Satanjeev Banerjee and Alon Lavie. **METEOR: An automatic metric for MT evaluation with improved correlation with human judgments**. In: *ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. 2005 (cited in page 79).

[BNS19] Ron Banner, Yury Nahshan, and Daniel Soudry. **Post training 4-bit quantization of convolutional networks for rapid-deployment**. In: *Advances in Neural Information Processing Systems*. 2019 (cited in pages 42, 74, 115).

[Bow+15] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. **A large annotated corpus for learning natural language inference**. In: *Conference on Empirical Methods in Natural Language Processing*. 2015 (cited in page 99).

[BS13] Pierre Baldi and Peter J Sadowski. **Understanding dropout**. In: *Advances in Neural Information Processing Systems*. 2013 (cited in page 14).

[BSM17] David Berthelot, Tom Schumm, and Luke Metz. **BEGAN: Boundary equilibrium generative adversarial networks**. *arXiv preprint arXiv: 1703.10717* (2017) (cited in pages 12, 13, 38).

[CBD15a] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. **BinaryConnect: Training deep neural networks with binary weights during propagations**. In: *Advances in Neural Information Processing Systems*. 2015 (cited in pages 4, 42, 51, 54, 55).

[CBD15b] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. **Training deep neural networks with low precision multiplications**. In: *Workshop of the International Conference on Learning Representations*. 2015 (cited in pages 5, 43).

[CF18] Tatjana Chavdarova and Francois Fleuret. **SGAN: An alternative training of generative adversarial networks**. In: *Proceedings of the IEEE international conference on Computer Vision and Pattern Recognition*. 2018 (cited in pages 4, 13).

[Che+17] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. **Mode regularized generative adversarial networks**. In: *International Conference on Learning Representations*. 2017 (cited in pages 4, 12, 13).

[Che+18] Chia-Yu Chen, Jungwook Choi, Daniel Brand, Ankur Agrawal, Wei Zhang, and Kailash Gopalakrishnan. **AdaComp: Adaptive residual gradient compression for data-parallel distributed training**. In: *AAAI Conference on Artificial Intelligence*. 2018 (cited in pages 43, 66, 73, 74).

[Che+19] Ting Chen, Xiaohua Zhai, Marvin Ritter, Mario Lucic, and Neil Houlsby. **Self-supervised gans via auxiliary rotation loss**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019 (cited in page 116).

[Che+20] Hanting Chen, Yunhe Wang, Han Shu, Changyuan Wen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. **Distilling portable generative adversarial networks for image translation**. In: *AAAI Conference on Artificial Intelligence*. 2020 (cited in pages 107, 108).

[Cho+20] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. **StarGAN v2: Diverse image synthesis for multiple domains**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2020 (cited in page 116).

[Cíf+18] Ondřej Cífka, Aliaksei Severyn, Enrique Alfonseca, and Katja Filippova. **Eval all, trust a few, do wrong to none: Comparing sentence generation models**. *arXiv preprint arXiv:1804.07972* (2018) (cited in pages 101, 102).

[CNL11] Adam Coates, Andrew Ng, and Honglak Lee. **An analysis of single-layer networks in unsupervised feature learning**. In: *International Conference on Artificial Intelligence and Statistics*. 2011 (cited in page 25).

[Con+17] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. **Supervised learning of universal sentence representations from natural language inference data**. In: *Conference on Empirical Methods in Natural Language Processing*. 2017 (cited in page 79).

[Dev+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. **BERT: Pre-training of deep bidirectional transformers for language understanding**. In: *Conference of the North American Chapter of the Association for Computational Linguistics*. 2019 (cited in pages 79, 92).

[DGM17] Ishan Durugkar, Ian Gemp, and Sridhar Mahadevan. **Generative multi-adversarial networks**. In: *International Conference on Learning Representations*. 2017 (cited in pages 4, 13, 17, 18, 24, 38).

[DMP19] Chris Donahue, Julian McAuley, and Miller Puckette. **Adversarial audio synthesis**. In: *International Conference on Learning Representations*. 2019 (cited in page 3).

