ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Machine Learning for Computer Vision

MIXING PRUNING AND DISTILLATION FOR LIGHTER DIFFUSION MODELS

CANDIDATE
Domenico Dell’Olio

SUPERVISOR
Prof. Samuele Salti

CO-SUPERVISORS
Prof. Akihiro Sugimoto
Ph.D. Minh-Duc Vo

Academic year 2023-2024
Session 1st

To the indomitable human spirit.

Abstract

Diffusion Models (DMs) represent the state-of-the-art in image generation tasks in terms of training stability and sample quality, but their sampling procedure is highly resource-intensive. This thesis addresses the efficiency problem of DMs by proposing a novel method combining *Progressive Distillation* with structured pruning, to reduce the computational and memory overhead without severely compromising image quality.

The *Progressive Distillation* method, introduced in *Progressive Distillation for Fast Sampling of Diffusion Models* by Salimans and Ho, reduces the number of required sampling steps by iteratively training a student DM to match the teacher model's output in half the steps. However, this method requires the student to retain the same network architecture as the teacher, limiting further compression. For this reason, we introduce a structured pruning technique during the distillation process, incorporating concepts such as pruning ratio differentiation based on the layer location and normalization-layer-lead pruning. We also introduce *Flexible Group Normalization* (FGN), a variation to the Group Normalization layer, to handle uneven channel groups post-pruning.

We validate our method with experiments on the CIFAR-10 dataset, conducting pruning sensitivity and weight magnitude variation analyses, and comparing different pruning scoring criteria to refine our approach. Though sacrificing some sample quality and not particularly optimized, the pruned models achieve significant reductions in computational requirements, with the best quality/compression trade-offs observed in 8-step and 4-step models. Our method provides a possible solution for DM efficiency and gives cues for further research exploring this family of combined complexity-reduction techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Generative AI has drawn a lot of the research community and public attention recently due to the development and deployment of new models capable of creating content of remarkably high quality. In the context of image generation, that is the generation of an image starting from a given dataset, a seed, and eventually a description (conditioned generation), or related image manipulation tasks (image inpainting, style transfer, super-resolution) an example is provided by the famous products by Stability AI [28], which exploit a state-of-the-art architecture to generate or manipulate images, usually starting from a text-prompt or another image. Said architectures are based on Diffusion Models (DM), which briefly after their introduction on this task largely outperformed the Generative Adversarial Networks (GAN) [7], which were traditionally used for generative tasks. In GAN-based methods, the network is trained to generate samples able to fool a discriminator network, which suffers from hard convergence problems as the so-called *mode collapse* [36]. Differently, the way DMs model the generative process can be seen as a series of network (generally U-Net [31] variants) evaluations that progressively denoise the starting seed to generate the image [16]. From a mathematical point of view, they learn a *Markovian process*, also named ”reverse process” as it is opposed to the ”forward” diffusion process adding (gaussian) noise to the image. This process is more stable to learn when compared with GANs and returns samples of higher quality. However, the generation process is highly inefficient as a single sample may require hundreds to thousands of network passes [5].

The DM sampling speed acceleration is a highly active branch of research, as obtaining faster and lighter models to execute would make them accessible to users with consumer-level hardware, would enable large real-time applications (*e.g.* video/3D/Audio generation extensions) and cut down energetic and, consequently, environmental costs for their usage. In this context, this work focuses on the refinement of a sample acceleration method for Diffusion Models, known as *Progressive Distillation*. It was developed by Salimans and Ho in [32] and it proposes to ”distill” a diffusion model in a student one requiring half of its sampling steps. This is done forcing its one-step output to match the results of two steps of the teacher. Said distillation is ”progressive”, *i.e.* once the student is trained it becomes the new teacher, and the process continues until a 1-step model is reached. This method yields tremendous speed-ups in the sampling procedure, but the authors limit its application to a setting where every distilled model employs the same architecture. This choice allows student models to be initialized with the weights of their teacher, which intuitively accelerates training convergence. Thus, this thesis proposes a method that can bypass this restriction and bring further compression to the model by also reducing its memory footprint. The main concept is to mix a process of*progressive structured pruning* to the *Progressive Distillation*, with the aim of producing lighter and faster DMs that can be used with standard hardware and libraries (*i.e.* not employing sparse computation). The pruning procedure [2] removes groups of weights that are found to be less ”important” or ”informative” by an aggregated *scoring criterion*. In the case at hand, it is applied in four ”equally-spaced in time” steps for each distillation iteration, which avoids further fine-tuning. It is based on concepts taken from the literature on pruning as keeping the signal and channel consistency when pruning filters [22], giving more importance and propagating pruning masks starting from Normalization Layers [24][18] and differentiate the pruning aggressivity depending on the position of the layer within the U-Net [19]. The method also introduces a variant to the Group Normalization Layer [40](GN) which is employed in the architecture. This variant is required as the vanilla GN performs normalization on channels, after splitting them into equal-sized groups. When one of the channels is removed, it is replaced by the immediately successive remaining channel, forcing the network to drastically change the learned weights and pre-computed statistics for that group. In addition, the vanilla GN requires that either groups are fully removed or that the group size is changed depending on the number of remaining channels. Our variant avoids these problems and restrictions by normalizing channels divided into groups with *flexible* dimension (Hence its name, *Flexible Group Normalization* or FGN) and empirically favoring the method convergence.

To validate the method, we conducted different experiments on the CIFAR-10 dataset [20] (60'000 32x32 color images) using the class-unconditional architecture (3 level U-Net), the `jax` [3] code base as well as the checkpoints of the 64 to 1 step models made available by the authors of [32]. Firstly, it was necessary to produce some preliminary results to establish the pruning method hyperparameters. Among them, we include the pruning ratio and the way it should be differentiated based on the network macro-blocks and the distillation iteration. To this end, a per-macro-block pruning sensitivity analysis as done in [19] was performed, where each macro-block is pruned in isolation and after that, the FID score of the network was computed. Also, a weight variation analysis was performed, comparing the difference between the student and the teacher model at each distillation iteration. Other experiments were required to select the pruning scoring function for the weights between the traditionally used L1 Norm [22], one based on the Taylor expansion of the disruption loss presented in [9] (which evaluates the deterioration of the loss if a weight is removed) and an adaptation of the latter on our specific case. This adaptation dynamically mixes the student disruption loss with the teacher one, which is intuitively deemed to be more reliable at the beginning of the distillation iteration. Finally, we also validate the efficacy of the FGN. We follow the results of these experiments to decide the method parameters and choices and we propose its final results, compared with the unpruned models in terms of FLOPs required from the diffusion process, number of weights and sample quality, using the FID score [14].

The thesis is organized as follows: Chapter 2 exhibits an exhaustive description of diffusion models with their advantages and disadvantages, as well as an *excursus* on the two main techniques employed in this work - Knowledge Distillation and Model Pruning. In Chapter 3 the alteration of the *Progressive Distillation* method, the employed network architecture, and the explanation of the reasons and implementation of the FGN are presented in detail. In Chapter 4 we report the results of our experiments. Finally, the conclusion on the experiments and final conclusions on the work are presented in Chapter 5.

Chapter 2

Background

2.1 Diffusion Models

Diffusion Models (DMs) represent the state-of-the-art for Generative AI (Images, Audio, Video) as well as powerful tools for Reinforcement Learning (RL), Bioinformatics and Black Box Optimization [5]. They were first presented in the seminal work by Sohl-Dickstein et al. [33], where taking an idea first applied in thermodynamics, they describe a way to create probabilistic models that are both flexible and easily evaluable. In particular, Diffusion Models aim to learn a reversal function that converts samples from a known and tractable distribution (*e.g.* 2D Gaussian noise) to those of a target complex data distribution (*e.g.* the dataset image distribution). This is achieved by modeling two *Markovian processes* (Fig. 2.1) [16][32] :

- The forward process. It is a fixed process that adds noise to a datapoint x_0 sampled from the target distribution q. In this way, the latter is perturbed to an analytically tractable (usually Gaussian) distribution $p(x_T)$:

$$\begin{aligned} q\left(x_{1:T}|x_0\right) &:= \prod_{t=1}^{T} q\left(x_t|x_{t-1}\right), \\ q\left(x_t|x_{t-1}\right) &= \mathcal{N}\left(x_t; \left(\alpha_t/\alpha_{t-1}\right) x_{t-1}, \sigma_{t|t-1}^2 \boldsymbol{I}\right) \end{aligned} \tag{2.1}$$

where t indicates the ”time-step” of the process, $\sigma^2_{t|t-1}$ equals to $\left(1 - e^{\lambda_t - \lambda_{t-1}}\right) \sigma_t^2$ and α_t, σ_t are differentiable functions such that the log signal-to-noise ratio $\lambda_t = log\left(\alpha_t^2 / \sigma_t^2\right)$ is monotonically decreasing with t increasing. The definition is usually given in continuous time (i.e. $t \in [0, 1]$), but can be easily adapted to the discrete formulation, which is fitting for the actual implementation.

- The reverse process. As the name suggests, this process ”reverts” the previous one by sampling data from the tractable Gaussian $p(x_T)$ and then repeatedly ”denoising” it to obtain a point in the $q(x_0)$ distribution:

$$\begin{aligned} p\left(x_{0:T}\right) &\coloneqq p\left(x_T\right) \prod_{t=1}^{T} p(x_{t-1}|x_t), \\ p(x_{t-1}|x_t) &\coloneqq \mathcal{N}\left(x_{t-1}; \mu\left(x_t, t\right), \Sigma\left(x_t, t\right)\right) \end{aligned} \tag{2.2}$$

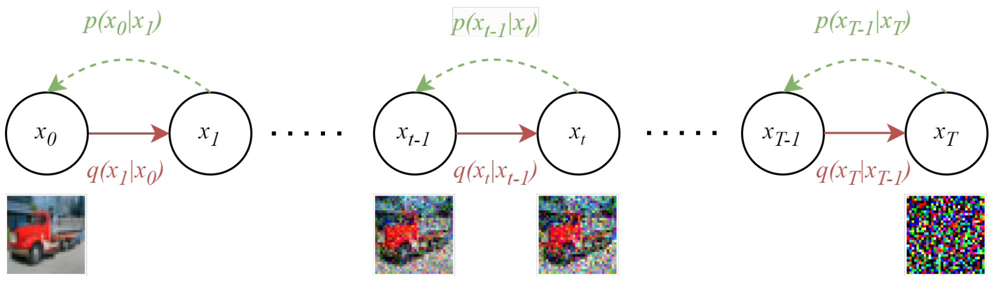


Figure 2.1: *Representation of the forward (in red) and reverse (in green) diffusion process.*

Since $p(x_{t-1}|x_t)$ cannot be analytically obtained, it is estimated using a neural network with parameters θ. The main objective of this network is to learn the *score function* of the unknown data distribution $q(x)$, which is defined as $\nabla_x \log q(x)$ or, equivalently, of the perturbation kernel $q(x_t|x_0)$ as in the so-called *Denoising score matching* [38]. The latter allows the computation of the network training objective without explicitly using the ground truth value of the score function (which is, as its reference distribution, unknown), exploiting finite samples. The corresponding loss function is a re-weighted variant of the Evidence Lower Bound (ELBO) [35] (in discrete formulation):

$$\boldsymbol{\theta}^* = \underset{\theta}{\operatorname{argmin}} \sum_{t=1}^{T} w(t)\mathbb{E}_{q(x)}\mathbb{E}_{q(x_t|x_0)} \left[\|\nabla_{x_t} \log q(x_t|x_0) - s_\theta\left(x_t, t\right)\|_2^2 \right] \tag{2.3}$$

where $w(t)$ is a weighting function (usually proportional to the SNR). Since $q(x_t|x_0)$ is Gaussian, then the score admits an analytical form [5][38]:

$$\nabla_{x_t} \log q(x_t|x_0) = -\frac{x_t - \alpha_t x_0}{1 - \alpha_t^2} \tag{2.4}$$

Moreover, we can see how the score value corresponds to the noise added to the image [5]. Thus the network practically tries to predict either the noise ϵ added to the original image up to a certain timestep or a denoised version of the image, given a noisy one, after t steps of forward process. In the latter case, the loss is re-written as [32]:

$$L_\theta = |\epsilon - \epsilon_\theta(x_t, t)| = \frac{\alpha_t^2}{\sigma_t^2}|x_0 - \hat{x_\theta}(x_t, t)| \tag{2.5}$$

The network can be trained also with conditioning, to obtain samples that reflect some specific properties (text-based generation, image inpainting, style transfer) [30]. This is usually done by incorporating class information into normalization layers [7] or cross-attention modules [30], and the quality of these samples can be improved by adding guidance from a classifier [7] or adaptive weights [17]. However, we will focus on unconditional models, thus further details on this topic are out of the scope of the work.

After the network is trained, it is then possible to sample new data in $q(x)$ by first sampling a point x_t from $p(x) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and then applying ancestral sampling [32][35]. This technique considers that the forward process can be written in ”reverse form” as:

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_{t-1|t}(x_t, x_0), \tilde{\sigma}_{t-1|t}^2 \boldsymbol{I}) \tag{2.6}$$

where:

$$\begin{aligned}\tilde{\sigma}^2_{t-1|t} &= (1 - e^{\lambda_t - \lambda_{t-1}})\sigma^2_{t-1} \\ \tilde{\mu}_{t-1|t}(x_t, x_0) &= e^{\lambda_t - \lambda_{t-1}}(\frac{\alpha_{t-1}}{\alpha_t})x_t + (1 - e^{\lambda_t - \lambda_{t-1}})\alpha_{t-1}x_0\end{aligned} \tag{2.7}$$

From this observation, the rule for ancestral sampling is then defined as follows:

$$\begin{aligned}x_{t-1} &= \tilde{\mu}_{t-1|t}(x_t, \hat{x}_\theta(x_t)) + \boldsymbol{\epsilon}\sqrt{(\tilde{\sigma}^2_{t-1|t})^{1-\gamma}(\sigma^2_{t|t-1})^{\gamma}} \\ &= e^{\lambda_t - \lambda_{t-1}}(\frac{\alpha_{t-1}}{\alpha_t})x_t + (1 - e^{\lambda_t - \lambda_{t-1}})\alpha_{t-1}\hat{x}_\theta(x_t, t) + \boldsymbol{\epsilon}\sqrt{(\tilde{\sigma}^2_{t-1|t})^{1-\gamma}(\sigma^2_{t|t-1})^{\gamma}}\end{aligned} \tag{2.8}$$

where $\boldsymbol{\epsilon}$ is Gaussian noise, γ is a hyperparameter controlling the quantity of noise added during sampling. This is the basic way to obtain samples from the network, but there exist improved methods. Some examples are the DPM-Solver++ [25] and DEIS [42] which focus on reducing the number of sampling timesteps by improving the way Ordinary/Stochastic Differential Equations (ODE/SDE) associated to the diffusion process are solved. Another important method is the one presented in [34], which is also employed in this contribution. Song et al. extended the concept of Denoising Diffusion Probabilistic Models (DDPMs) from the Markovian reverse process to a non-Markovian one without changing the objective function used to train the model. In this way, any DDPM can be treated as Denoising Diffusion Implicit Model (DDIM) and allows to adopt a deterministic sampling procedure that produces better samples at a faster rate by simply modifying the variance parameter $\tilde{\sigma}^2_{t-1|t}$ for the reverse process. In fact, the definition provided in Eq. 2.6 and taken from [32] is already extended to describe also non-Markovian distributions, as each x_{t-1} depends both on x_t and x_0. Since the variance is set as shown in Eq. 2.7, the process is rendered Markovian [34], but if it is set to 0 for all t, we obtain the deterministic sampling process as adapted in [32] from [34]:

$$\begin{aligned} x_{t-1} &= \alpha_{t-1}\hat{x}_\theta(x_t,t) + \sigma_{t-1}\frac{x_t - \alpha_t\hat{x}_\theta(x_t,t)}{\sigma_t} \\ &= e^{(\lambda_t-\lambda_{t-1})/2}(\frac{\alpha_{t-1}}{\alpha_t})x_t + (1 - e^{(\lambda_t-\lambda_{t-1})/2})\alpha_{t-1}\hat{x}_\theta(x_t,t) \end{aligned} \tag{2.9}$$

The way this technique accelerates sampling resides in the possibility of considering a generative process that does not approximate all the steps of the reverse process (and, so, of the forward one), as it is not required anymore by its non-Markovian nature (the strict dependency of a state to its previous one is lost). Moreover, since the sampling process does not include the addition of random noise ϵ, the result of every step is determined given the sample at a previous time-step. Thus, the generating process takes the form of a determined ”sampling trajectory” [34] which may span over a subset of the T steps of the corresponding forward process. Whenever this subset is much smaller than T, we then obtain an acceleration of the generation process.

