Master ATIAM :

Internship Report

Compressing audio generative models for embedded device

Supervisors :

Cyran Aouameur

Javier Nistal Hurle

Student :

Jeremy Uzan

2020-2021

Acknowledgments

I would first like to express my gratitude to my main supervisors Cyran Aouameur and Javier Nistal for their constant availability and help.

I would also like to thank Emmanuel, Gaëtan, David, Peter, Theis, Simon, Michael, Pietro, members of the Sony CSL team, for their helpful hints all along this internship.

Internship context

Sony Computer Science Laboratories (Sony CSL), Paris. This Laboratory was established in 1988 with the purpose of contributing to humankind and society by creating new research areas, research paradigms, new technologies and new businesses. Sony CSL is organized with different research teams, including the Music Team composed with 2 PhD students, 8 permanent researchers and a few freelance developers.

Researchers work with Sony-affiliated musicians and content providers to push the boundaries of creativity and understand the complexity of modern music production processes. These collaborations with artists help to improve the tools they develop to best meet musicians' workflow. The team recently participated in the AI Song Contest, in partnership with french artist Whim Therapy. This international competition explores the use of AI in the songwriting process, with teams composed of musicians, researchers, data scientists and developers.

Abstract

These recent years, deep neural networks models have demonstrated their performance in many tasks like source separation, style classification, and music generation. Nevertheless, these models are often very heavy and computationally expensive. Given DrumGAN, a GAN-based model for drum sounds synthesis, we want to train a lighter model using compression methods, notably knowledge distillation. This would allow to cut the inference time, thus unlocking new interaction possibilities for artists working with this tool. Based on the state of the art of generative models in audio and different compression techniques, we explain the challenges posed by this research project. After testing different types of knowledge distillation on a toy classifier model, we implement these techniques on the DrumGAN model. Even if the sounds generated by our compressed model, or student model, are not as rich as the basic model, we still get faster samples generations. Using the Fréchet Audio distance, an evaluation metric for audio generative models, we compare the different distilled models we trained.

Keywords: generative adversarial networks, drum sounds, knowledge distillation, quantization

Ces dernières années, les modèles de réseaux de neurones profonds ont démontré leurs performances dans de nombreuses tâches comme la séparation de sources, la classification de styles, et la génération de musique. Néanmoins, ces modèles sont souvent très lourds et coûteux en calcul. Dans le cas de DrumGAN, un GAN pour la synthèse de sons de batterie, nous souhaitons entraîner un modèle plus léger en utilisant des méthodes de compression, notamment la distillation de connaissances. Cela permettrait de réduire le temps d'inférence, ouvrant ainsi de nouvelles possibilités d'interaction pour les artistes travaillant avec cet outil. Sur la base de l'état de l'art des modèles génératifs en audio et des différentes techniques de compression, nous expliquons les défis posés par ce projet de recherche. Après avoir testé différents types de distillation sur un modèle de classificateur, nous mettons en œuvre ces techniques sur DrumGAN. Si les sons générés par notre modèle compressé, ou modèle étudiant, ne sont pas aussi riches que le modèle de base, nous obtenons tout de même des générations de sons plus rapides. En utilisant la distance audio de Fréchet, une métrique d'évaluation pour les modèles génératifs audio, nous comparons les différents modèles distillés que nous avons entraîné.

Mots-clés: réseaux adversariaux génératifs, sons de batterie, distillation de connaissances, échantillonage

Contents

1 Introduction

1.1 New tools for Music Production

Generative music has become an answer to a simple demand: more music is needed than ever, thanks to a ballooning number of content creators on streaming and social media platforms. In the next decade, more and more music content will be written partially or entirely with machine learning software. The latest development in AI is already fueling its use in popular music, and reinvents the interaction between music artists and their tools [1]. For instance, Fiebrink [2] developed the Wekinator for mapping arbitrary gestures of human beings to parameters of sound synthesis algorithms. Regarding music generation, machine learning-based generative models also open great avenues for the future, as they are able to capture specific aspects of music generation like melody and composition [3], timbre [4, 5] or vocals [6, 7].

Early generative music models were able to generate music in the symbolic domain, that is to say the specific timing of notes, their pitch and velocity [8, 3, 9]. The symbolic approach allows to model the problem in a relatively low-dimensional space, but constrains the generations to music that can be accurately expressed in those terms. Indeed, if modeling Bach chorales in the symbolic domain is relevant, one misses the timbre information which is essential when modeling other music genres such as contemporary popular music. Other promising researches have then tried to generate music directly in raw audio form. Some models were successfully trained to generate music either in the raw audio domain, [10, 11, 5], or in the spectral domain [12, 4]. A wide variety of architectures are being used, ranging from variational auto-encoders [13, 14] to generative adversarial networks [15].