[Dry+16] Nikoli Dryden, Tim Moon, Sam Jacobs, and Brian Van Essen. **Communication quantization for data-parallel training of deep neural networks**. In: *Workshop on Machine Learning in HPC Environments*. 2016 (cited in page 43).

[Dum+17] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. **Adversarially learned inference**. In: *International Conference on Learning Representations*. 2017 (cited in page 38).

[EMH19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. **Neural architecture search: A survey**. *Journal of Machine Learning Research* (2019) (cited in page 108).

[ES16] Harrison Edwards and Amos Storkey. **Censoring representations with an adversary**. In: *International Conference on Learning Representations*. 2016 (cited in page 3).

[Gan+17] Zhe Gan, Liqun Chen, Weiyao Wang, Yuchen Pu, Yizhe Zhang, Hao Liu, Chunyuan Li, and Lawrence Carin. **Triangle generative adversarial networks**. In: *Advances in Neural Information Processing Systems*. 2017 (cited in pages 4, 13).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. **Deep learning**. MIT press, 2016 (cited in page 1).

[GE18] Aditya Grover and Stefano Ermon. **Boosted generative models**. In: *AAAI Conference on Artificial Intelligence*. 2018 (cited in pages 4, 13).

[GG16] Yarin Gal and Zoubin Ghahramani. **A theoretically grounded application of dropout in recurrent neural networks**. In: *Advances in neural information processing systems*. 2016 (cited in page 14).

[Gho+18] Arnab Ghosh, Viveka Kulharia, Vinay P Namboodiri, Philip HS Torr, and Puneet K Dokania. **Multi-agent diverse generative adversarial networks**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2018 (cited in pages 4, 13).

[GKV17] Swaminathan Gurumurthy, Ravi Kiran Sarvadevabhatla, and R Venkatesh Babu. **DeLiGAN: Generative adversarial networks for diverse and limited data**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017 (cited in page 77).

[GLS99] William Gropp, Ewing Lusk, and Anthony Skjellum. **Using MPI: portable parallel programming with the message-passing interface**. MIT press, 1999 (cited in page 62).

[Goo+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. **Generative adversarial nets**. In: *Advances in Neural Information Processing Systems*. 2014 (cited in pages 3, 11, 12, 14, 23–25, 27, 29, 37, 38, 44).

[Gre+12] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. **A kernel two-sample test**. *Journal of Machine Learning Research* (2012) (cited in page 77).

[Gup+15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. **Deep learning with limited numerical precision**. In: *International Conference on Machine Learning*. 2015 (cited in pages 5, 43, 44, 129).

[Har+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. **Array programming with NumPy**. *Nature* (2020) (cited in pages 48, 58).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep residual learning for image recognition**. In: *IEEE conference on computer vision and pattern recognition*. 2016 (cited in pages 50, 70).

[HE16] Jonathan Ho and Stefano Ermon. **Generative adversarial imitation learning**. In: *Advances in Neural Information Processing Systems*. 2016 (cited in page 3).

[Heu+17] 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 (cited in pages 5, 18, 34, 77–79, 87, 91, 99, 121, 122).

[HMD16] Song Han, Huizi Mao, and William J. Dally. **Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding**. In: *International Conference on Learning Representations*. 2016 (cited in pages 4, 42, 50).

[Hoa+18] Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung. **MGAN: Training generative adversarial nets with multiple generators**. In: *International Conference on Learning Representations*. 2018 (cited in pages 4, 13, 38, 129).

[Hub+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. **Binarized neural networks**. In: *Advances in neural information processing systems*. 2016 (cited in pages 4, 42, 51, 54, 55).

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. **Distilling the knowledge in a neural network**. *arXiv preprint arXiv:1503.02531* (2015) (cited in page 107).

[Kar+19] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. **Error feedback fixes SignSGD and other gradient compression schemes**. In: *International Conference on Machine Learning*. 2019 (cited in pages 5, 43, 70–72).

[Kar+20a] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. **Training generative adversarial networks with limited data**. In: *Advances in Neural Information Processing Systems*. 2020 (cited in pages 116, 121).

[Kar+20b] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. **Analyzing and improving the image quality of StyleGAN**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2020 (cited in pages 86, 116, 125).