These are a few examples of how in the DMs research field most of the efforts are put toward improving the efficiency and speed of sampling. DMs surpassed GANs considering the sample quality and the relatively higher training stability [7] but still fall behind if one considers inference speed due to the iterative nature of their sampling method. Thus, to the best of our knowledge, to reduce the number of timesteps required without affecting quality past works have either tried to develop new sampling techniques (which is difficult, considering that the theory around DMs is limited and complex [5]) or to apply alternative approaches, such as distillation over the timesteps as in [32].

2.2 Progressive Distillation

Model Distillation or Knowledge Distillation (KD) was introduced by Hinton et al. in [15] as a mean to ”condense” the knowledge from a resource-intensive ”teacher” model or ensemble to a smaller ”student” model. In the context of classification, the student model is trained by optimizing the cross-entropy loss between its output and both the ground truth and the output of the teacher model. Moreover, the term of the loss dealing with the latter employs high-temperature softmax to turn the teacher output distribution into ”soft labels”. These allow the student model to learn subtle differences in the class distributions that are not present in the GT. For example, if a picture contains both a cat and a dog but is labeled only as a picture of a cat, the teacher model would return a less certain probability distribution. Typically, this leads to improved outcomes in the student model, along with a more compact and efficient structure.

In the field of Diffusion Models, the KD has been mostly applied as a mean to obtain networks with higher sampling speed. To the best of our knowledge, the first kind of this contribution is the one by Luhman & Luhman [26], where a pre-trained DM is distilled into a single-step student network. This technique involves initializing the latter with the weights of the teacher (which speeds up convergence and favors knowledge transfer) then, for each x_T sampled point to be used as input for the training, the teacher model is fully run on x_T to produce $F_{teacher}(x_T)$ with DDIM sampling, which is deterministic. Then, the student network is trained with the usual loss (Eq. 2.5) but forcing its output after a single pass to match the many-step result $F_{teacher}(x_T)$. The results show that the student yields competitive results to some other SOTA model with comparable or considerably increased sampling speed.

Another independently developed work by Salimans & Ho, on which this work heavily relies, proposes a similar approach that yields better results and reduces some possible scalability issues of Luhman & Luhman’s one. Said method is named *Progressive Distillation* [32], as it requires the student to compress two teacher’s DDIM timesteps in a single one and, once it is trained, it becomes the teacher for a new student, allowing a process that *progressively halves* (starting from 1024) the number of required sampling steps by each network.

The algorithm can be summarized as follows [32]:

Algorithm 1 *Progressive Distillation* algorithm

```
Require: Trained teacher model $\hat{\boldsymbol{x}}_\eta(\boldsymbol{x_t})$, Dataset $\mathcal{D}$, Loss weight function
  $w()$, noise scheduling functions $\alpha$ and $\sigma$, student sampling steps $N$

  for $K$ iterations do
    $\theta \leftarrow \eta$                                   ▷ Initialize student with teacher $\eta$
    while not converged do
      $\boldsymbol{x} \sim \mathcal{D}$
      $t \leftarrow i/N,\ i \sim Cat\,[1, 2, \ldots, N]$
      $\epsilon \sim N(0, \boldsymbol{I})$
      $\boldsymbol{x_t} \leftarrow \alpha_t \boldsymbol{x} + \sigma_t \epsilon$                ▷ Add random noise to initial sample

      $t' \leftarrow t - 0.5/N, \quad t'' \leftarrow t - 1/N$    ▷ 2 DDIM steps with teacher $\eta$
      $\boldsymbol{x_{t'}} \leftarrow \alpha_{t'} \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_t}) + \frac{\sigma_{t'}}{\sigma_t}(\boldsymbol{x_t} - \alpha_t \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_t}))$
      $\boldsymbol{x_{t''}} \leftarrow \alpha_{t''} \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_{t'}}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\boldsymbol{x_{t'}} - \alpha_{t'} \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_{t'}}))$
      $\tilde{\boldsymbol{x}} \leftarrow \frac{\boldsymbol{x_{t''}} - (\sigma_{t''}/\sigma_t)\boldsymbol{x_t}}{\alpha_{t''} - (\sigma_{t''}/\sigma t)\alpha_t}$                ▷ Build target for student

      $\lambda_t \leftarrow \log\left[\alpha_t^2/\sigma_t^2\right]$
      $L_\theta \leftarrow w(\lambda_t)|\tilde{\boldsymbol{x}} - \hat{\boldsymbol{x}}_\theta(\boldsymbol{x_t})|_2^2$
      $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
    end while
    $\eta \leftarrow \theta$                                   ▷ Update teacher with student
    $N \leftarrow N/2$                                          ▷ Halve sampling steps
  end for
```

It is important to note that, differently from Luhman & Luhman's approach, this algorithm never requires to run the teacher model for its total number of sampling steps. In this way, fewer resources are required to run this process. Another difference resides in the student model being trained to produce either $\hat{x}_\theta(x_T)$ or the "velocity" vector $v = \alpha_t \epsilon - \sigma_t \boldsymbol{x}$, which stabilize and ensure convergence at fewer time-steps. Moreover, *Progressive Distillation* allows control of the tradeoff between sample quality and model efficiency. The method generally allows a speed-up of 64x with almost no loss in sample quality across different datasets, but from the 8-step model to the single step one, the quality remarkably decreases. *Progressive Distillation* was also extended to Latent Diffusion Models, classifier-free guided DMs and to employ a stochastic sampling technique in [27]. There is, however, a constant element in all these works, that is: the student model is always initialized with the weights of the teacher, thus forcing the former to have the same architecture as the latter. This excludes the possibility of reducing the dimension of the architecture to further gain efficiency in terms of FLOPs and sampling time. So, the primary goal of this contribution is to explore ways to bypass or dampen this restriction and investigate the impact on performances.

2.3 Model Pruning

With ”Model Pruning” we refer to a class of techniques developed to optimize neural networks. As the name suggests, they entail the removal or ”pruning” of unnecessary components (usually weights) according to a chosen criterion to ”slim” the network. The number of weights to remove is usually indicated with the *pruning ratio*, a relative measure computed as the ratio between the number of removed weights and the original number of weights of the network. The foundations on which pruning is based are the empirical observations suggesting that large models enclose some degree of redundancy. So, it is possible to exploit this aspect to make the model more resource-efficient while maintaining or improving performances, thanks to the generalization power contained in the removal of too specialized weights. These techniques can have a lot of differences among them, depending on the application context and the specific task objective. This leads to a very complex taxonomy, as well as to a lack of standardized datasets, metrics, and experimental practices [2]. To give an overview of these methods, it is important to remark that they can exhibit differences along two orthogonal directions: ”how” to apply the pruning and ”when” to apply it. Regarding ”how” to apply pruning, the techniques could be distinguished by:

- *Structure and granularity*. Pruning can be performed at different degrees of granularity, starting from the atomic unit of the weights up to filters, layers, or modules. When it is sparsely applied only on weights, it is referred to as *unstructured* pruning. An example is given in [21], where each parameter is associated with a saliency score based on the hessian of the loss function w.r.t. a perturbed version of the parameter (*i.e.* emulating its removal) . The network is iteratively trained and pruned by setting to 0 and freezing parameters with low scores. Differently, methods focusing on more complex substructures are referred to as *structured* methods. An example is given by Li et al. [22], where they propose a technique to remove full convolutional filters with a low sum of the L1 Norm of their kernel weights. They also underline the advantages of structured pruning over unstructured techniques. If unstructured pruning can precisely locate and remove redundant weights, thus possibly yielding better performances, it still requires *ad hoc* hardware or libraries dealing with sparse calculus to effectively reduce inference time on more complex structures like convolutional layers. On the contrary, structured pruning allows the removal of entire filters or blocks such that the gain in performances can be achieved with standard Neural Network frameworks like `pytorch` or `tensorflow`, but posing new challenges on how to aggregate or compute scores for the networks sub-structures.

- *Scope and uniformity*. Methods may differ in the way different sections of the network are scored and pruned. Each parameter could be scored locally or globally, *i.e.* one can decide to remove a fraction of the weights with the lowest importance score w.r.t. other weights within a sub-structure (layer, block...) or within the whole network [2]. This entails that architectures may be pruned unevenly or also with heterogeneous pruning ratios across the model sub-structures. Kim et al. [19]

exploit this possibility to maximize pruning on a U-Net [31]. First, they conduct a *sensitivity analysis* to verify how the performance deteriorates when each structure is pruned in isolation. Then, when trimming the whole network, each block is pruned with a ratio that is proportional to its ”insensitivity”. In this way, they achieve an overall higher weight reduction and better retain the performances when compared to uniform pruning.

- *Pruning criterion*. Every method requires some kind of criterion to select the weights to be removed. These rules may take the form of a fixed function or can be learned. In the first kind, we may find the *random pruning* heuristic which randomly shuts down a fraction of the weights and might be used as a baseline or debugging indicator [5]. Another example is *magnitude pruning*, where each weight/filter is associated with a score equal to its L1 norm (or L2), as in [22] or [19]. This allows the removal of weaker weights which produce low activation values that, in turn, loosely influence the inference process. Because of this, the norm function is sometimes computed only on the weights of the normalization layers [24], since among their purposes they learn a scaling factor γ to be applied on the activation map. Finally, within the fixed criteria, we may find the gradient-based ones (as in [21] or [9]), which try to minimize the loss disruption brought by the reduction of the network capacity. In the case of criteria requiring learning, the most common choices are either to add a learned scaling parameter to each channel, such that, at the end of the training phase, the channels with the lowest associated parameter can be safely removed (*e.g.* [39]) or to perform Neural Architecture Search via Reinforcement Learning [13], or genetic search algorithms [23]. Generally, these kinds of rules allow an elegant end-to-end process to obtain a pruned model but, when compared with fixed criteria, are less general and more computationally demanding as they require training a network with a specific architecture from scratch.

While the aspects to consider about the ”when” to apply pruning are:

- *Scheduling*. Network compression may be performed in multiple steps during the training process or in a single step at its end or even beginning. Some efforts were put into researching the possibility of pruning neural networks at initialization or particularly early in the training. A famous example is the *lottery ticket hypotesis* by Frankle and Carbin [10], that states that ”A randomly-initialized, dense neural network contains a sub-network (the ”winning pruning lottery ticket”) that is initialized such that —when trained in isolation— it can match the test accuracy of the original network after training for at most the same number of iterations, based on the idea that an untrained architecture already encloses in itself optimal sub-structures”. So their work focuses on training the network for a few epochs, performing *magnitude pruning* and then restarting training with the initial weights. Similarly, Renda et al. [29] propose *learning rate rewinding* which restarts the learning rate schedule after pruning instead of re-initializing the weights. These, as many other techniques, bring good results with a little to no overhead on the training but are still beaten by ”after-training” techniques [11].

- *Fine-tuning requirement*. Some methods may require additional fine-tuning after the removal of weights to recover some of the lost performance. These techniques are usually more general as they could be also applied to pre-trained foundational models, even if they require some training overhead. Other methods may propose a training process that allows pruning *by design*. Some examples are the works implementing a learned criterion like in [39] or introducing some regularization during training like CentripetalSGD [8]. In the first case, the values of the activation maps in unimportant channels are already dampened by the learned scaling factor during training so they can be safely removed. In the second one, SGD is modified allowing to learn groups of redundant filters where, at the end of training, only one is preserved from each group.

Network Pruning can be a powerful tool to use to increase the efficiency of a model and it works 'orthogonally' w.r.t distillation techniques. Aghli et al. [1] proposed a method to compress deep convolutional networks which consists of pruning a pre-trained network using the Average Percentage of Zeros (percentage of zero activations of a neuron after the ReLU) saliency score, fine-tuning it and then distilling it in a smaller student network to further reduce the dimension of the model. We also already briefly mentioned the work by Vo et al. [39] where a GAN network is progressively pruned during training, while simultaneously transferring knowledge from a considerably bigger teacher to better preserve performances. The positive results of both these works show that it is possible to mix pruning and distillation to further exploit their strengths.

Chapter 3

Proposed Method

3.1 Introducing pruning to *Progressive Distillation*

Our proposed method extends the seminal work by Salimans & Ho on *Progressive Distillation* [32]. In particular, we try to explore the possibility, expressed as possible future work in the paper, to adapt the distillation framework to allow student network with a different architecture w.r.t the teacher network. In particular, we try to obtain a smaller and, thus, more efficient student. The main obstacle to this is the fact that using the same architecture helps convergence, which is central for the training of complex networks as Diffusion Models. To work around this problem, we tried to integrate a simple magnitude pruning in the distillation process to reach our goal. So we modified Algorithm 1 as shown in Algorithm 2.