Based on such models, researchers try to develop a new generation of musical tools infused with machine learning such as Google's Magenta Studio [16], or Sony CSL's DrumNet [17], BassNet [18] and NONOTO [19]. However, there exists a huge competition in music tools, as new plugins are released everyday. Therefore, in order to be actually utilized by professionals, new tools must at least meet the industry standards. In its situation, working with Sony-affiliated artists, Sony CSL puts a strong focus on the sample rate of generated sounds, which should at least be 44100 Hz, and the models' ability to operate in real-time, or very close. Another key element is the portability and accessibility of the models. Indeed, machine learning models rely on deep networks with millions of parameters and require strong computation capabilities to train and run them, such as powerful Graphic Processor Units (GPUs). Even if this is becoming more of a standard with time, most computers are not

equipped with such computing power. Researchers then have no choice but to find ways to miniaturize their models, in order to be able to deploy them tomorrow.

1.2 Compressing Neural Networks

Compressing techniques have been proposed to reduce the size of deep models, without losing too much quality. As increasingly large neural networks are considered, reducing their storage and computational cost becomes a must, especially for some real-time applications such as virtual instruments. This challenge also arises when deploying deep learning systems to portable devices with limited resources (e.g. memory, CPU, energy, bandwidth). Smartphones and micro computers like Raspberries and Jetson Nano now embed graphic processors, making them suitable for running powerful models. However, their limited battery and the lack of large cooling system will still force models to become more and more compressed.

Recent works on compressing and accelerating deep neural networks show ways to achieve these goals. Pruning based methods explore the redundancy in the model parameters and try to remove the redundant and uncritical ones. The knowledge distillation based methods aim at training a more compact neural network to reproduce the output of a pre-trained larger network.

1.3 The Problem with DrumGAN

The main goal of this internship is to apply network compression techniques to DrumGAN, a GAN-based model for drum synthesis that was developed at SONY CSL [20]. DrumGAN allows to synthesize high-fidelity drum samples based on perceptual characteristics chosen by the user.

User experience has shown that the DrumGAN plug-in is limited by the time needed to generate a sound. At the time being, this causes problems for Sony CSL developers. To expose them, let us start from the DrumGAN interface, shown on figure 1. This interface features an 2D slider, that allows the user to explore portions of the higher-dimensional latent space. When clicking on a point on this slider, the corresponding sound is generated by the model, and a buffer is filled with it. This interface is explicitly designed for exploration, as the user is expected to click multiple times in order to refine their choice. Therefore, a first problem appears here, as the generation lag occurs multiple times, which can be very frustrating. Indeed, being able to generate sounds instantly would make the interaction

Figure 1: DrumGAN's interface

between the music producer and the machine way smoother.

A solution to cut this refinement time could be to get the 2D slider to display information about what it contains, like in another Sony CSL plugin called Impact Drums (Fig. 2) where audio descriptors are shown. However, this leads us to a second issue : this 2D slider shows a randomly chosen 2D plane, lying in the higher dimensional latent space, and it can (should) be modified during the exploration process. Therefore, it is impossible to pre-compute descriptors. Indeed, to display audio features in the space, we need to sample points from the currently displayed portion of the space, generate audios from them and then analyze their features. We then fallback on the first issue with generation time, which is exacerbated here because of the large amount of audio we need to generate and analyze, in order to have a smooth visualization in the latent space.