[KB15] Diederik P. Kingma and Jimmy Ba. **Adam: A method for stochastic optimization**. In: *International Conference on Learning Representations*. 2015 (cited in pages 50, 59).

[KD18] Durk P Kingma and Prafulla Dhariwal. **Glow: Generative flow with invertible 1x1 convolutions**. In: *Advances in neural information processing systems*. 2018 (cited in page 86).

[Kel+19a] Alexander Keller, Gonçalo Mordido, Noah Gamboa, and Matthijs Van Keirsbilck. **Representing a neural network utilizing paths within the network to improve a performance of the neural network** (2019). US Patent App. 16/352,596 (cited in page 3).

[Kel+19b] Alexander Keller, Gonçalo Mordido, Noah Gamboa, Matthijs Van Keirsbilck, Xiaodong Yang, Pavlo Molchanov, and Jan Kautz. **Structural sparsity: Speeding up training and inference of neural networks by linear algorithms**. *GPU Technology Conference* (2019) (cited in page 3).

[Kim+17] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. **Learning to discover cross-domain relations with generative adversarial networks**. In: *International Conference on Machine Learning*. 2017 (cited in pages 4, 12).

[KL51] Solomon Kullback and Richard A Leibler. **On information and sufficiency**. *The annals of mathematical statistics* (1951) (cited in page 78).

[KLA19] Tero Karras, Samuli Laine, and Timo Aila. **A style-based generator architecture for generative adversarial networks**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019 (cited in pages 86, 116, 125).

[KMV21] Alexander Keller, Gonçalo Mordido, and Matthijs Van Keirsbilck. **Incorporating a ternary matrix into a neural network** (2021). Filed patent. (cited in page 3).

[KO18] Valentin Khrulkov and Ivan Oseledets. **Geometry score: A method for comparing generative adversarial networks**. In: *International Conference on Machine Learning*. 2018 (cited in page 77).

[Kod+17] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. **On convergence and stability of GANs**. *arXiv preprint arXiv:1705.07215* (2017) (cited in pages 23, 25, 37).

[Kor+19] Animesh Koratana, Daniel Kang, Peter Bailis, and Matei Zaharia. **LIT: Learned intermediate representation training for Mmdel compression**. In: *International Conference on Machine Learning*. 2019 (cited in page 108).

[Kre11] Wolfgang Kreitmeier. **Optimal vector quantization in terms of Wasserstein distance**. *Multivariate Analysis* (2011) (cited in page 60).

[Kre89] Charles Krebs. **Ecological methodology**. Harper & Row New York, 1989 (cited in pages 92, 96).

[Kri09] Alex Krizhevsky. **Learning multiple layers of features from tiny images** (2009) (cited in pages 18, 34, 88).

[Kyn+19] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. **Improved precision and recall metric for assessing generative models**. In: *Advances in Neural Information Processing Systems*. 2019 (cited in pages 5, 6, 77, 79, 83, 86, 87, 91, 92, 94, 99, 109, 110, 112–114, 116, 121–123).

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. **MNIST handwritten digit database**. *ATT Labs* (2010) (cited in pages 18, 34).

[LD15] Shuying Liu and Weihong Deng. **Very deep convolutional neural network based image classification using small training sample size**. In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. 2015 (cited in pages 50, 109).

[LDS90] Yann LeCun, John S Denker, and Sara A Solla. **Optimal brain damage**. In: *Advances in Neural Information Processing Systems*. 1990 (cited in pages 4, 42).

[LEc+18] Pierre L'Ecuyer, David Munger, Christian Lécot, and Bruno Tuffin. **Sorting methods and convergence rates for Array-RQMC: Some empirical comparisons**. *Mathematics and Computers in Simulation* (2018) (cited in page 49).

[Li+17] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabas Poczos. **MMD GAN: Towards deeper understanding of moment matching network**. In: *Advances in Neural Information Processing Systems*. 2017 (cited in page 12).

[Li+20] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. **GAN compression: Efficient architectures for interactive conditional GANs**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2020 (cited in pages 107, 108).