We introduce a pruning ratio to be achieved by the end of each distillation iteration. This ratio is further divided and applied in smaller steps as to introduce a *progressive pruning* phase within each diffusion step. This was inspired by works such as [39] and [18] since not only does one-shot pruning require fine-tuning and may be too aggressive, but also has been proven ineffective when applied to other generative models such as GANs [39]. Furthermore, it is important to notice that the pruning ratio r_p is a hyperparameter which can be chosen *a priori*, allowing to control the compression vs. quality

Algorithm 2 *Progressive Distillation* w/ pruning algorithm

```
Require: Trained teacher model $\hat{\boldsymbol{x}}_\eta(\boldsymbol{x}_t)$, Dataset $\mathcal{D}$, Distillation iterations K,
  Training steps S, Loss weight function $w()$, noise scheduling functions $\alpha$
  and $\sigma$, student sampling steps $N$, pruning function $f_p()$, iteration pruning
  ratio(s) $r_p$, train steps between pruning $s_{tp}$, pruning steps $s_p$

  for $K$ iterations do
      $\theta \leftarrow \eta$                                  ▷ Initialize student with teacher $\eta$
      for $S$ training steps do
          $\boldsymbol{x} \sim \mathcal{D}$
          $t \leftarrow i/N,\ i \sim Cat\,[1, 2, \ldots, N]$
          $\epsilon \sim N(0, \boldsymbol{I})$
          $\boldsymbol{x_t} \leftarrow \alpha_t \boldsymbol{x} + \sigma_t \epsilon$                 ▷ Add random noise to initial sample

          $t' \leftarrow t - 0.5/N, \quad t'' \leftarrow t - 1/N$     ▷ 2 DDIM steps with teacher $\eta$
          $\boldsymbol{x_{t'}} \leftarrow \alpha_{t'} \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_t}) + \frac{\sigma_{t'}}{\sigma_t}(\boldsymbol{x_t} - \alpha_t \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_t}))$
          $\boldsymbol{x_{t''}} \leftarrow \alpha_{t''} \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_{t'}}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\boldsymbol{x_{t'}} - \alpha_{t'} \hat{\boldsymbol{x}}_\eta(\boldsymbol{x_{t'}}))$
          $\tilde{\boldsymbol{x}} \leftarrow \frac{\boldsymbol{x_{t''}} - (\sigma_{t''}/\sigma_t)\boldsymbol{x_t}}{\alpha_{t''} - (\sigma_{t''}/\sigma t)\alpha_t}$                        ▷ Build target for student

          $\lambda_t \leftarrow \log\left[\alpha_t^2/\sigma_t^2\right]$
          $L_\theta \leftarrow w(\lambda_t)|\tilde{\boldsymbol{x}} - \hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x_t})|_2^2$
          $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
          if (current training step % $s_{tp}$) = 0 then
              $\theta \leftarrow f_p(\theta, r_p/s_p)$
          end if
      end for
      $\eta \leftarrow \theta$                                  ▷ Update teacher with student
      $N \leftarrow N/2$                                    ▷ Halve sampling steps
  end for
```

and the compression vs. distillation tradeoffs, depending on the task. We exploited this possibility in our experiments to decrease the compression ratio once the distillation process reaches the last iterations, where the model is put under maximum compression stress. Moreover, since the pruning method is applied *post hoc*, we can leverage pre-distilled models as starting teachers. In fact, from [32] we can see that the chain of teacher/student models tends to retain the original model quality up to 64 and 32-steps models. Consequently, the method can be applied also stating from these models, further reducing training and time resources requirements.

3.2 Network structure and Pruning criteria

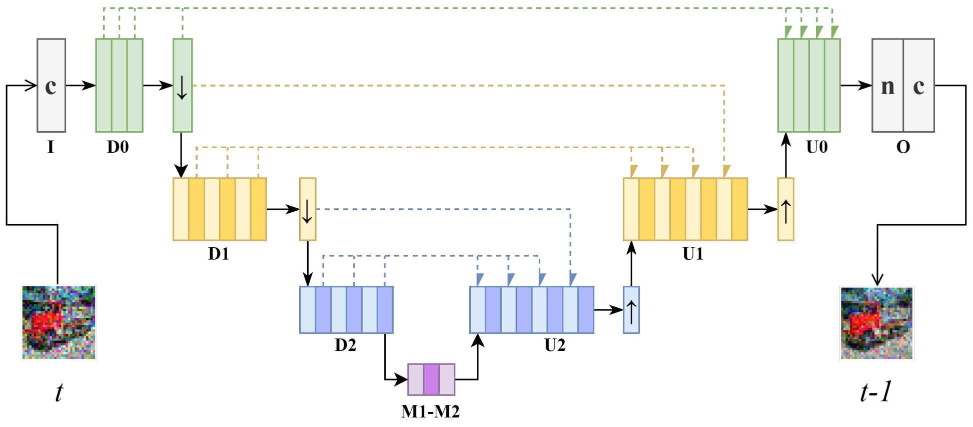
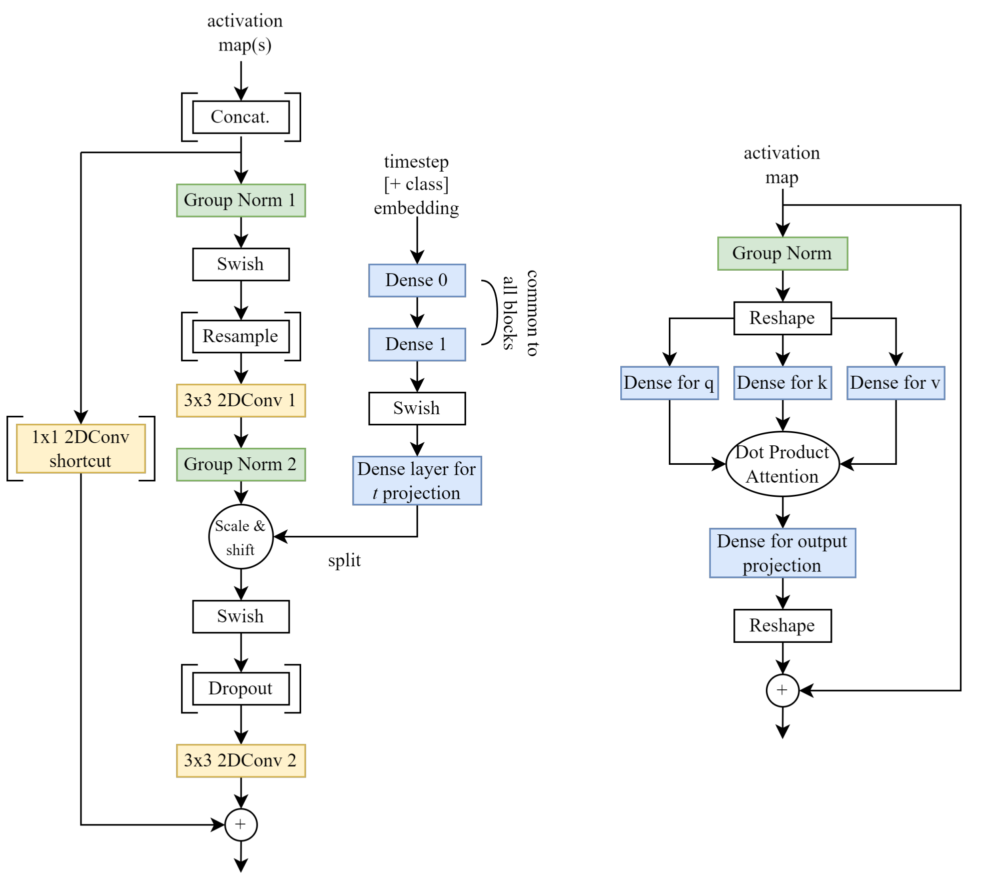


Figure 3.1: *Representation of an instance of the employed U-Net.* Each rectangle represents a ResNet (brighter color) or Residual Attention block (darker color). Each level is marked with a different color. Boxes with ↑ and ↓ are respectively upsampling and downsampling blocks, while those marked with ”c” and ”n” are 2D convolutional layers or Group Norm layers.

The pruning method we introduce is *rule-based* and structured, so to obtain performance gains without employing specific software or hardware for sparse computation. Although it was tailored around the network architecture used in the reference paper, it can still be generalized and adapted to other variants or structures. Said structure is a U-Net [31] as the one used in [32] and [16]. It is composed of different levels, each of them made of alternating ResNet [12] blocks and self-attention [37] residual attention blocks. At the end of each level, the resolution is halved (downward path) or doubled (upward path) using a resampling ResNet block as in [4]. An example of the architecture as well as the detailed composition of the blocks are presented in Figures 3.1 and 3.2.



(a) *ResNet Block*. The layers/operations in square brackets are either optional (dropout) or used in specific blocks (resample for re-sampling blocks, shortcut and concatenation for up-path blocks)

(b) *Residual Attention Block*. Activation maps are flattened for attention computation.

Figure 3.2: *Detailed representation of the blocks employed in the U-Net.*

Considering this network structure, we can list the rules applied to prune the network by the function f_p (Alg. 2):

- Pruning is performed following the *Magnitude pruning* paradigm, where every filter or neutron is associated with a saliency score through a scoring function s. In fact, a filter or neuron $\mathcal{K}$ is pruned if the score:

$$s(\mathcal{K}) = \sum_{W_k \in \mathcal{K}} |W_k| \tag{3.1}$$

(where W_k are its weights) is among the lowest when compared with the scores of the other sub-structures belonging to the same layer. The scope of the scoring function s is restricted to the layer so that the ones in the same macro-block have the same pruning ratio applied to them.

- The removal of the sub-structures is not performed in a single step, but one layer at a time, starting from the input one and following the main signal path. The scoring is performed ”greedily”, meaning that neurons/filters in a given layer are scored after the elimination of neurons/filters in the previous structure. In fact, after pruning a layer, a pruning mask is propagated to the immediately next layer(s) to prune its input channels (Figure 3.3). This criterion was shown to be more effective than cutting and scoring layers in isolation in [22]. Moreover, it also allows the layers closer to the input to be pruned with more ”independence” w.r.t. the others, which intuitively helps preserving quality. This happens because the perturbations added to the signal at the

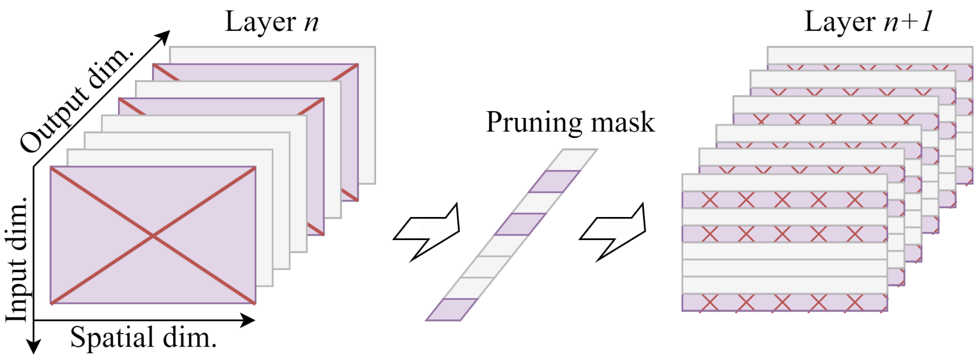


Figure 3.3: *Representation of the pruning propagation through masks*. This procedure ensures signal consistency and enables ”greedy” pruning.

beginning of the network pass are the most impacting ones on the performance, so its crucial to perform correct, precise and unconditioned choices when removing information in this stage.

- For obvious reasons, input and output channels are not pruned in order to avoid heavy damage to the input signal and obtain well formed color images;

- A pruning mask is propagated also through skip connections within a ResNet block and between networks macro-blocks. In the latter case, the initial pruning mask for the ResNet block is obtained by concatenating the one propagated from the down-branch and the one yielded by the immediately previous block.

- If the pruning ratio is different between structures where a mask is being propagated, the latter is either filled or further cut. Activation maps are expanded with zero-channels or compressed accordingly during inference. This allows to keep consistence between the activation maps dimensions.

- When removing the filters from the first convolutions (2DConv 1) and the dense timestep embedding layer in a ResNet block (Fig. 3.2a), their saliency scores are not obtained by aggregating the scores of their kernel weights or neurons. Instead, they are assigned the score of the corresponding neurons of the second Group Normalization layer (Group Norm 2), following the works on Norm-based pruning [24][18]. So, for example, if one considers the convolutional filter responsible for the computation of the n^{th} output channel, the saliency score associated to it will be computed from the learned parameters for the n^{th} channel of the Group Normalization Layer. The same will happen for the n^{th} and the $k + n^{th}$ neurons (where k is equal to half of the output dimension of the layer) of the dense timestep embedding layer.

- The attention blocks are pruned by scoring together and removing triplets of corresponding neurons used for *k*, *q*, and *v* computation. Also, each head is independently pruned (Fig. 3.4).

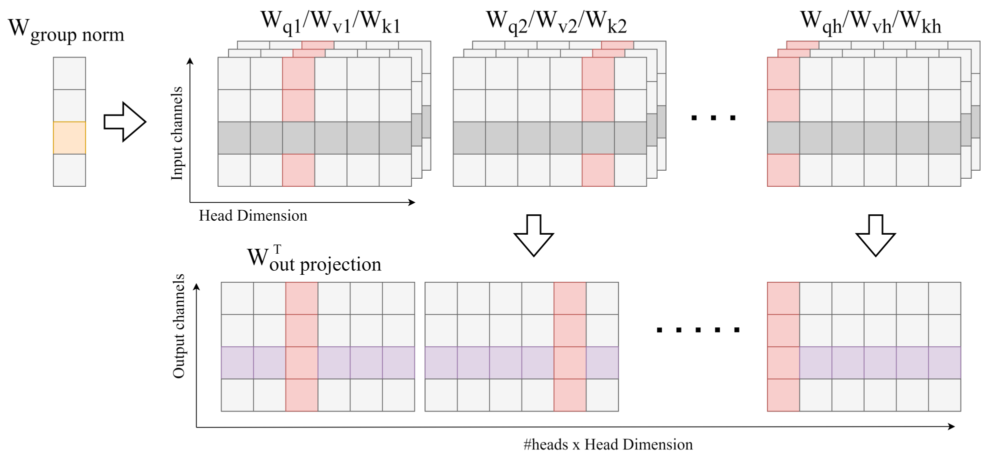


Figure 3.4: *Graphical representation of the attention block pruning.* Neurons with the same colors (purple, red) are scored together(refer to Fig. 3.2b for details). Gray neurons are removed as a consequence of pruning mask propagation.

The pruning ratio is not applied uniformly to the whole network, but following [19], we perform a sensitivity analysis to detect which macro-blocks are less or more sensible to pruning. In order to do so, each block is pruned independently and in isolation w.r.t. the others with different pruning ratios, the network performances are compared with the unpruned network considering the FID score [14], without performing fine-tuning. Thus, considering the sensitivity of each macro-block and the desired network dimension, the pruning ratios r_p can be determined.