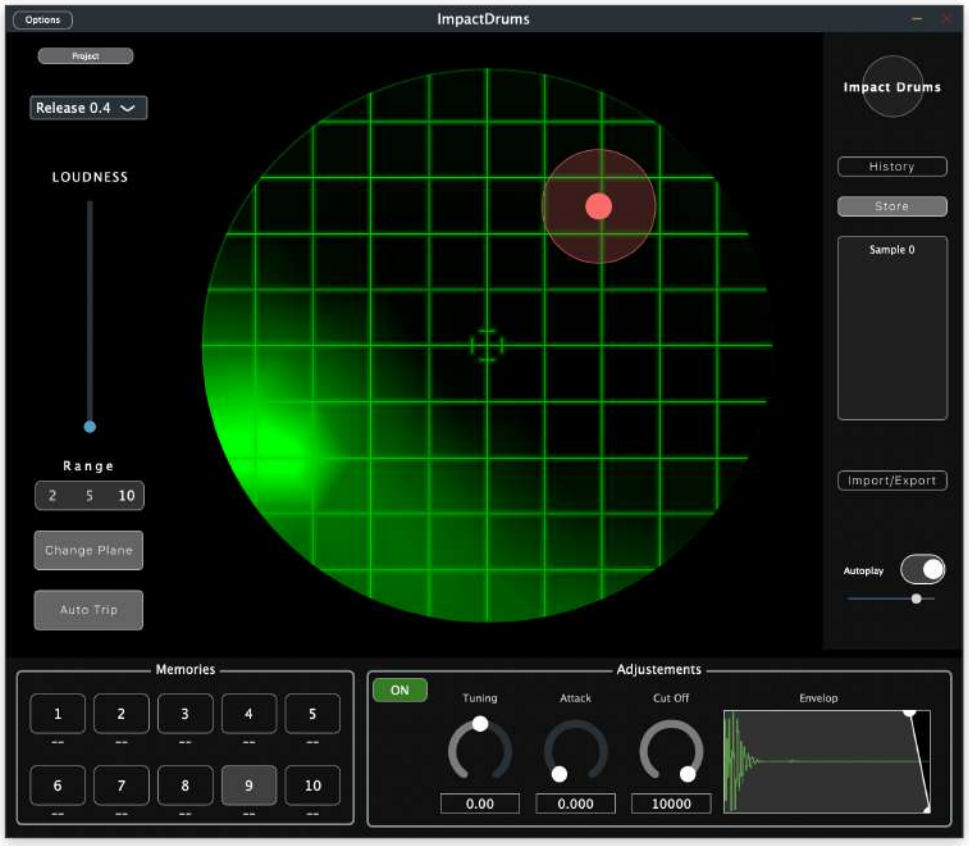


Figure 2: Impact Drums' interface. The 2D slider displays information about the sounds (here the release time).

1.4 Our Proposal

In this project, we want to compress DrumGAN without sacrificing the sound quality of generations.

In the remainder of this document, we provide an introduction to deep learning, starting from the basic concepts of neural networks (Sec. 2.1). Then, we briefly present recent generative models targeted at learning audio data (Sec. 2.1.5). We then introduce Generative Adversarial Networks (GANs) (Sec. 2.2), and some network compression techniques (Sec. 2.4). After this state-of-the-art (Sec. 2), we will present the experiments and results that we have obtained (Sec. 3). This will be followed by a discussion and comparisons of the different experiments, in order to analyze and understand the strengths and weaknesses of our approach. Finally, a brief conclusion will summarize the content of the report, stress our contributions and introduce directions for future work (Sec. 4).

2 State of the art

In this work, we will mostly work with Generative Adversarial Networks (GANs) which rely on Convolutional Neural Networks (CNNs) as their core component. Hence, we here briefly introduce the basics of CNNs and GANs, and the most commonly used tweaks that allow training them efficiently. Then, we will introduce the framework of knowledge distillation, that we heavily rely on in this work, as well as other network compression techniques.

2.1 Basics of Neural Networks

In this section, we are going to give a brief introduction on neural networks. For more detailed explanations, we refer the reader to some more in-depth works [21, 22]

2.1.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a network composed of units, also called neurons or nodes. For the sake of brevity, we are going to consider units that constitute what are called *fully-connected* layers. These units are operators applying a non-linear transformation to an input (Fig. 3).

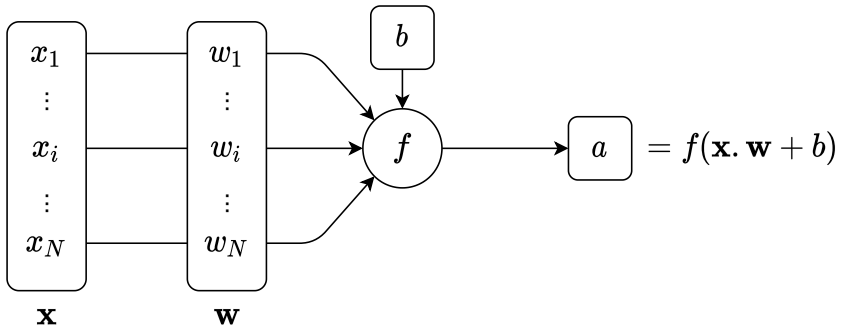


Figure 3: Figure of a neuron in a fully-connected layer. The neuron outputs the activation a, which is a non-linear transformation of an input $\mathbf{x}$

The output a , called the *activation*, is computed as such:

$$a = f(\mathbf{x}.\mathbf{w} + b) \tag{1}$$

$$= f(\sum_{i=1}^{N} x_i.w_i + b) \tag{2}$$

where $\mathbf{x}$ is the input vector of dimension N, $\mathbf{w}$ the *weights* of the unit, b the *bias*, and f a non-linear function called *activation function*.

To build an ANN, we stack *layers* of such units on top of each other. Every unit of a layer is connected to all units of the previous layer, creating a weighted computation graph producing an output y from an input x. If we denote Θ the set of all parameters (weights and biases of all neurons), we can also see the neural network as a parametric function $g_\Theta : X \to Y$, where X and Y are respectively the space of inputs and outputs.