[Lin+16] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. **Neural networks with few multiplications**. In: *International Conference on Learning Representations*. 2016 (cited in pages 4, 42).

[Lin+18a] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. **Deep gradient compression: Reducing the communication bandwidth for distributed training**. In: *International Conference on Learning Representations*. 2018 (cited in pages 43, 61, 63, 65, 69, 72, 73).

[Lin+18b] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. **PacGAN: The power of two samples in generative adversarial networks**. In: *Advances in Neural Information Processing Systems*. 2018 (cited in page 13).

[Lin04] Chin-Yew Lin. **ROUGE: A package for automatic evaluation of summaries**. In: *Text Summarization Branches Out*. 2004 (cited in page 79).

[Liu+15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. **Deep learning face attributes in the wild**. In: *International Conference on Computer Vision*. 2015 (cited in pages 18, 34).

[Liu+19] Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. **Linguistic knowledge and transferability of contextual representations**. In: *Conference of the North American Chapter of the Association for Computational Linguistics*. 2019 (cited in page 102).

[Liu+20] Jinglan Liu, Jiaxin Zhang, Yukun Ding, Xiaowei Xu, Meng Jiang, and Yiyu Shi. **Binarizing weights wisely for edge intelligence: Guide for partial binarization of deconvolution-based generators**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020) (cited in page 108).

[LLT08] Pierre L'Ecuyer, Christian Lécot, and Bruno Tuffin. **A randomized quasi-Monte Carlo simulation method for Markov chains**. *Operations Research* (2008) (cited in page 49).

[Lou+19] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. **Relaxed quantization for discretized neural networks**. In: *International Conference on Learning Representations*. 2019 (cited in pages 5, 43, 55, 56).

[Luc+18] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. **Are GANs created equal? A large-scale study**. In: *Advances in Neural Information Processing Systems*. 2018 (cited in pages 19, 23, 118).

[LZL16] Fengfu Li, Bo Zhang, and Bin Liu. **Ternary weight networks**. In: *International Workshop on Efficient Methods for Deep Neural Networks*. 2016 (cited in pages 4, 42, 50, 54–56).

[Mac67] James MacQueen. **Some methods for classification and analysis of multivariate observations**. In: *Berkeley Symposium on Mathematical Statistics and Probability*. 1967 (cited in page 78).

[Mao+17] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. **Least squares generative adversarial networks**. In: *IEEE International Conference on Computer Vision*. 2017 (cited in pages 23, 25, 37).

[MBC19] Nitika Mathur, Timothy Baldwin, and Trevor Cohn. **Putting evaluation in context: Contextual embeddings improve machine translation evaluation**. In: *Annual Meeting of the Association for Computational Linguistics*. 2019 (cited in page 79).

[Mel+17] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. **Ternary neural networks with fine-grained quantization**. *arXiv preprint arXiv:1705.01462* (2017) (cited in pages 42, 56).

[Met+17] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. **Unrolled generative adversarial networks**. In: *International Conference on Learning Representations*. 2017 (cited in pages 12, 21).

[MHM18] Leland McInnes, John Healy, and James Melville. **UMAP: Uniform manifold approximation and projection for dimension reduction**. *arXiv preprint arXiv:1802.03426* (2018) (cited in pages 80, 81, 83).

[Miy+18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. **Spectral normalization for generative adversarial networks**. In: *International Conference on Learning Representations*. 2018 (cited in page 116).

[MM20] Gonçalo Mordido and Christoph Meinel. **Mark-Evaluate: Assessing language generation using population estimation methods**. In: *International Conference on Computational Linguistics*. 2020 (cited in pages 2, 77).

[MNG17] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. **The numerics of GANs**. In: *Advances in Neural Information Processing Systems*. 2017 (cited in pages 4, 12).

[MNM21] Gonçalo Mordido, Julian Niedermeier, and Christoph Meinel. **Assessing image and text generation with topological analysis and fuzzy logic**. In: *IEEE Winter Conference on Applications of Computer Vision*. 2021 (cited in pages 2, 77).

[Moc+18] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. **Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science**. *Nature communications* (2018) (cited in pages 4, 42).