Finally, the weight scoring function used at the beginning of the development and for the sensitivity analysis was the L1 Norm, as it is commonly used for its simplicity for magnitude pruning. In a second moment, the scoring function proposed by Fang et al. in [9], was adopted as it produced empirically better results. Said function falls in the gradient-based category, as it tries to individuate those structures that, when removed, minimize the loss disruption:

$$\min_{\theta'} \left| L_{\theta'} - L_{\theta} \right|, \qquad s.t. \left\| \theta' \right\|_0 \leq 1 - r_{pg} \tag{3.2}$$

where r_{pg} is the global pruning ratio. By exploiting the Taylor expansion to approximate the loss disruption, they find the following score for the substructure $\mathcal{K}$:

$$s(\mathcal{K}, x) = \sum_{W_k \in \mathcal{K}} \left| W_k \cdot \sum_t \nabla_{W_k} L_t(\theta, x) \right|, \quad s.t. L_t / L_{max} \geq \mathcal{T} \tag{3.3}$$

So, at each pruning round, the network is run on the input x in order to accumulate gradients over different timesteps. However, not all of the latter are exploited as the accumulation is stopped once the ratio between a given step loss and the maximum recorded step loss falls under a threshold $\mathcal{T}$. This is done to avoid considering noisy and redundant gradients that can be found toward the end of the generation process [9]. Moreover, some experiments were run to try mixing information coming from both the student and the teacher network in a dynamic manner during a single distillation step. The intuition behind it was to consider more reliable information coming from the teacher network at the beginning of the pruning since it had a more ”stable” architecture and slowly transferred importance to the student network once it was reaching its final form. These experiments yielded worse results, so the information is strictly taken from the student model only.

3.3 Flexible Group Normalization

During the development and testing of the method, another important challenge emerged. It is linked to the presence of *Group Normalization* layers [40] in the network, which are more stable and batch-size-independent when compared to ”plain” batch normalization. In fact, normalization statistics are computed within fixed-sized groups of channels. In mathematical terms, given a feature x_{ijc} belonging to the c^{th} channel of an activation map, it normalizes it:

$$\tilde{x}_{ijc} = \frac{1}{\sigma_g}(x_ijc - \mu_g), \quad g = \lfloor \frac{c}{|G|} \rfloor \tag{3.4}$$

where σ_g and μ_g are the standard deviation and the mean of the g^{th} group of channels and $|G|$ is the group size. In more detail, σ_g and μ_g are computed as follows:

$$\mu_g = \frac{1}{|G| \cdot |A_m|} \sum_{k=g\cdot|G|}^{g\cdot|G|+|G|} \sum_{i,j} x_{ijk} \tag{3.5}$$

$$\sigma_g = \sqrt{\frac{1}{|G| \cdot |A_m|} \sum_{k=g\cdot|G|}^{g\cdot|G|+|G|} \sum_{i,j} (x_{ijk} - \mu_g)^2 + \epsilon} \tag{3.6}$$

where $|A_m|$ is the spatial dimension of the activation map to which the x_{ijk} belongs to (height $\times$ width) and ϵ is a small constant added for numerical stability. Then, as the other normalization technique, the layer learns a per-channel pair of weights γ and β to apply a linear transformation that can compensate a possible loss of representational ability [40]:

$$y_{ijc} = \gamma_c \tilde{x}_{ijc} + \beta_c \tag{3.7}$$

The application of pruning on architectures employing this type of normalization entails that the removal of some channels may cause the others to shift into different groups from the original one (Fig. 3.5). Also, due to implementation restrictions the number of remaining channels had to be a multiple of the group size or the group size had to be changed in order to fit this requirement.

This problem was initially avoided by executing tests on a sparse model, *i.e.* the pruned weights were frozen and set to 0. But, once the first tests were run on a compact model with restrictions on the pruning strategy, the reported results were comparably worse. So, the normalization layer was modified in order to accept variable-sized groups, based on the idea that moving channels

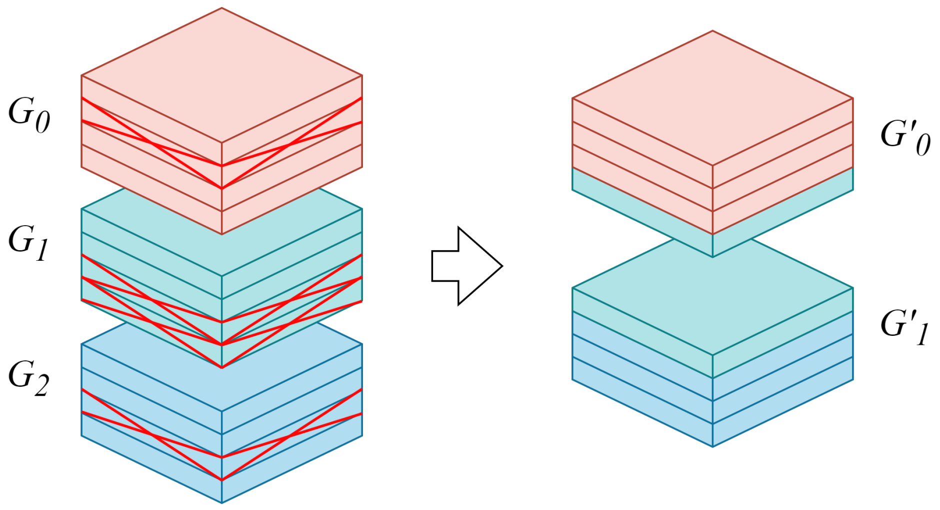


Figure 3.5: *Illustration of the group shifting problem.*

across different groups would highly destabilize the network. This also avoids adding more constraints to the pruning procedure. We name this variation as *Flexible Group Normalization*. From the point of view of its mathematical description, we modified the assigment of the normalization statistics to a given channel (Eq. 3.4) and their computation (Eqs. 3.5-3.6) in the following way:

$$\tilde{x}_{ijc} = \frac{1}{\sigma_g}(x_ijc - \mu_g), \quad c \in G'_g \tag{3.8}$$

$$\mu_g = \frac{1}{|G'_g| \cdot |A_m|} \sum_{k \in G'_g} \sum_{i,j} x_{ijk} \tag{3.9}$$

$$\sigma_g = \sqrt{\frac{1}{|G'_g| \cdot |A_m|} \sum_{k \in G'_g} \sum_{i,j} (x_{ijk} - \mu_g)^2 + \epsilon} \tag{3.10}$$

Where G'_g is g^{th} group after applying pruning. This modification was implemented by re-using some parts of the already existing Group Normalization layer and adding a mask initialized as a $C \times C$ block diagonal matrix (C is the number of channels of the activation map), with each block being a matrix of ones of dimension $|G| \times |G|$. This mask is used to correctly aggregate channels (the $k \in G'_g$ condition in Eq. 3.5 and 3.6) to compute the inter-group mean μ_g and standard deviation σ_g. In fact, at every pruning step, when a channel is removed from the normalization layer, the corresponding row and column are discarded, so that the activation map values are then aggregated for each group only within the remaining channels (Fig. 3.6).

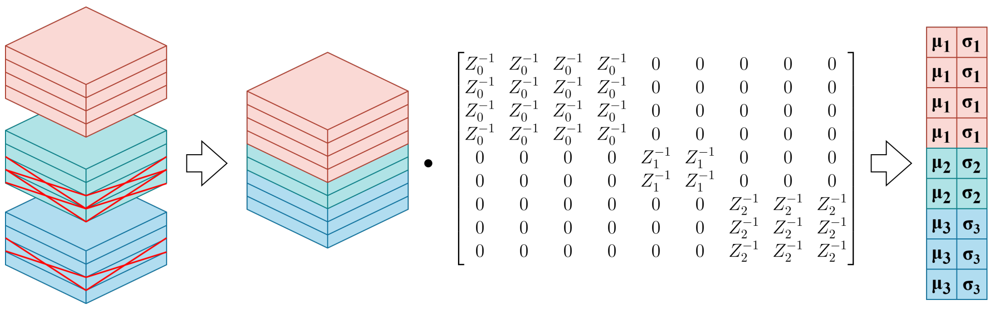

$\begin{bmatrix} Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & 0 & 0 & 0 & 0 & 0 \\ Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & 0 & 0 & 0 & 0 & 0 \\ Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & 0 & 0 & 0 & 0 & 0 \\ Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & Z_0^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Z_1^{-1} & Z_1^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Z_1^{-1} & Z_1^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Z_2^{-1} & Z_2^{-1} & Z_2^{-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & Z_2^{-1} & Z_2^{-1} & Z_2^{-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & Z_2^{-1} & Z_2^{-1} & Z_2^{-1} \end{bmatrix}$
μ_1 σ_1
μ_1 σ_1
μ_1 σ_1
μ_1 σ_1
μ_2 σ_2
μ_2 σ_2
μ_3 σ_3
μ_3 σ_3
μ_3 σ_3

Figure 3.6: *Implementation of the Flexible Group Normalization Layer.* Z_g^{-1} is equal to $|G'_g| \cdot |A_m|$.

Empirically we report slightly better results to those obtained with the sparse model, so we adopted it in the final models. We suppose that its success is due to:

- Reduction of the de-stabilization of the weights and normalization caused by channel shifting between groups;
- Retention of the previous weights and statistics as ”warm start” for the newly pruned model;
- Execution of the activation map normalization in a proper way, without considering zeroed channels in the equation.

Chapter 4

Results and ablations

In this Chapter we present the results of the experiments, starting from preliminary results and then presenting our final findings. We implemented the framework starting from the code base provided by the authors of [32] and modified where needed. Said code is mostly based on `Jax` [3], `Flax` (wrapper for deep learning) and `Optim` libraries. Most of the experiments were performed on a desktop equipped with two Nvidia RTX 2080 Ti GPUs with 11GB VRAM each (which `Jax` can exploit in parallel), Intel Xeon 3.9GHz W-2123 CPU, 64GB of RAM, and Ubuntu 20 as OS. Some of the latest experiments employed a cluster equipped with a Ryzen 9 3900X (3.8 GHz x 12 cores), 128 GB of RAM, two Nvidia RTX 3090 Ti, and Ubuntu 20 as OS. A full training required about two to three days in both settings. Moreover, some side tasks as sample generation were performed using a pro subscription on Google Colab with the TPU runtime option.

The reference paper on *diffusion ditillation* reports experiments on different popular datasets (ImageNet [6], LSUN [41]), but due to time requirements and available training resources, our experiments employed the CIFAR-10 [20] dataset, which is a ”Tiny Images” dataset subset with 60’000 independently labeled examples of 32x32 color images. For the same reasons, our contribution considers only unconditional image generation and the distillation procedure for the preliminary experiments was often truncated after 4 to

5 iterations, also due to their explorative nature (missing results are reported with an hyphen in the tables).

We evaluate the quality of our models with the Frechèt Inception Distance [14] generally computed on 30k samples (unless otherwise stated) and their efficiency considering the number of FLOPs per network pass and the number of weights in the architecture.

4.1 Training recipe and sensitiviy analysis

The training recipe and hyperparameter are drawn from those used in [32] for the distillation process. The number of channels for the activations in each level is kept at 256 and the attention blocks have a single head each. We employ Adam as optimizer with a learning rate set to 5e-5, gradient clipping to 1, and without weight or EMA decay. The noise schedule is set to the cosine one $\alpha_t = cos(0.5\pi t)$, the loss is computed on a ”$\mathbf{x}$-prediction” model and its reweighting is performed with the ”truncated SNR” [32] function:

$$L_\theta = \max\left(\frac{\alpha_t^2}{\sigma_t^2}, 1\right) \|\tilde{\boldsymbol{x}} - \hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t)\|_2^2 \tag{4.1}$$

The only two hyperparameters that are tweaked to meet our available resources and time requirements are the batch size, which is set to 128, and the number of training epochs per distillation step set to 50’000 for all the distillation iterations. It is the maximum number of parameter update employed in the paper, except for the 2 and 1 step models where they employ 100’000 epochs.

Our method introduces two new hyperparameters to set: the number of pruning steps per distillation iteration and the pruning ratio for each pruning step. Both were set in order to achieve a pre-determined pruning ratio in the last distillation step and to maximize performance. In order to do so, a coarse sensitivity analysis similar to the one in [19], was performed. Each of the

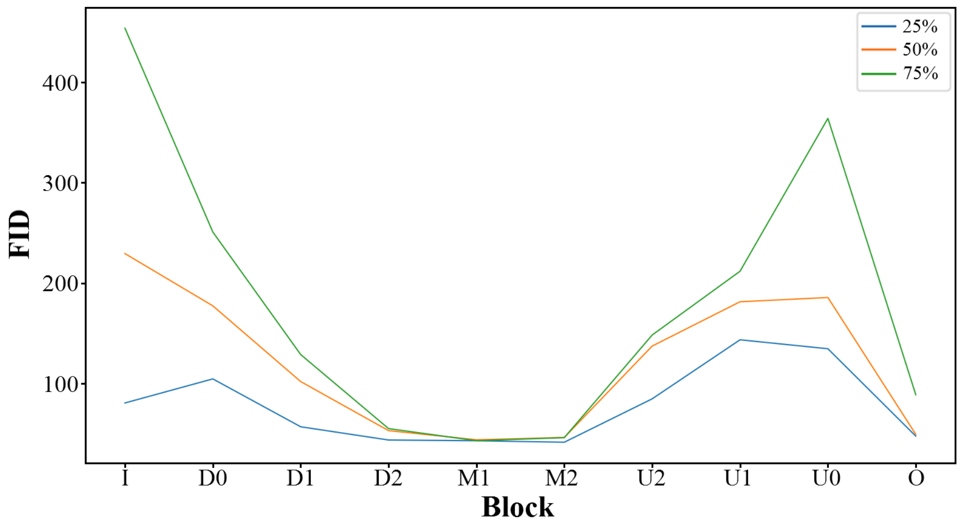


Figure 4.1: *Results of the pruning sensitivity analysis on the 32-steps model.*

network macro-blocks (marked with ”I/D#/U#/O” in Figure 3.1) was independently pruned with 25%, 50% and 75% pruning ratio and considering L1 Norm as pruning score. In this stage the network was kept sparse and attention blocks weren’t pruned. Also, after zeroing the weights, the model is run without applying any finetuning. For these reasons, the FID scores computed to test the macro-block sensitivity are notably very high. These scores were computed on around 20k samples and the results were plotted on a graph, shown in Figure 4.1.

As expected, and as also noted in [19], the middle section of the U-Net is more insensitive to pruning, as we assume it contains more redundant channels. To further validate this assumption, we also produced a sample for each pruned model and show the results in Figure 4.2.