Once an ANN is built, we want to *train* it, in order for it to approximate a function. Typically, in a classifier model, we have data pairs $(x, y) \in X \times Y$, where x is a data point and y its associated label. We want the network g_Θ to learn how to produce an output $\hat{y} = g_\Theta(x)$ that is as close to y as possible. Thus, we can consider an optimization problem for the network:

$$\underset{\Theta}{\operatorname{argmin}}\, L(\hat{y}, y) \tag{3}$$

where $\hat{y}$ is the output of the network and y is the expected output for a specific given input x. L is a loss function that measures the error between $\hat{y}$ and y. Typical loss functions include Mean-Squared Error or Binary Cross Entropy.

The goal is to reach a value of Θ for which the loss $L(\hat{y}, y)$ is as small as possible. We start from a random initialization of Θ and perform successive optimization iterations to change its value into one that reduces the loss. The most basic update rule for Θ is called the gradient descent:

$$\Theta \leftarrow \Theta - \lambda * \frac{\partial L(\hat{y}, y)}{\partial \Theta} \tag{4}$$

where λ is called the learning rate. There exist many more optimization algorithms that perform better than the vanilla gradient descent, such as the Stochastic Gradient Descent [23] or the widely used ADAM algorithm [24].

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep neural networks. Their layers and units are a bit different from the ones presented above, making them particularly suitable to the analysis and generation of visual imagery.

In a convolutional layer, the set of weights and biases is completed with a set of *filters* (or *kernels*) that are learnt through optimization. At computation time, these filters are convoluted across the layer's input. This discrete convolution operation, denoted by $\star$, consists in sliding the filters across the input and computing the dot product of the filter and the input,

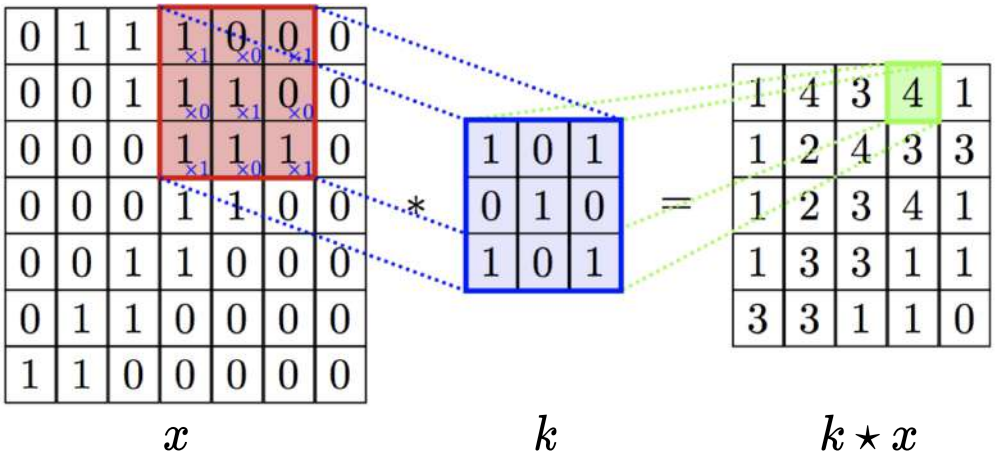


Figure 4: An example of a 2D discrete convolution: the filter k is slid over the input x. For every position, the dot product between the filter and a particular region of the input is computed, producing a feature map.

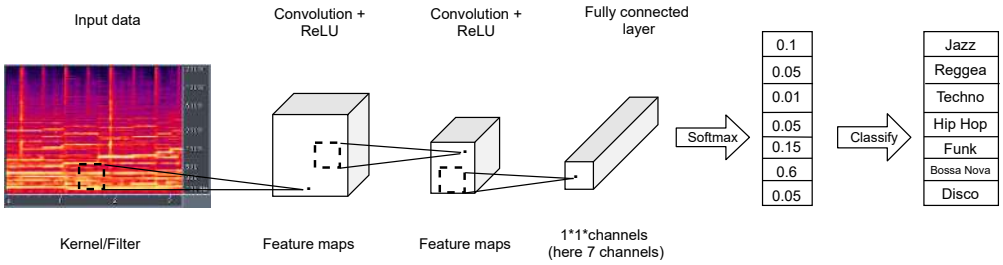


Figure 5: A convolutional neural network used for music genre classification. It is composed of two convolutional layers followed by ReLUs, and one fully connected layer

for each position (Fig. 4) This operation is defined by parameters such as the step-size, most commonly called *stride*, the input padding and others. The result of this discrete convolution operation then goes through an activation function, just as above. Given a convolutional layer l, let w^l be the set of its weights, b^l its biases and $\{k_i^l\}_{i\in[1,N]}$ the set of its N filters. The k_i^l share a fixed dimensionality, called the *filter size*. Given an input x, the activation a^l produced by the layer l is computed as follows:

$$\forall i \in [1, N], a_i^l = f(w_i^l(k_i^l \star x) + b_i^l) \tag{5}$$

Therefore, the activation a^l is made of N channels, also called *feature maps*, an important notion we will use later in this work.