[Mol+17] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. **Pruning convolutional neural networks for resource efficient inference**. In: *International Conference on Learning Representations*. 2017 (cited in page 74).

[MSG17] Youssef Mroueh, Tom Sercu, and Vaibhava Goel. **McGan: Mean and covariance feature matching GAN**. In: *International Conference on Machine Learning*. 2017 (cited in page 12).

[MSM93] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. **Building a large annotated corpus of English: The Penn Treebank**. *Computational Linguistics* (1993) (cited in page 65).

[MVK19] Gonçalo Mordido, Matthijs Van keirsbilck, and Alexander Keller. **Instant quantization of neural networks using Monte Carlo methods**. In: *Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (to appear)*. 2019 (cited in pages 2, 41, 115).

[MVK20] Gonçalo Mordido, Matthijs Van Keirsbilck, and Alexander Keller. **Monte Carlo gradient quantization**. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2020 (cited in pages 2, 41).

[MVK21a] Gonçalo Mordido, Matthijs Van Keirsbilck, and Alexander Keller. **1x1 convolutions by random ternary matrices**. *GPU Technology Conference* (2021) (cited in page 3).

[MVK21b] Gonçalo Mordido, Matthijs Van Keirsbilck, and Alexander Keller. **Compressing 1D time-channel separable convolutions using sparse random ternary matrices**. In: *INTERSPEECH (to appear)*. 2021 (cited in page 3).

[MYM18] Gonçalo Mordido, Haojin Yang, and Christoph Meinel. **Dropout-GAN: Learning from a dynamic ensemble of discriminators**. *KDD Deep Learning Day* (2018) (cited in pages 1, 4, 11).

[MYM20] Gonçalo Mordido, Haojin Yang, and Christoph Meinel. **microbatchGAN: Stimulating diversity with multi-adversarial discrimination**. In: *IEEE Winter Conference on Applications of Computer Vision*. 2020 (cited in pages 2, 4, 11).

[MYM21] Gonçalo Mordido, Haojin Yang, and Christoph Meinel. **Evaluating post-training compression in GANs using locality-sensitive hashing**. *arXiv preprint arXiv:2103.11912* (2021) (cited in pages 2, 107).

[NBC17] Behnam Neyshabur, Srinadh Bhojanapalli, and Ayan Chakrabarti. **Stabilizing GAN training with multiple random projections**. *arXiv preprint arXiv:1705.07831* (2017) (cited in page 17).

[NCT16] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. **f-gan: Training generative neural samplers using variational divergence minimization**. In: *Advances in Neural Information Processing Systems*. 2016 (cited in page 12).

[Nee+15] Arvind Neelakantan, Luke Vilnis, Quoc Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. **Adding gradient noise improves learning for very deep networks**. *arXiv preprint arXiv:1511.06807* (2015) (cited in page 59).

[Net+11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. **Reading digits in natural images with unsupervised feature learning**. In: *Advances in Neural Information Processing Systems*. 2011 (cited in page 50).

[Ngu+17] Tu Nguyen, Trung Le, Hung Vu, and Dinh Phung. **Dual discriminator generative adversarial nets**. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. 2017 (cited in pages 4, 13, 29, 38, 129).

[NGV12] Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. **Annotated Gigaword**. In: *Joint Workshop on Automatic Knowledge Base Construction and Web-Scale Knowledge Extraction*. 2012 (cited in page 101).

[NH10] Vinod Nair and Geoffrey E Hinton. **Rectified linear units improve restricted boltzmann machines**. In: *International Conference on Machine Learning*. 2010 (cited in page 44).

[NMM20] Julian Niedermeier, Gonçalo Mordido, and Christoph Meinel. **Improving the evaluation of generative models with fuzzy logic**. *Workshop on Evaluating Evaluation of AI Systems* (2020) (cited in page 77).

[Oti+78] David L Otis, Kenneth P Burnham, Gary C White, and David R Anderson. **Statistical inference from capture data on closed animal populations**. *Wildlife monographs* (1978) (cited in pages 92, 96).

[Pap+02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. **BLEU: A method for automatic evaluation of machine translation**. In: *Annual Meeting of the Association for Computational Linguistics*. 2002 (cited in page 79).