As we can observe from the samples, the first interesting aspect is that independently of which part is being pruned and its pruning ratio, once the seed for the generation is fixed, the model always returns a sample with generally the same content (red truck, horse on grass, gray car). What does change depending on the damage brought to the network is the quality of the content. We can notice that any damage brought to the middle section of the network

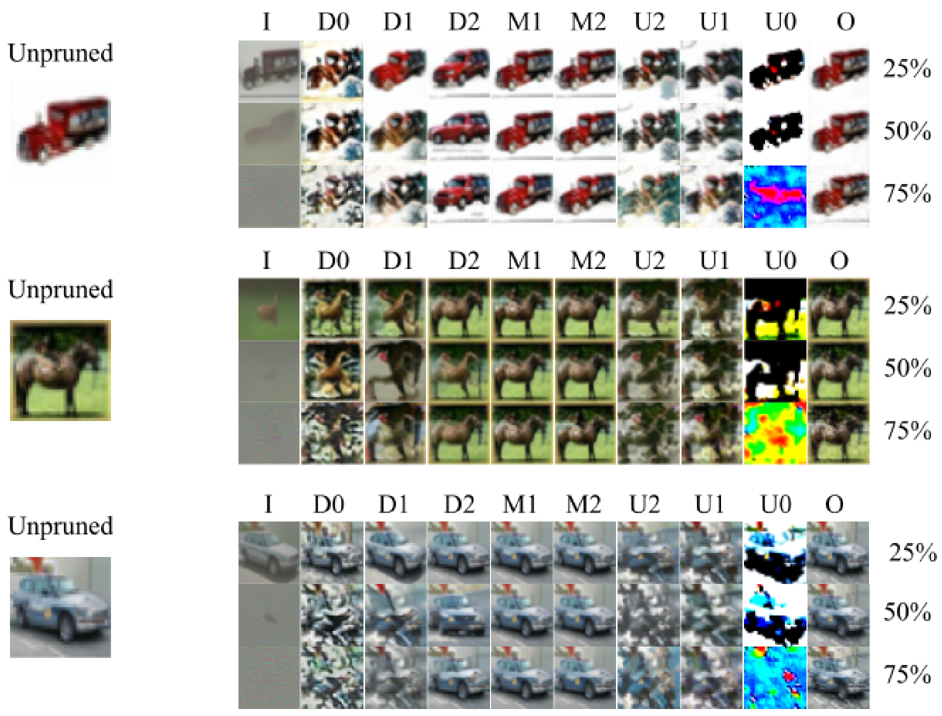


Figure 4.2: *Samples obtained with three different seeds after each round of pruning during sensitivity analysis*.

(M1-M2) yields samples that are almost indistinguishable (given also the images' modest dimension) from the original ones. We can also see that if any of the ”down-path” macro-blocks is damaged, the shape and pose of the object are affected, while if the ”up-path” macro-blocks are damaged, not only the shape but also the texture is deeply affected. This is especially evident when U0 is damaged. Finally, we can highlight the importance of avoiding heavy perturbations on the input, since any pruning in the input convolution layer and dense timestep embedding layer leads the model to produce only noise. This also further validates the *modus operandi* of our pruning procedure, where the pruning masks are propagated starting from the input layer in order to consistently keep important parts of the input signal.

Another analysis was also performed in order to see if there was a pattern in the way the weights evolved during distillation. It consisted of a quick test computing the L1 Norm on the difference of corresponding weights on the

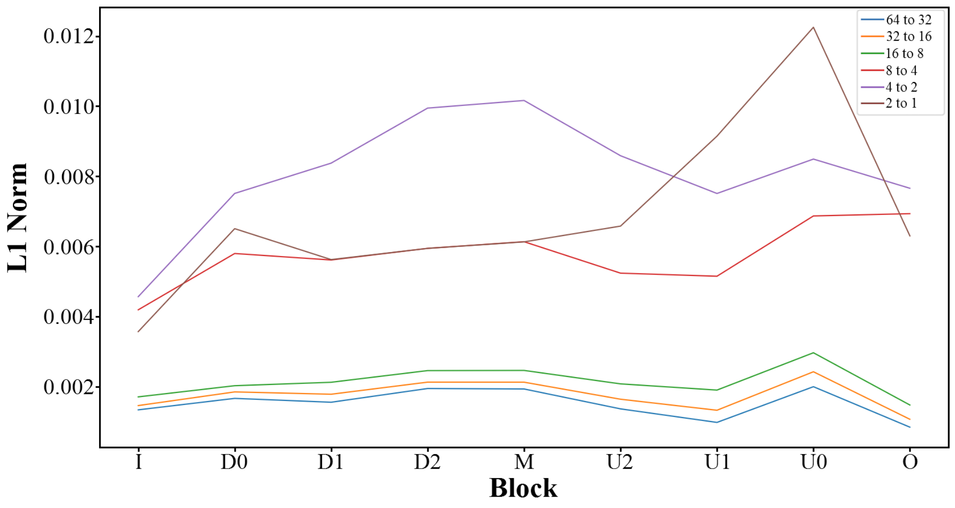


Figure 4.3: *Weight variation analysis plot.*

unpruned models at the end of two successive distillation iterations. This distance is normalized on the number of weights of the layer, grouped by macro-block, and then normalized again on the number of layers within the block. We report the relative plot in Figure 4.3. As we can see, we cannot distinguish clear patterns from the graph, but we notice that in the last three distillation steps, the model tends to change more and more unpredictably. This may be due to the fact that the model needs to increasingly compress information in the available structure.

Considering these preliminary results we decided to set the number of pruning steps per iteration to 4, which are equispaced in time (one step after 10'000 epochs), and the pruning ratio for each macro-block is set to obtain a final 1-step model with a specific compression ratio w.r.t. of the number of the original weights that allows the removal of a number of neurons that is consistent and coherent at each pruning step. Also, we set the ratios to start more aggressively in the first half of the distillation process and later reduced in the second section. The pruning ratios per macro-block, together with the expected final pruning ratio and the actual one are listed in Table 4.1. The last two are different as the unit of the pruning function is the layer, so depending on their number and dimension in a block, the number of removed

	I	**D0**	**D1**	**D2**	**M1**	**M2**	**U2**	**U1**	**U0**	**O**
32/16/8 steps models	4.6875%	4.6875%	9.375%	15.625%	15.625%	15.625%	9.375%	9.375%	4.6875%	4.6875%
4/2/1 steps models	3.125%	3.125%	7.8125%	9.375%	9.375%	9.375%	7.8125%	7.8125%	3.125%	3.125%
Final expected pruning ratio	23.4375% (25%)	23.4375% (25%)	51.5625% (50%)	75%	75%	75%	51.5625% (50%)	51.5625% (50%)	23.4375% (25%)	23.4375% (25%)
Final actual pruning ratio	37.69%	41.36%	72.74%	90.28%	90.28%	89.76%	73.87%	73.56%	41.50%	23.42%

Table 4.1: *Pruning ratio for each macro-block and distillation iteration, with expected final pruning ratio and actual one (1-step model).*

neurons may be higher than expected. Even if this difference is quite large, we decided to keep the ones reported in Table 4.1 as the overall compression achieved by the networks, which is later presented in Section 4.4, was deemed to be appropriate and not exceedingly aggressive.

4.2 Pruning scoring function

Most of the initial experiments were run considering a simple Magnitude Pruning approach, employing the L1 Norm as weight saliency scoring function. During the execution of experiments the work by Fang et al. [9] was published, introducing their criterion based on the Taylor-expansion of the loss distruption (to which we refer to as simply ”Taylor-based”). Thus, we decided to exploit and test their results for our task. Other than directly applying their method, we also tried to adapt it to our task at hand by following an intuition. The idea stemmed from the observation that the score highly depends on the gradients generated by the student model and, since it quickly changes during the distillation iteration, they would be less precise or informative. Therefore, we propose to mix the gradient information from the teacher model, which is very similar to the student model at the beginning of the distillation iteration and doesn’t change throughout it. It follows that the mixture is treated dynamically, by giving more importance to the teacher at the beginning of the distillation iteration and transferring it to the student at the end of the step by means of a parameter that we named γ, linearly changing in $[0, 1]$.

The scoring function is the following:

$$I(\theta_i,x)=\sum_k\left|\theta_{ik}\cdot\left(\gamma\cdot\sum_{\frac{LT_t}{LT_{max}}>T}\nabla_{\theta_{ik}}LT_t(\theta,x)+(1-\gamma)\cdot\sum_{\frac{LS_t}{LS_{max}}>T}\nabla_{\theta_{ik}}LS_t(\theta,x)\right)\right| \tag{4.2}$$

where LT is the loss from the teacher model and LS is the loss on the student model. Moreover, the original Taylor-based criterion employs the loss computed on the ϵ-prediction of the model, while the model was trained using a loss on the **x**-prediction with a reweighting function ("Truncated-SNR") to prevent mathematical problems arising when evaluating the loss at low SNRs. So, we also adapt the objective functions LS and LT to follow the training one. Some experiments were carried out, and their results are reported in Table 4.2. It's important to notice that the pruning is performed with a slightly different technique from the final one, attention blocks are kept unpruned and the models are sparse in this exploratory stage of the experimentation.

	32-steps	16-steps	8-steps	4-steps	2-steps
Unpruned model	2.69	2.73	2.83	3.23	4.86
L1 Norm	3.27	4.38	5.98	-	-
Taylor-based	3.16	**3.61**	**4.80**	**6.87**	-
Taylor (adapted)	**2.94**	3.73	5.06	7.20	12.69
Removed params	∼15%	∼28%	∼40%	∼45%	∼55%

Table 4.2: *FID comparison for different pruning score function.* (Lower is better)

As expected, the Taylor-based criterion outperforms the L1 Norm one. Unfortunately, it outperforms also our adaptation which reports slightly worse results, suggesting that our intuition was not entirely correct and that probably the student and teacher model diverge faster than what we expected within the distillation iteration. Also, we can notice that the adaptation actually reports comparable results in the first two steps and later its performance tends to drop.

This follows from one of the observations resulting from the sensitivity analysis, which showed that the teacher and student model tends to change more and more unpredictably as the distillation process continues (Fig. 4.3). Therefore, the gradients obtained from the teacher are increasingly less informative for the student architecture. Moreover, we used the same threshold $T = 0.05$ to filter the noisy timesteps (suggested by the reference paper) and due to the difference in magnitudes and variation between the ϵ-loss and the weighted $\mathbf{x}$-loss, the timesteps weren't filtered in our adaptation. However, we didn't explore this hyperparameter since the "plain" Taylor criterion performed better, even when compared with intermediate forms of our adaptation (different γ ranges, loss on ϵ...) that are not reported in the table but can be found in Appendix A. Together with these adaptations, we also report some tests with more training epochs which improve the FID score on the network but partially sacrifice the higher training efficiency brought by pruning. In conclusion, the adaptation was discarded from the final method. Finally, we can notice, even if these results are not the definitive ones, the trend of the quality dropping in a quicker way when compared with unpruned distillation results due to the effects of pruning. Further discussion of these aspects will be presented in the following chapters.

4.3 Flexible Group Normalization effectiveness

The last preliminary results are the tests on the Flexible Group Normalization. As explained, this repurposing of the Group Normalization Layer [40] was required to effectively compress the model and keep the consistency between groups of channels. We tested two variants: the first performs normalization considering the removed channel as effectively present but "zeroed-out", while the second one doesn't. This translates into the fact that the former considers the pre-pruning group dimension (8) in the normalization constant, while the second one considers the actual post-pruning one. The results of these ablations are presented in Table 4.3. In these runs, the attention blocks were pruned together with the rest of the network and the chosen scoring criterion was the ”plain” Taylor-based one.

	32-steps	16-steps	8-steps	4-steps	2-steps	1-step
Unpruned Model	2.69	2.73	2.83	3.23	4.86	9.43
Sparse Model	3.36	4.38	6.10	8.98	-	-
”Zeroed” FGN	3.36	4.59	6.56	9.30	15.33	**24.57**
”Proper” FGN	**3.06**	**3.60**	**4.90**	**7.31**	**14.31**	24.79
Removed params	∼16%	∼31%	∼43%	∼52%	∼60%	∼67%

Table 4.3: *FID comparison for Flexible Group Normalization variants.* (Lower is better)

The zeroed-channel variant reports results that are comparable with the uncompressed model. This is expected as they theoretically work in the same way. The variant performing proper normalization returns non-negligible improvements instead. This may be interpreted considering that even if ”unimportant” channels do not heavily influence the pre-computed statistics when removed, they still cannot be considered present as they do not allow a correct normalization of the remaining ones, generating worse results.

4.4 Final method results

Finally, we present the results of our definitive method. Differently from the results produced in the previous paragraphs, a last tweak was introduced in the pruning method, that is the pruning mask propagation in the ResNet block (penultimate bullet point in Section 3.2). In fact, among the last experiments, we noticed that propagating the masks following the path 2Dconv 1 $\rightarrow$ norm 2 $\rightarrow$ 2Dconv 2 yielded slightly worse performances than generating the masks from the second norm layer and then propagating them to the adjacent convolutional layers. Moreover, we also tested a longer training schedule, using

	32-steps	16-steps	8-steps	4-steps	2-steps	1-step
Pruned (50k)	2.97	3.65	5.25	7.31	13.09	22.40
Pruned (100k)	3.15	3.78	5.02	6.94	11.30	19.49
Unpruned	2.69	2.73	2.83	3.23	4.86	9.43

Table 4.4: *FID comparison between pruned and unpruned models* (Lower is better)

100'000 epochs per distillation iteration. This number was used for 2 and 1 step models in [32], while we applied it to all the models and performed pruning every 20'000 steps. The FID results, compared with the scores of the unpruned model, are in Table 4.4. We also report the comparisons in terms of parameters and FLOPs in Table 4.5. We point out that the FLOPs requirement was computed directly with the tools available within the `flax` library.

As we can point out in the previous preliminary results, the removal of weights yields a quick drop in the quality of samples when compared with the same drop yielded from the distillation process. In fact, the unpruned model maintains almost the same quality in the first three distillation steps and then gets worse in the last models. This entails that intensive pruning does not reach its full potential as a compression method as it does for classification models, where the same technique may even improve performances. Also, the longer training schedule does not remarkably help the recovery of the models, except for the 2-steps and the 1-step model, where we measure a gain of 2/3 FID points. This pattern confirms the choice made in [32] to train these two last models for longer. However this improvement, even if positive, may still appear poor if one considers that it is obtained by doubling the training resources consumption. We thus consider the results obtained with the 50k training epochs as our reference ones.