A typical CNN combines convolutional layers to extract high-level features in the data, and fully-connected layers, to recombine them and produce the output the probability vector (Fig. 5).

2.1.3 Supervised and unsupervised learning

Neural networks can be used for a wide variety of tasks. Traditionally in machine learning, we distinguish *discriminative* and *generative* tasks, associated with two learning frameworks, *supervised* and *unsupervised* learning respectively.

Supervised learning is typically used to train algorithms to classify data or to predict outcomes accurately. It can be applied to many discriminative tasks, such as music genre classification (Fig. 5). The main feature of supervised learning is that it requires the training dataset to be labeled. For every input, we need to know the *ground truth*, the output that we wish the network to produce. By choosing the appropriate loss function (Eq. (4)), we can compare the network prediction to the ground truth, and tell how well the network is doing. Then it is a matter of training the network by back-propagating the gradient of this loss.

In case one wants to tackle generative tasks, supervised learning does not do the trick anymore. Indeed, if one wants to generate new data, there is no ground truth as this data should be inherently new. Researchers then rely on unsupervised learning. These algorithms discover hidden patterns and features in data, which is of the greatest interest since explicitly defining the rules of a painting or music is tedious and difficult. We want to talk briefly about generative models, which are unsupervised learning models.

2.1.4 Generative models

Generative models are trained to approximate the distribution of some data. Whether it is image or sound, data distributions are complicated, high-dimensional probability distributions for which there usually exists no formula. The key goal is to learn an approximation of the probability distribution p_{data} over the space $\mathbb{D}$ that supports the data. To do this, we can use a number of samples from p_{data} that we refer as the training data.

The goal is to obtain a generator $g : \mathbb{R}^q \to \mathbb{D}$ that maps samples from a simple distribution p_z supported in a space Z, typically $\mathbb{R}^q$, to points in $\mathbb{D}$ that resemble the given data. We thus have a generator that can map points from a random distribution p_z to a complex distribution p_{data}. Therefore, when trained successfully, we can use generative models to easily create new samples from the simple p_z distribution.

Generative models have now reached an impressive generation quality, with some models being able to create ultra realistic faces [25, 26], or to generate text that is almost indistinguishable from those written by human authors [27]. These models have also been used on music. For instance, in [3], the authors are able to learn how to harmonize melodies in

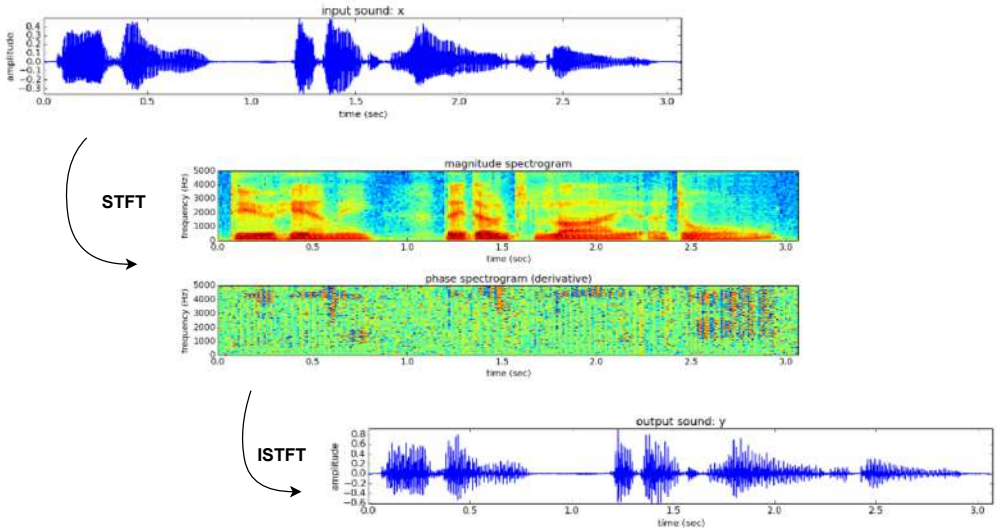


Figure 6: From audio to spectrogram, and back to audio. The x-axis represents time (in seconds) and the y-axis represents the frequency (in Hz).

the style of Bach chorales. However, when considering music as symbolic data, the sound rendering is left to the user. Therefore, a whole branch of research is interested in how to apply these models to directly generate audio content.