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. **PyTorch: An imperative style, high-performance deep learning library**. In: *Advances in Neural Information Processing Systems*. 2019 (cited in page 53).

[Ped+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. **Scikit-learn: Machine learning in Python**. *Journal of machine learning research* (2011) (cited in page 65).

[Ras+16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. **XNOR-net: ImageNet classification using binary convolutional neural networks**. In: *European Conference on Computer Vision*. 2016 (cited in pages 4, 42, 54–56).

[Rea+18] Brandon Reagan, Udit Gupta, Bob Adolf, Michael Mitzenmacher, Alexander Rush, Gu-Yeon Wei, and David Brooks. **Weightless: Lossy weight encoding for deep neural network compression**. In: *International Conference on Machine Learning*. 2018 (cited in page 42).

[Rei+20] Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. **COMET: A Neural Framework for MT Evaluation**. In: *Conference on Empirical Methods in Natural Language Processing*. 2020 (cited in page 79).

[RG19] Nils Reimers and Iryna Gurevych. **Sentence-BERT: Sentence embeddings using siamese BERT-networks**. In: *Conference on Empirical Methods in Natural Language Processing and Conference on Natural Language Processing*. 2019 (cited in pages 79, 92, 98, 101).

[Ric75] William Edwin Ricker. **Computation and interpretation of biological statistics of fish populations**. *Bulletin of the Fisheries Research Board of Canada* (1975) (cited in pages 92, 94).

[RM51] Herbert Robbins and Sutton Monro. **A stochastic approximation method**. *The Annals of Mathematical Statistics* (1951) (cited in pages 44, 57).

[RMC16] Alec Radford, Luke Metz, and Soumith Chintala. **Unsupervised representation learning with deep convolutional generative adversarial networks**. In: *International Conference on Learning Representations*. 2016 (cited in pages 18, 38).

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. **ImageNet large scale visual recognition challenge**. *International Journal of Computer Vision* (2015) (cited in pages 25, 116).

[Saj+18] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. **Assessing generative models via precision and recall**. In: *Advances in Neural Information Processing Systems*. 2018 (cited in pages 5, 77, 78, 83, 87, 91, 99).

[Sal+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. **Improved techniques for training GANs**. In: *Advances in Neural Information Processing Systems*. 2016 (cited in pages 5, 13, 19, 24, 38, 77, 87).

[Sau+20] Jonathan Sauder, Ting Hu, Xiaoyin Che, Gonçalo Mordido, Haojin Yang, and Christoph Meinel. **Best student forcing: A simple training mechanism in adversarial language generation**. In: *Language Resources and Evaluation Conference*. 2020 (cited in pages 3, 11, 40).

[SC08] Malcolm Slaney and Michael Casey. **Locality-sensitive hashing for finding nearest neighbors [lecture notes]**. *IEEE Signal processing magazine* (2008) (cited in page 111).

[Sch38] Zoe Emily Schnabel. **The estimation of the total fish population of a lake**. *The American Mathematical Monthly* (1938) (cited in pages 92, 95).

[SCJ18] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. **Sparsified SGD with Memory**. In: *Advances in Neural Information Processing Systems*. 2018 (cited in pages 5, 43, 65, 68).

[SDP20] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. **BLEURT: Learning robust metrics for text generation**. In: *Annual Meeting of the Association for Computational Linguistics*. 2020 (cited in page 79).

[Sei+14] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. **1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs**. In: *Conference of the International Speech Communication Association*. 2014 (cited in pages 5, 43).

[SGM19] Emma Strubell, Ananya Ganesh, and Andrew McCallum. **Energy and policy considerations for deep learning in NLP**. In: *Annual Meeting of the Association for Computational Linguistics*. 2019 (cited in pages 4, 42).

[Shu+19] Han Shu, Yunhe Wang, Xu Jia, Kai Han, Hanting Chen, Chunjing Xu, Qi Tian, and Chang Xu. **Co-evolutionary compression for unpaired image translation**. In: *IEEE International Conference on Computer Vision*. 2019 (cited in page 107).