Differently, the drop in parameter numbers and FLOPs is consistent, even if the two are not directly related. In fact, the implementation of the Flexible Group Normalization, with an equal number of groups and channels, requires more operations when compared with the vanilla Group Normalization, as it

	32-steps	16-steps	8-steps	4-steps	2-steps	1-step
			#Weights			
Pruned	50M	42M	34M	29M	24M	20M
Unpruned	60M					
			FLOPs single pass/full diffusion			
Pruned	110M/3.52G	101M/1.62G	92M/734M	85M/341M	79M/158M	73M
Unpruned	120M/3.84G	120M/1.92G	120M/960M	120M/480M	120M/240M	120M

Table 4.5: *Weights and FLOPs comparison between pruned and unpruned models* (Lower is better)

is not optimized. For the same reason, coupled with the presence of some functions within the network that remove or add zero channels whenever the pruning ratio changes between different macro-blocks, the wall-clock time for generation is generally a bunch of seconds longer for compressed models (this is not noticed during training time, as usually the model becomes faster to train with the increase in pruning). This, of course, leaves some necessary room for optimization in order to obtain practical speed-ups, which could be done by finding an optimal form for the FGN that does not exploit matrix multiplication or add ”transition layers” (1x1 convolution) between macro-blocks to automatically adapt the channel dimensions (similarly to the ones used in [39]). However, the latter would require learning a model from scratch with the given layers and cutting them accordingly later, as some experiments were performed to add these layers to the pre-trained models, but yielded very high FID results which are not worth reporting. Despite all of this, we can observe how the gain in terms of flops is still equivalent to the removal of 3 to 1 sample steps of the unpruned counterpart, while contemporarily having a network architecture with a smaller memory footprint.

Considering the full collection of distilled models, the 8 and 4-steps ones propose the best tradeoff between original quality preservation and compression rate. Both have acceptable FID scores and a gain in terms of FLOPs equal to the removal of two and one unpruned sampling steps each, rendering them roughly equivalent to a 6-steps and 3-steps unpruned model. Moreover, the

8-step model is the most comparable one in terms of pruning ratio and FID score to the results in [9]. In this work, they employ a smaller U-Net (half of the weights of the one we and [32] use) and reach an FID of 5.29 with a pruning ratio of around 45%. Their model is still 15M parameters smaller but requires more than twice the FLOPs employed by our 8-step model since their sampling process requires 100 DDIM steps. Moreover, our reference results have been produced with half of the fine-tuning steps required for their network (100k). This underlines the potential behind the mixture of pruning and distillation that we are proposing.

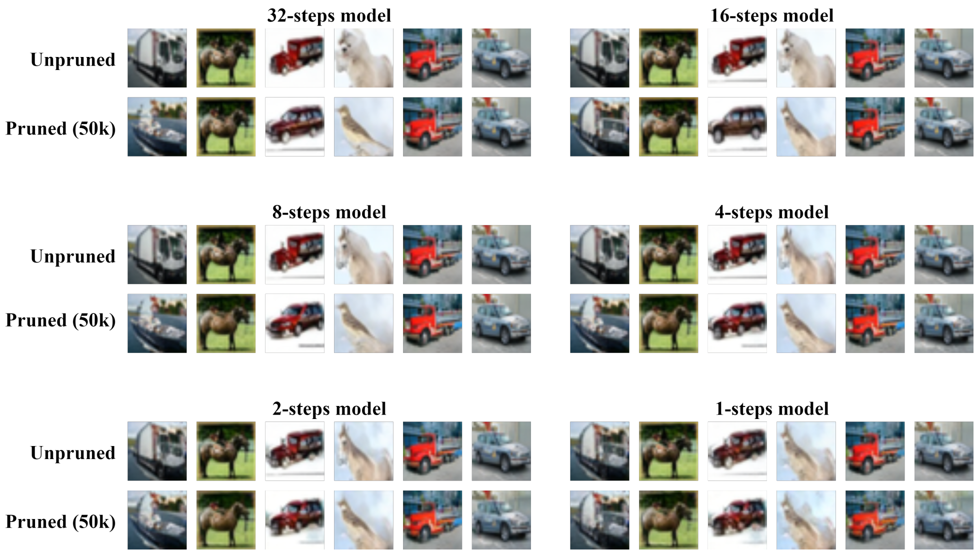


Figure 4.4: *Selected samples from a batch of 16 generated images*. Each batch was generated with the same noise seed (0).

Finally, in Figure 4.4 we report some selected samples from both the pruned and unpruned models. The proposed generated images give a visual validation for the FID results in the previous table. All the unpruned models produce same-quality samples up until the 2 and 1-step models, while the pruned models show samples with degraded quality already around the 4-steps model, confirming our previous statement. Also, as observed, it's interesting to notice that pruning does not completely twist the diffusion trajectory, but that the general content, shape, texture, and color of the image are preserved. There is at most some slight change in subject, where, for example, in the first image the truck is converted to a boat, and in the third image, the bus turns into a car. A similar phenomenon can be observed, even if it's less obvious, also on the model where only the distillation is applied. As a matter of fact, if one focuses on the fourth sample, it initially represents something vaguely remembering a long-haired dog and then slowly turns into a white horse. The network seems, thus, to retain some kind of semantic field information even if it is a class-unconditional generative model, since the change in subject is usually around the same concept (vehicle, animal...). For the sake of completeness, we present other randomly generated samples in Appendix B.

Chapter 5

Conclusions

In this work, we focused on an efficiency problem of the state-of-the-art models in image generation. Lately, they are represented by Diffusion Models [5], which allow the generation of images employing a neural network (generally a U-Net) that models a Markovian process iteratively ”de-noising” an image (initially represented by pure noise) and returning a ”clean” counterpart. Their related sampling method requires multiple network passes and is, therefore, highly resource-intensive. Current literature focuses either on improving the efficiency of samplers or on reducing the number of sampling steps required for a high-quality image. In the latter category we find the work by Salimans & Ho [32], where they introduce a method to progressively distill a DM into a student that requires half the number of sampling steps. The student is, however, forced to preserve the same network architecture. This prevents the possibility of yielding a network with a smaller memory footprint which could, in turn, return a further improvement in resource usage efficiency. We decided to refine this method and tried to bypass this limitation by mixing progressive distillation with a progressive structured pruning method. The procedure was conceived by incorporating different concepts found in the topic literature as: keeping the activation maps channels consistent and cut the network ”greedily” (progressively following the signal path) [22], giving more importance to weight saliency scores and pruning masks generated from the Normalization layers [24][18], apply pruning during training [39], differentiate the pruning ratio depending on the level where the layer is placed in the U-Net [19]. Further refinements are backed up by experiments. In particular, we performed some preliminary tests to choose the additional hyperparameters imposed by the method (pruning ratio values and their variation per block and distillation step) by performing macro-block sensitivity analysis (following [19]), weight variation analysis during distillation and tests on different pruning scoring function. Finally, we tested our variation of the Group Normalization Layer, named *Flexible Group Normalization*. Said variation allows the normalization between groups with an uneven number of channels and was required due to the way channels are removed by our pruning method. To execute these experiments we employed the class-unconditional Net architectures and checkpoints (from 64 to 1 steps) provided by [32] and we used CIFAR-10 [20] as training dataset. The results, entail that:

- We pruned each macro-block (listed in Figure 3.1) in isolation with a fixed pruning ratio and computed the FID without fine-tuning. We found results similar to the ones in [19], where the innermost blocks of the U-Net are more insensible to pruning. From this, we choose three final pruning ratios we want to achieve (25%, 50%, 75%) per macro-block depending on their sensitivity, and distribute them in the 6 distillation steps, further divided into 4 pruning sub-steps. We also noticed that the ”down-path” blocks are mostly focused on rendering the object’s shape and pose, while the ”up-path” blocks are involved in texture rendering. Finally, as similarly shown in [9], we observe that, given a fixed seed, pruning does not completely corrupt or twist the generated sample, which preserves the general content and color.
- By computing the L1 Norm of the difference of the weights of two consequent distilled models, normalized by layer and macro-block dimensions, we show that the last three distillation steps are the most critical

ones when it comes to information compression. Because of this, we choose to reduce the pruning ratio, and consequently the ”stress” posed on the model, in the last iterations of the distillation.

- We compared the commonly used L1 Norm with a scoring criterion based on the Taylor expansion of the loss distruption, proposed in [9] and an adapted variant as pruning score functions. The variant we propose tries to dynamically mix the gradient information from the student with the same information taken from the teacher, based on the intuition that it could be more reliable in the first phases of the distillation steps. The results show that the plain criterion returns better quality samples, especially on lower step models, so we adopt it ”as is” for our method.
- The idea behind the FGN is to avoid the network to re-train the weights and re-compute the normalization statistics whenever a channel is removed from a group (”Group shifting problem”). We tested two variations: the first one implements the normalization by considering the removed channels as present but ”zeroed-out” (as it happens in a sparse model), and the second performs ”proper” normalization by considering the post-pruning group cardinality. We see that the latter represents the better alternative as, even if the removed channels are deemed ”unimportant”, their magnitude is non-negligible when applying normalization.

After these preliminary results, we adopt the techniques returning the best results (differentiated pruning based on sensibility and information compression, Taylor-expansion-based scoring function, ”proper” FGN) and perform some last tweaks to obtain our final models. We reported the results and the comparisons with the unpruned models in terms of quality (FID score) and gain in efficiency (number of weights and FLOPs), from which we observe that:

- The quality tends to drop faster when adding pruning instead of considering only the distilled model, reaching a 1-step model with more than twice the FID of the uncompressed model, even with a comparable training schedule. This indicates, as noticed in [9], that intensive pruning does reach its full potential as a compression method for DMs, where the weight removal brings a fast degradation of the quality of the samples, differently from simpler networks and tasks (as classification) where pruning may even improve the performances of a model [2]. Also, only the most compressed models (2 and 1 step) benefit from longer training schedules to an extent that may not justify the higher resource consumption.

- The achieved compression ratios are remarkable, and even with the implementation being not optimized due to the presence of time-consuming operations of channel removal or zero-channels addition when there is a change in pruning ratio or the matrix multiplication employed in FGN, each model has a gain in terms of FLOPs equal to 3 to 1 sampling steps of the unpruned model. This, depending on the model employed, may represent a fourth (8-step, 4-step) or even half of the required FLOPs for the full unpruned sampling.

Following these observations, the 8 and 4 steps models (50k epochs) are the ones presenting the best tradeoff between compression ($\sim$56% and $\sim$48% of the original dimension respectively) and sample quality (FID 5.25 and 7.31 respectively) when compared with the models in [9] or [26] and surpass them by either considering the number of FLOPs required for the full diffusion process or sample quality.

In conclusion, we can state that the presented method reaches its proposed task of obtaining a lighter Diffusion Model for image generation, which requires fewer training resources given the smaller architecture, by sacrificing some sample quality. It also offers, thanks to the *Progressive Distillation* method used as its core, the possibility to choose which model best fits the available resources and/or the quality requirements. We think that most of the importance in this work resides in its explorative value, as it is, to the best of our knowledge, one of the few works studying the possibility of time and space compression for neural networks and especially for Diffusion Models. It opens, thus, multiple possibilities for future works and extensions. One may focus on optimizing the proposed FGN layer or the network architecture (*e.g.* by adding and training from scratch transition layers that guarantee channel consistency) to obtain wall-clock time improvements during inference. Additional tests of the proposed method may be performed on conditional models and datasets providing larger images as ImageNet [6] or LSUN [41], which are easier to inspect and might lead to further information and conclusions. Finally, further pruning methods that are not covered in this work may be tested, like the ones ”by design” similar to those proposed in [39] or [8], to close or shrink the quality gap between the unpruned models and the pruned ones.

Appendix A

Further experiments on gamma and longer training

Before abandoning the idea of adapting the Taylor-expansion-based criterion to our method, we tested some different configurations of the value of γ. We didn't include these results in the dedicated Chapter as they were highly experimental and difficult to compare with the other results, considering that the models weren't compressed, the attention blocks were not pruned, the threshold $\mathcal{T}$ for the criterion wasn't properly set and contextually, due to some time constraints, we performed these experiments while also testing longer training schedules. However, we believe that it is still useful to report these results.

After testing the adaptation reported in Paragraph 4.2, we decided to try fixing the value of γ to a value in $\{0.8, 0.5, 0.2\}$ (which represents a ”softer” mixing than the one presented previously) throughout the distillation process and using the training loss function in evaluating the pruning saliency score. The results are reported in Table A.1. As we can see from the table, the relative comparison between the method adaptations shows that the value of gamma should decrease during distillation, as the teacher gradients become less and less informative for the student structure, confirming our intuition and the observation done in Section 4.1. If we focus on the FID values, we observe a ”curve” trend the gamma value should follow. We find that a close analytical

	32-steps	16-steps	8-steps	4-steps	2-steps
$\gamma = 0.8$	**2.90**	3.78	5.55	7.98	-
$\gamma = 0.5$	2.94	**3.71**	5.10	7.83	13.52
$\gamma = 0.2$	3.11	3.75	**4.86**	**7.08**	**-**
Taylor (adapted)	2.94	3.73	5.06	7.20	12.69

Table A.1: *FID scores of further experiments fixing the value of γ.* (Lower is better)

curve is the one described by the exponential decay:

$$\gamma = 0.8e^{-k \cdot step} \tag{A.1}$$

where $step$ is the distillation iteration index in $[0, 5]$ and k is set to 0.75 to mimic our desired curve. We performed this experiment both with the loss reweighting and computation as done during training (thus, without timestep truncation) and with the original loss on the ϵ-prediction (threshold for timestep truncation set to $\mathcal{T} = 0.05$). This time, each distillation iteration comprises 100'000 training epochs rather than 50'000 (each pruning step is performed once every 20'000 epochs instead of 10'000). The results are in table A.2.

	32-steps	16-steps	8-steps	4-steps	2-steps	1-step
Taylor-based (50k)	3.16	3.61	4.80	6.87	-	-
Taylor-based (100k)	**3.00**	**3.32**	**4.33**	**5.98**	-	-
Gamma exp. decay (100k)	3.06	3.56	4.70	6.70	-	-
Gamma exp. decay w/ trunc. loss (100k)	3.05	3.55	4.58	7.09	11.05	18.44
Removed params	$\sim$15%	$\sim$28%	$\sim$40%	$\sim$45%	$\sim$55%	$\sim$62%

Table A.2: *FID results and percentage of removed params of the definitive score function adaptations with longer training time.* (Lower is better)

As we can see, the results between the two adaptations are mostly comparable, showing that the loss reweighting does affect the result of the technique. Moreover, the method taken out-of-the-box from [9] is better than the adaptations, and thus it became our final choice. Its also important to notice that adding training epochs to the distillation process brings noticeable improvements to the quality of samples, but it may be impractical as it nullifies the time-saving benefits of the model compression during training compared with *Progressive Distillation* alone. We, however, also remark that in our final experiments the improvements brought by the longer training schedule was obtained only in the 2 and 1-step models.

Appendix B

Additonal samples

We report some additional samples generated with our final pruned model (50’000 training epochs) and their unpruned counterparts. They were generated by setting the seed to 0 and the batch size to 64.

Figure B.1: *Additional samples from the 32-steps models*.