2.1.5 Generative modeling for audio

As presented earlier, there exist different techniques that have been implemented to train generative models in music. Some have been trained directly from raw audio, relying on 1D convolutions or recurrent neural networks in order to process the 1-dimensional waveforms [10, 11]. These autoregressive models have been very influential and are still at the forefront in various audio synthesis tasks [11]. However, unlike with GANs, the autoregressive setting results in slow generation as output audio samples must be fed back into the model one at a time. The Differentiable Digital Signal Processing (DDSP) model [5], overcomes this issue by inferring generate control parameters for a synthesizer, rather than generating directly raw audio outputs.

On the other hand, through the use of the Short Time Fourier Transform (STFT), many works have transposed image processing models to audio processing [28]. Fig. 6 shows an example of a spectrogram, which is a time-frequency representation of sound. Audio files can be transformed into spectrograms through the STFT, and back to audio via the inverse STFT. However, the inverse transform requires that we keep the imaginary part of the STFT, which is a complex representation.

Approaches using Variational Auto-Encoders on spectrograms allow to manipulate audio in latent spaces learnt directly from the audio data [14] or by imposing precise perceptive features on the structure of these spaces [13]. However, these models not allow to generate convincing phase information. Hence, authors have to rely on alternative solutions, such as using the Griffin-Lim algorithm to estimate the phase, like in SpecGAN [15]. Some works even rely on the use of an auxiliary neural network which sole purpose is to convert amplitude spectrograms back to the audio domain.

Another more straightforward way to overcome the phase issue is to rely on models that can generate a convincing phase information, such as GANSynth [4]. By replacing the phase information by its derivative, the more stable *instantaneous frequency*, the authors were able to generate clean phase and perfectly invert the STFT. GANSynth even outperforms WaveNet for the task of synthesizing musical notes using labels [4], with a much lesser generation time. In this project, we are particularly interested in such Generative Adversarial Networks.

2.2 Generative Adversarial Networks

2.2.1 Principle of the Vanilla GAN

Generative Adversarial Networks (GANs) is an approach to generative modeling using deep learning methods. It was introduced in 2014 by Goodfellow et al. [29].

GANs are a clever way to train a generative model by presenting the problem as a supervised learning problem with two submodels: the *generator* (G) is trained to generate new examples, and the *discriminator* (D) is trained to distinguish between real (from the dataset) or fake (generated) examples (Fig. 7). The two models are trained together in a zero-sum game, until the discriminator model is fooled about half the time, which means that the generator model generates plausible examples.

In the GAN framework, we define z as the input (latent) *noise*. This noise comes from a predefined distribution called the *prior* distribution, usually chosen as a multi-dimensional Gaussian. We then define our generator G_θ which is a differentiable function represented by an artificial neural network with parameters θ. The generator implicitly defines a probability distribution p_G as the distribution of samples $G(z)$ obtained when $z \sim \ p_z$. We want the generator with initial distribution p_G to learn the distribution p_{data}. There is convergence when p_G is similar to p_{data}. We also define our discriminator as a second neural network D_ϕ with parameters ϕ. Given an input x, D produces a single scalar $D_\phi(x)$ which represents the probability that x comes from the data (hence real) rather than from $G_\theta(z)$ (generated

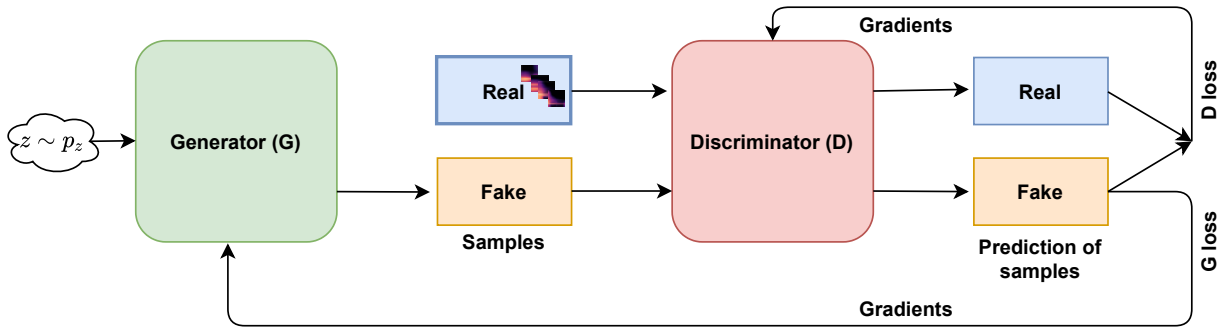


Figure 7: The discriminator needs to distinguish the real and the fake (generated) samples, while the generator needs to fool the discriminator by generating the most realistic samples.

data).