[SLF18] Oran Shayer, Dan Levi, and Ethan Fetaya. **Learning discrete weights using the local reparameterization Trick**. In: *International Conference on Learning Representations*. 2018 (cited in pages 42, 55, 56).

[Son+08] Soeren Sonnenburg, Vojtech Franc, Elad Yom-Tov, and Michele Sebag. **Pascal large scale learning challenge**. In: *International Conference on Machine Learning Workshop*. 2008 (cited in page 65).

[Spr15] Jost Tobias Springenberg. **Unsupervised and semi-supervised learning with categorical generative adversarial networks**. *arXiv preprint arXiv:1511.06390* (2015) (cited in page 13).

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. **Dropout: a simple way to prevent neural networks from overfitting**. *Journal of Machine Learning Research* (2014) (cited in pages 14, 18, 44).

[SSA18] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. **How good is my GAN?** In: *European Conference on Computer Vision*. 2018 (cited in page 77).

[SSG18] Stanislau Semeniuta, Aliaksei Severyn, and Sylvain Gelly. **On accurate evaluation of GANs for language generation**. *arXiv preprint arXiv:1806.04936* (2018) (cited in pages 79, 99, 101, 102).

[Str15] Nikko Strom. **Scalable distributed DNN training using commodity GPU cloud computing**. In: *Annual Conference of the International Speech Communication Association*. 2015 (cited in page 43).

[Sut+17] Dougal J Sutherland, Hsiao-Yu Tung, Heiko Strathmann, Soumyajit De, Aaditya Ramdas, Alex Smola, and Arthur Gretton. **Generative models and model criticism via optimized maximum mean discrepancy**. In: *International Conference on Learning Representations*. 2017 (cited in page 12).

[Sze+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. **Rethinking the Inception architecture for computer vision**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016 (cited in page 78).

[Sze+20] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. **Efficient processing of deep neural networks**. *Synthesis Lectures on Computer Architecture* (2020) (cited in page 41).

[TOB16] Lucas Theis, Aäron van den Oord, and Matthias Bethge. **A note on the evaluation of generative models**. In: *International Conference on Learning Representations*. 2016 (cited in page 77).

[Ueh+16] Masatoshi Uehara, Issei Sato, Masahiro Suzuki, Kotaro Nakayama, and Yutaka Matsuo. **Generative adversarial nets from a density ratio estimation perspective**. *arXiv preprint arXiv:1610.02920* (2016) (cited in page 12).

[Unt+18] Thomas Unterthiner, Bernhard Nessler, Calvin Seward, Günter Klambauer, Martin Heusel, Hubert Ramsauer, and Sepp Hochreiter. **Coulomb GANs: Provably optimal Nash equilibria via potential fields**. In: *International Conference on Learning Representations*. 2018 (cited in page 12).

[Val74] Sergei Vallender. **Calculation of the Wasserstein distance between probability distributions on the line**. *Theory of Probability & Its Applications* (1974) (cited in page 60).

[VNM17] Ganesh Venkatesh, Eriko Nurvitadhi, and Debbie Marr. **Accelerating deep convolutional networks using low-precision and sparsity**. In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2017 (cited in pages 4, 42, 55).

[Wan+13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. **Regularization of neural networks using DropConnect**. In: *International Conference on Machine Learning*. 2013 (cited in page 44).

[Wan+17] Ruohan Wang, Antoine Cully, Hyung Jin Chang, and Yiannis Demiris. **MAGAN: Margin adaptation for generative adversarial networks**. *arXiv preprint arXiv:1704.03817* (2017) (cited in pages 13, 38).

[Wan+19a] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. **HAQ: Hardware-aware automated quantization with mixed precision**. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019 (cited in page 42).

[Wan+19b] Peiqi Wang, Dongsheng Wang, Yu Ji, Xinfeng Xie, Haoxuan Song, XuXin Liu, Yongqiang Lyu, and Yuan Xie. **QGAN: Quantized generative adversarial networks**. *arXiv preprint arXiv:1901.08263* (2019) (cited in pages 107, 108, 115).