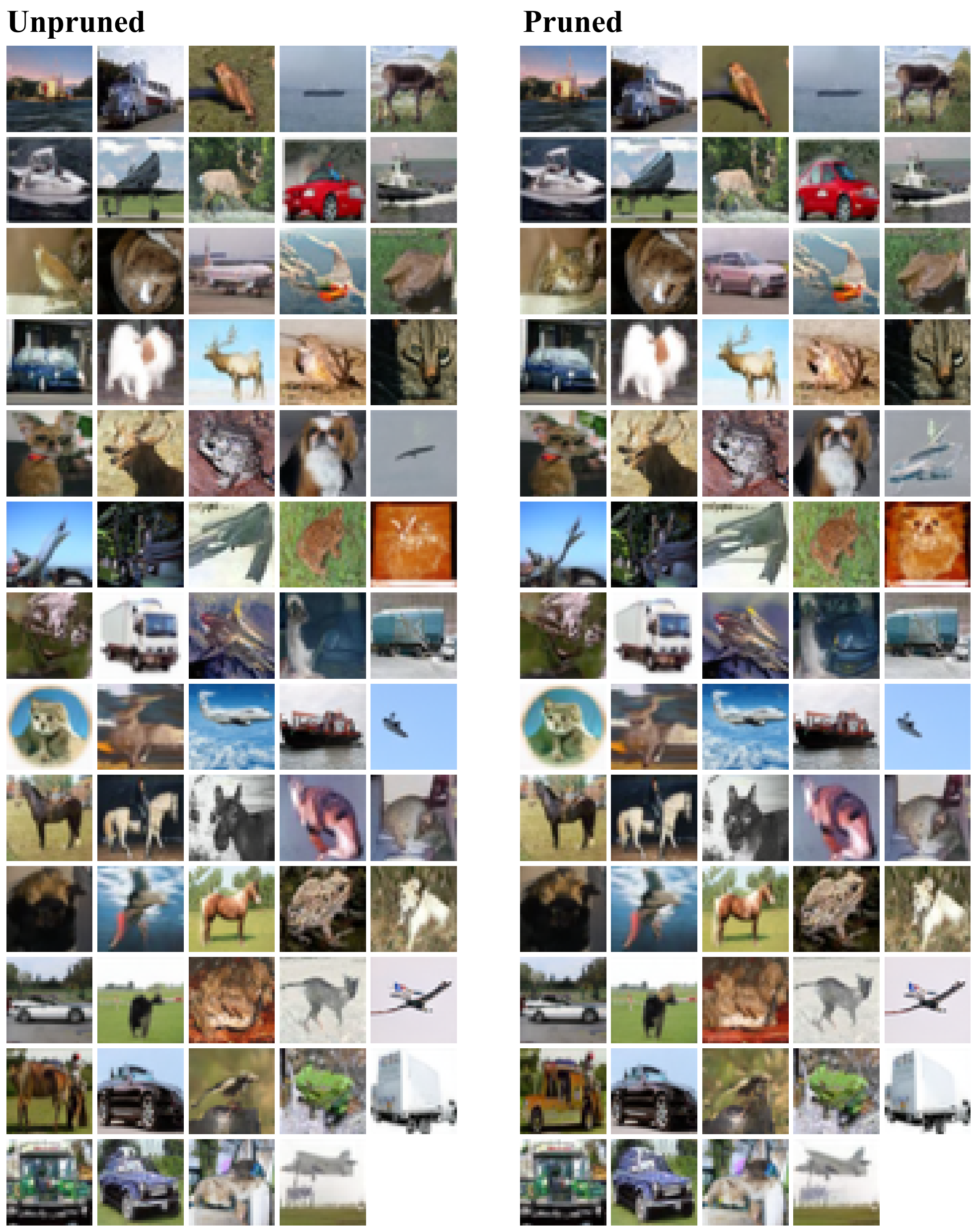


Figure B.2: *Additional samples from the 16-steps models.*

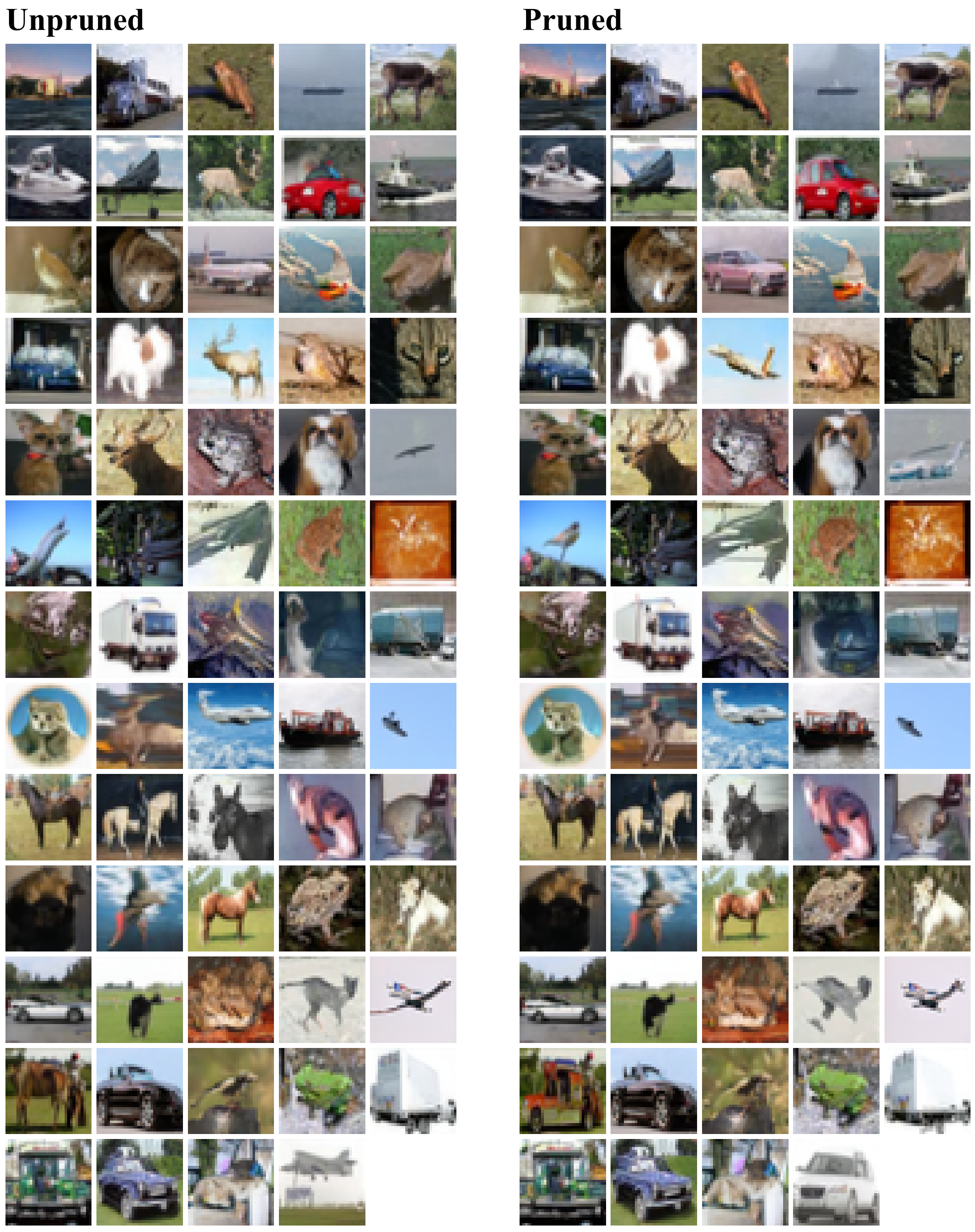


Figure B.3: *Additional samples from the 8-steps models.*

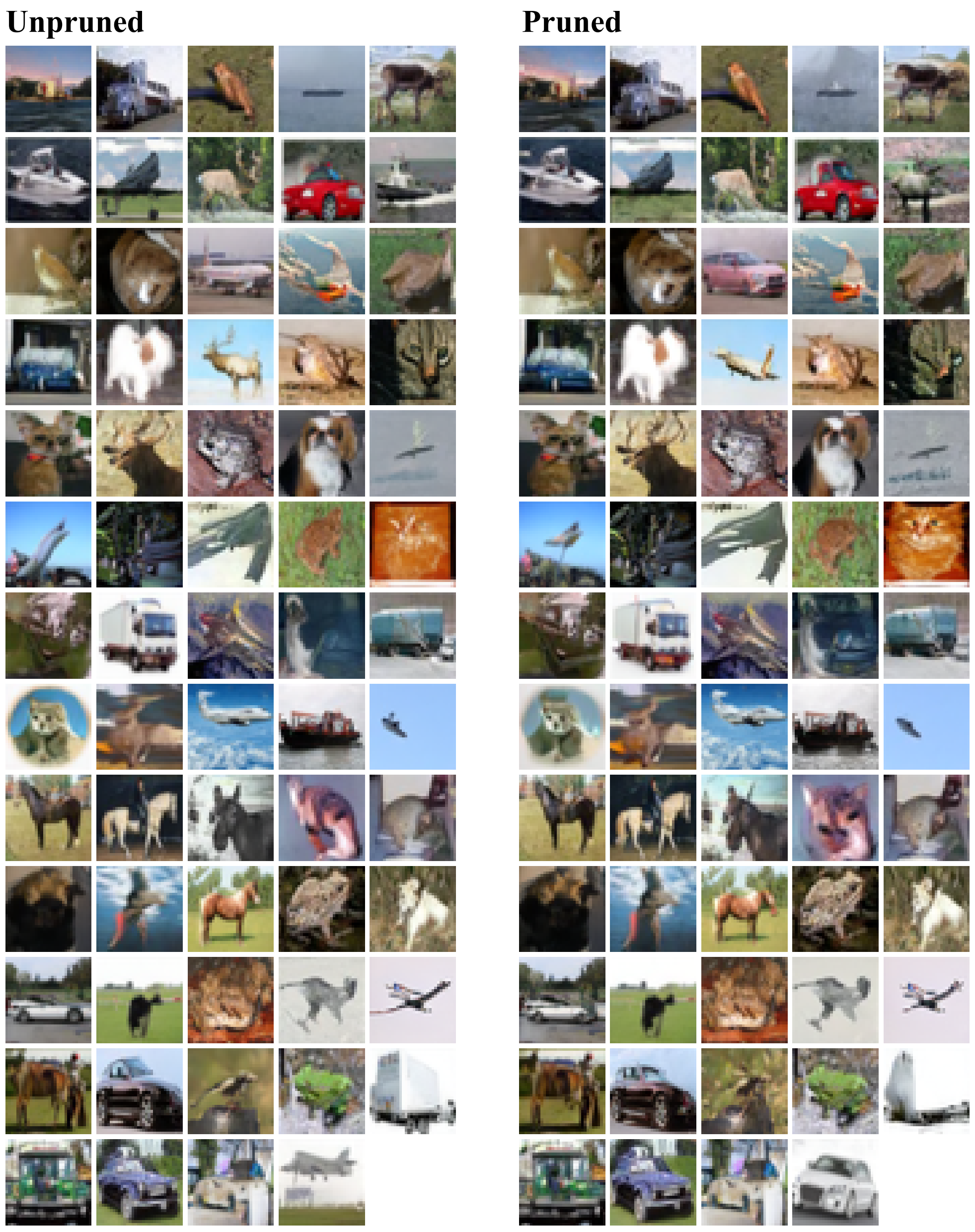


Figure B.4: *Additional samples from the 4-steps models*.