We train D_ϕ to maximize $log(D_\phi(x))$, i.e. the probability of assigning the correct label to both real data (i.e. "1") and generations of G (i.e. "0"). We simultaneously train G_θ to minimize $log(1 - D_\phi(G_\theta(z)))$, i.e. minimize the number of data generation that will be listed as false data (Fig. 7). In other words, D_ϕ and G_θ play the following two player minimax game with the value function V(G,D):

$$\min_\theta \max_\phi V(D,G) = \mathbb{E}_{\mathbf{x}\sim\mathbf{p_{data}}}[log(D_\phi(x))] + \mathbb{E}_{\mathbf{z}\sim\mathbf{p_z}}[log(1 - D_\phi(G_\theta(z)))] \tag{6}$$

2.2.2 Issues observed in GAN training

GAN training is commonly known as unstable. Among the numerous issues researchers face, we describe two of the most common here.

Vanishing gradient In GANs architecture, the discriminator tries to minimize a cross-entropy while the generator tries to maximize it. If the discriminator's confidence is high and starts to reject the samples that are produced by the generator, the generator's gradient vanishes. In effect, an optimal discriminator doesn't provide enough information for the generator to make progress [30].

Mode collapse Usually you want the GAN to produce a wide variety of outputs. You want, for example, a different face for every random input to your face generator. However, if a generator produces an especially plausible output, it may learn to produce only that output. In fact, the generator is always trying to find the one output that seems most

plausible to the discriminator. If the generator starts producing the same output (or a small set of outputs) over and over again, the discriminator's best strategy is to learn to always reject that output. But if the next generation of discriminator gets stuck in a local minimum and doesn't find the best strategy, then it's too easy for the next generator iteration to find the most plausible output for the current discriminator. Each iteration of generator over-optimizes for a particular discriminator, and the discriminator never manages to learn its way out of the trap. As a result the generators rotate through a small set of output types. This form of GAN failure is called mode collapse [31] .

2.2.3 Wasserstein GANs

In an attempt to stabilize GAN training, the more robust Wasserstein GANs (WGANs) were introduced in [32]. Instead of using a discriminator to classify or predict the probability of generated images as being real or fake, the WGAN replaces it with another model called *WGAN* critic, that scores the realness or fakeness of a given image through a metric called the *Wasserstein distance*. As explained in [32] and mentioned in [31], the discriminator in a classic GAN can very quickly learn to distinguish between fake and real, and at the end, provide no reliable gradient information for updating the generator model. As a contrary, the WGAN critic has better properties and will give remarkably clean gradients all along the training. The benefit of the WGAN is that the training process is more stable and less sensitive to the model architecture and choice of hyper-parameter configurations.

2.2.4 Progressive Growing of GANs

Despite the improvement brought by the introduction of the Wasserstein distance, it was still difficult to generate high-fidelity images in high resolution. Therefore, in parallel, a new GAN training procedure called Progressive Growing of GANs (PGGAN or PGAN) was introduced by Nvidia researchers [25]. They succeeded in obtaining ultra-realistic human face of size 1024*1024. The main idea behind the progressive training is to ask the model to not directly produce huge images, but rather start with small images and increase the size progressively (Fig. 8). This allows to reduce the complexity of the task and help the model to learn faster. In high resolutions, the discriminator and the generator are extremely complex functions and the generator can easily stay stuck in a minimum local. Progressively zooming in the precision and the details will actually help the model to reach a global minimum.

In practice, the generator and the discriminator are made of blocks that consist of nearest-neighbor interpolation (Fig. 9) and convolutions. Throughout the training, blocks are pro-

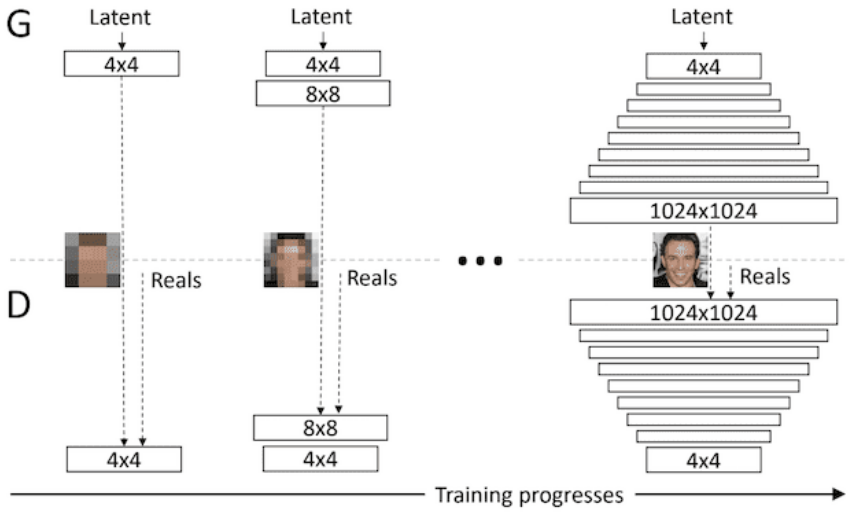


Figure 8: Progressive Growing of GANs. As the training progresses, new blocks are added, which allow to reach higher resolutions. Image from [25]