[Wan+20] Haotao Wang, Shupeng Gui, Haichuan Yang, Ji Liu, and Zhangyang Wang. **GAN slimming: All-in-one GAN compression by a unified optimization framework**. In: *European Conference on Computer Vision*. 2020 (cited in page 108).

[War+14] David Warde-Farley, Ian J. Goodfellow, Aaron C. Courville, and Yoshua Bengio. **An empirical analysis of dropout in piece-wise linear networks**. In: *International Conference on Learning Representations*. 2014 (cited in page 14).

[WB17] David Warde-Farley and Yoshua Bengio. **Improving generative adversarial networks with denoising feature matching**. In: *International Conference on Learning Representations*. 2017 (cited in pages 13, 38, 129).

[Wen+17] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. **TernGrad: Ternary gradients to reduce communication in distributed deep learning**. In: *Advances in Neural Information Processing Systems*. 2017 (cited in pages 5, 43).

[WNB18] Adina Williams, Nikita Nangia, and Samuel Bowman. **A broad-coverage challenge corpus for sentence understanding through inference**. In: *Conference of the North American Chapter of the Association for Computational Linguistics*. 2018 (cited in pages 98, 99).

[Wu+18] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. **Training and inference with integers in deep neural networks**. In: *International Conference on Learning Representations*. 2018 (cited in pages 5, 43, 50).

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. **Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms**. *arXiv preprint arXiv:1708.07747* (2017) (cited in page 88).

[Yan+18] Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. **Improving neural machine translation with conditional sequence generative adversarial nets**. In: *Conference of the North American Chapter of the Association for Computational Linguistics*. 2018 (cited in page 3).

[YP20] Chong Yu and Jeff Pool. **Self-supervised generative adversarial compression**. In: *Advances in Neural Information Processing Systems*. 2020 (cited in pages 107, 108).

[Yu+15] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. **LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop**. *arXiv preprint arXiv:1506.03365* (2015) (cited in page 86).

[Yu+17] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. **SeqGAN: Sequence generative adversarial nets with policy gradient**. In: *AAAI Conference on Artificial Intelligence*. 2017 (cited in page 3).

[Zha+18a] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. **LQ-Nets: Learned quantization for highly accurate and compact deep neural networks**. In: *European Conference on Computer Vision*. 2018 (cited in page 42).

[Zha+18b] Junbo Zhao, Yoon Kim, Kelly Zhang, Alexander Rush, and Yann LeCun. **Adversarially regularized autoencoders**. In: *International Conference on Machine Learning*. 2018 (cited in page 102).

[Zha+19a] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. **Improving neural network quantization without retraining using outlier channel splitting**. In: *International Conference on Machine Learning*. 2019 (cited in pages 42, 74, 115, 117).

[Zha+19b] Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer, and Steffen Eger. **MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance**. In: *Conference on Empirical Methods in Natural Language Processing and Joint Conference on Natural Language Processing*. 2019 (cited in pages 79, 102, 104).

[Zha+20a] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. **BERTScore: Evaluating text generation with BERT**. In: *International Conference on Learning Representations*. 2020 (cited in pages 79, 102).

[Zha+20b] Zhengli Zhao, Sameer Singh, Honglak Lee, Zizhao Zhang, Augustus Odena, and Han Zhang. **Improved consistency regularization for GANs**. *arXiv preprint arXiv:2002.04724* (2020) (cited in page 116).

[Zho+16] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. **DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients** (2016) (cited in pages 5, 42, 43, 50, 52, 55, 56).

[Zho+17] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. **Incremental network quantization: Towards lossless CNNs with low-precision weights**. In: *International Conference on Learning Representations*. 2017 (cited in pages 42, 56).

[Zhu+17] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. **Trained ternary quantization**. In: *International Conference on Learning Representations*. 2017 (cited in pages 4, 42, 55, 56).

[ZK19] Dan Zhang and Anna Khoreva. **Progressive augmentation of GANs**. In: *Advances in Neural Information Processing Systems*. 2019 (cited in page 115).

[ZML17] Junbo Jake Zhao, Michaël Mathieu, and Yann LeCun. **Energy-based generative adversarial network**. In: *International Conference on Learning Representations*. 2017 (cited in page 12).