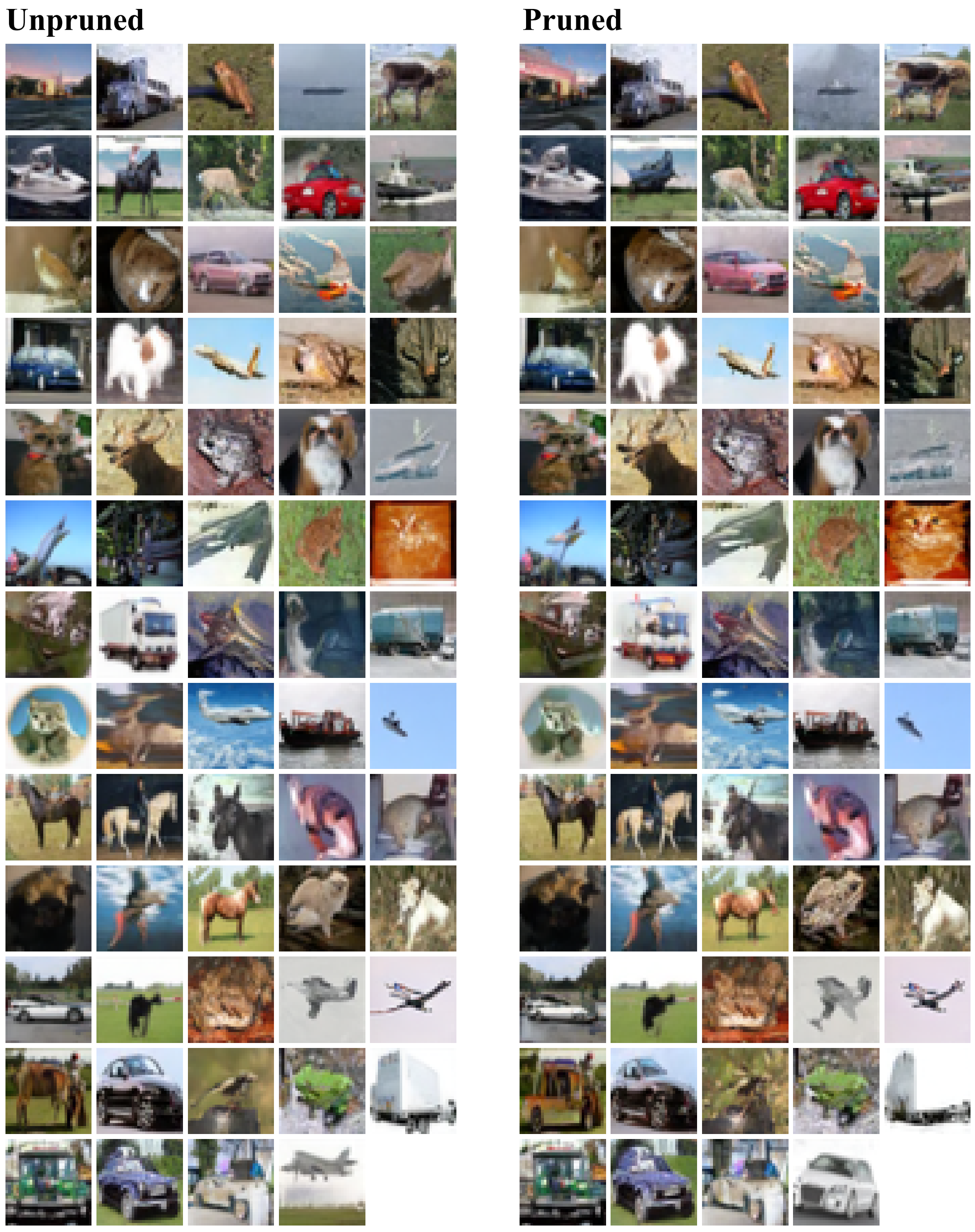


Figure B.5: *Additional samples from the 2-steps models.*

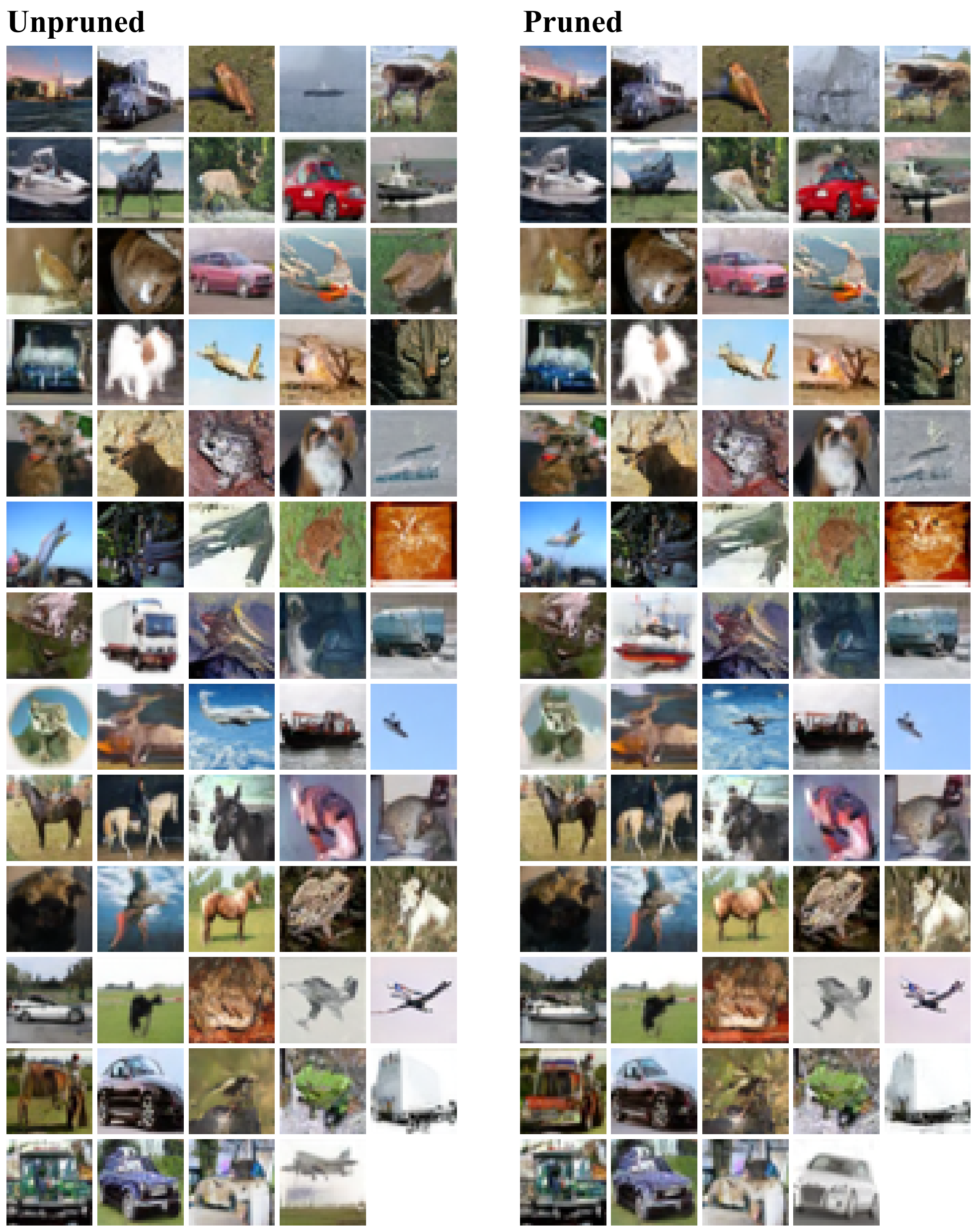
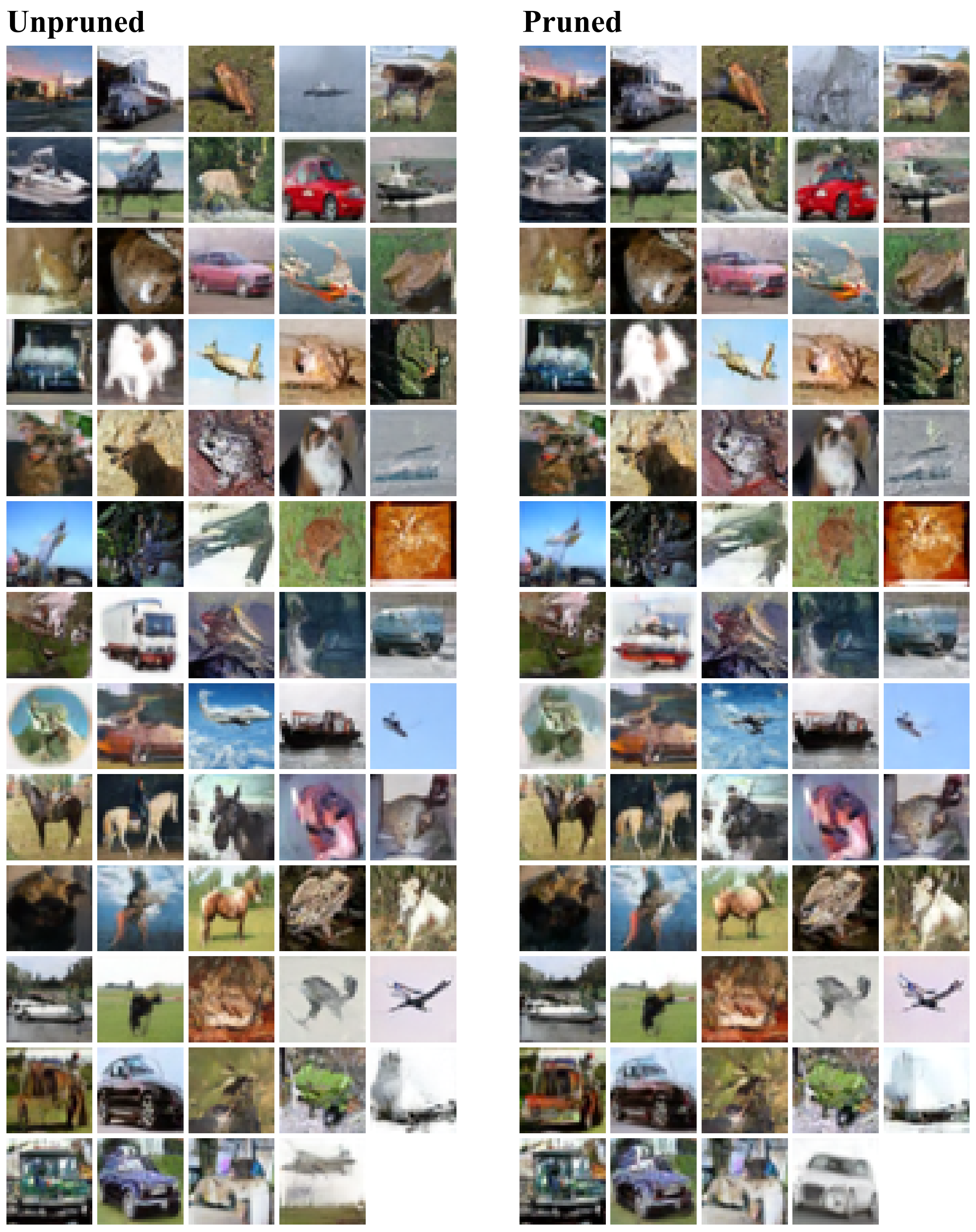


Figure B.6: *Additional samples from the 1-step models*.



Bibliography

[1] N. Aghli and E. Ribeiro. Combining weight pruning and knowledge distillation for cnn compression. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3185–3192, 2021. DOI: `10.1109/CVPRW53098.2021.00356`.

[2] D. W. Blalock, J. J. G. Ortiz, J. Frankle, and J. V. Guttag. What is the state of neural network pruning? *CoRR*, abs/2003.03033, 2020. arXiv: `2003.03033`. URL: `https://arxiv.org/abs/2003.03033`.

[3] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, version 0.4.19, 2018. URL: `http://github.com/google/jax`.

[4] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018. arXiv: `1809.11096`. URL: `http://arxiv.org/abs/1809.11096`.

[5] M. Chen, S. Mei, J. Fan, and M. Wang. An overview of diffusion models: applications, guided generation, statistical rates and optimization, 2024. arXiv: `2404.07771 [cs.LG]`.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: a large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233, 2021. arXiv: `2105.05233`. URL: `https://arxiv.org/abs/2105.05233`.

[8] X. Ding, G. Ding, Y. Guo, and J. Han. Centripetal SGD for pruning very deep convolutional networks with complicated structure. *CoRR*, abs/1904.03837, 2019. arXiv: `1904.03837`. URL: `http://arxiv.org/abs/1904.03837`.

[9] G. Fang, X. Ma, and X. Wang. Structural pruning for diffusion models, 2023. arXiv: `2305.10924 [cs.LG]`.

[10] J. Frankle and M. Carbin. The lottery ticket hypothesis: training pruned neural networks. *CoRR*, abs/1803.03635, 2018. arXiv: `1803.03635`. URL: `http://arxiv.org/abs/1803.03635`.

[11] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. Pruning neural networks at initialization: why are we missing the mark? *CoRR*, abs/2009.08576, 2020. arXiv: `2009.08576`. URL: `https://arxiv.org/abs/2009.08576`.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[13] Y. He and S. Han. ADC: automated deep compression and acceleration with reinforcement learning. *CoRR*, abs/1802.03494, 2018. arXiv: `1802.03494`. URL: `http://arxiv.org/abs/1802.03494`.

[14] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017. arXiv: `1706.08500`. URL: `http://arxiv.org/abs/1706.08500`.

[15] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015. arXiv: `1503.02531 [stat.ML]`.

[16] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models, 2020. arXiv: 2006.11239 [cs.LG].

[17] J. Ho and T. Salimans. Classifier-free diffusion guidance, 2022. arXiv: 2207.12598 [cs.LG].

[18] M. Kang and B. Han. Operation-aware soft channel pruning using differentiable masks, 2020. arXiv: 2007.03938 [cs.LG].

[19] B.-K. Kim, S. Choi, and H. Park. Cut inner layers: a structured pruning strategy for efficient u-net gans, 2022. arXiv: 2206.14658 [cs.LG].

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. In 2009. URL: https://api.semanticscholar.org/CorpusID:18268744.

[21] Y. Lecun, J. Denker, and S. Solla. Optimal brain damage. In volume 2, pages 598–605, January 1989.

[22] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. arXiv: 1608.08710. URL: http://arxiv.org/abs/1608.08710.

[23] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K. Cheng, and J. Sun. Metapruning: meta learning for automatic neural network channel pruning. *CoRR*, abs/1903.10258, 2019. arXiv: 1903.10258. URL: http://arxiv.org/abs/1903.10258.

[24] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. *CoRR*, abs/1708.06519, 2017. arXiv: 1708.06519. URL: http://arxiv.org/abs/1708.06519.

[25] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. Dpm-solver++: fast solver for guided sampling of diffusion probabilistic models, 2023. arXiv: 2211.01095 [cs.LG].

[26] E. Luhman and T. Luhman. Knowledge distillation in iterative generative models for improved sampling speed, 2021. arXiv: 2101.02388 [cs.LG].

[27] C. Meng, R. Rombach, R. Gao, D. P. Kingma, S. Ermon, J. Ho, and T. Salimans. On distillation of guided diffusion models, 2023. arXiv: 2210.03142 [cs.CV].

[28] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: improving latent diffusion models for high-resolution image synthesis, 2023. arXiv: 2307.01952.

[29] A. Renda, J. Frankle, and M. Carbin. Comparing rewinding and fine-tuning in neural network pruning. *CoRR*, abs/2003.02389, 2020. arXiv: 2003.02389. URL: https://arxiv.org/abs/2003.02389.

[30] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. *CoRR* , abs/2112.10752, 2021. arXiv: 2112.10752. URL: https://arxiv.org/abs/2112.10752.

[31] O. Ronneberger, P. Fischer, and T. Brox. U-net: convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[32] T. Salimans and J. Ho. Progressive distillation for fast sampling of diffusion models, 2022. arXiv: 2202.00512 [cs.LG].

[33] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. arXiv: 1503.03585 [cs.LG].

[34] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *CoRR*, abs/2010.02502, 2020. arXiv: 2010.02502. URL: https://arxiv.org/abs/2010.02502.

[35] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *CoRR*, abs/2011.13456, 2020. arXiv: `2011.13456`. URL: `https://arxiv.org/abs/2011.13456`.

[36] H. Thanh-Tung and T. Tran. On catastrophic forgetting and mode collapse in generative adversarial networks, 2020. arXiv: `1807.04015`.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR* , abs/1706.03762, 2017. arXiv: `1706.03762`. URL: `http://arxiv.org/abs/1706.03762`.

[38] P. Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011. DOI: `10.1162/NECO_a_00142`.

[39] D. M. Vo, A. Sugimoto, and H. Nakayama. Ppcd-gan: progressive pruning and class-aware distillation for large-scale conditional gans compression, 2022. arXiv: `2203.08456 [cs.CV]`.

[40] Y. Wu and K. He. Group normalization. *CoRR*, abs/1803.08494, 2018. arXiv: `1803.08494`. URL: `http://arxiv.org/abs/1803.08494`.

[41] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao. Lsun: construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[42] Q. Zhang and Y. Chen. Fast sampling of diffusion models with exponential integrator, 2023. arXiv: `2204.13902 [cs.LG]`.

Acknowledgements

I'd like to thank all the NII staff for the wonderful experience, and especially my Japanese co-supervisor Prof. Akihiro Sugimoto for all the constant support, the ideas, and the guidance on this relatively new and tricky topic. A special thanks to Ph.D. Minh-Duc Vo in providing, other than his experience on the task, some of his computing resources to allow me to finish my experiments for this work. Furthermore, I want to acknowledge my supervisor at UniBo, Prof. Samuele Salti for supporting me and providing ideas after the end of my Internship. Finally, a deep thanks to my family who was always with me even if I was in the opposite part of Italy at first and in the opposite part of the world later.