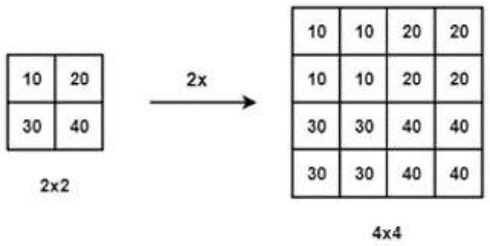
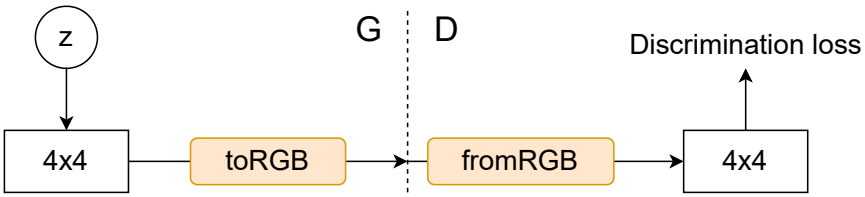
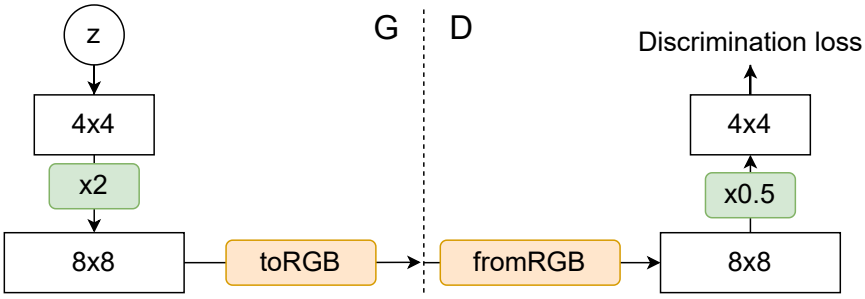


Figure 9: Nearest-neighbor interpolation used in PGAN. Here, we upsample the input by a factor 2, going from resolution 2x2 to 4x4. In the discriminator, we rather downsample the input.

gressively added to both the generator and the discriminator. Anytime during training, we must be able to retrieve results to feed our discriminator and compare with real data. In [25], the authors introduce the *toRGB* layer in order not to impose an information bottleneck in the network's graph. Indeed, if the network had to produce data-sized images in its core, these images could only contain as many channels as there are in the data. Usually, this is a small amount (3 for RGB images, 2 for complex spectrograms), much smaller than the number of channels typically used in convolutional layers. Hence, these *toRGB* layers are added in parallel to the network, and allow to retrieve data-sized images anywhere in the network, without reducing its learning abilities. The reverse thing happens in the discriminator, with the *fromRGB* layers (Fig. 10).



(a) The *toRGB* and *fromRGB* layers allow to retrieve an image-shaped output from the network.



(b) The *toRGB* and *fromRGB* layers are outside of the growing main network, allowing to preserve the amount of convolution channels (feature maps) throughout the main computational graph.

Figure 10: The *toRGB* and *fromRGB* layers.

2.3 DrumGAN

In this project, we aim at compressing a model called DrumGAN [20], a drum synthesizer able to generate drums based on perceptual features such as boominess, hardness or depth. DrumGAN is based on a convolutional Wasserstein GAN [32] and is trained following the progressive growing framework [25] introduced in the previous section.

2.3.1 Model architecture

In the original paper [20], the input to the generator is the concatenation of the latent noise $z \in \mathbb{R}^{128}$, sampled from an independent Gaussian distribution, and a conditioning vector $c \in \mathbb{R}^8$, representing the 8 perceptual features. But the DrumGAN version we have worked with was trained with 3 *soft labels* in place of the features, obtained from a classifier for cymbals, high-hats and kicks. It is still represented as a vector c, but of size 3. As an example, this vector represents the soft label of one random kick sample taken from the dataset $c = [0.003, 9.235e-06, 0.996]$

The resulting vector of size 131 is fed through 6 convolutional and up-sampling blocks to generate the output signal $x = G_\theta(z, c)$. As depicted in figure 11, the generator's input

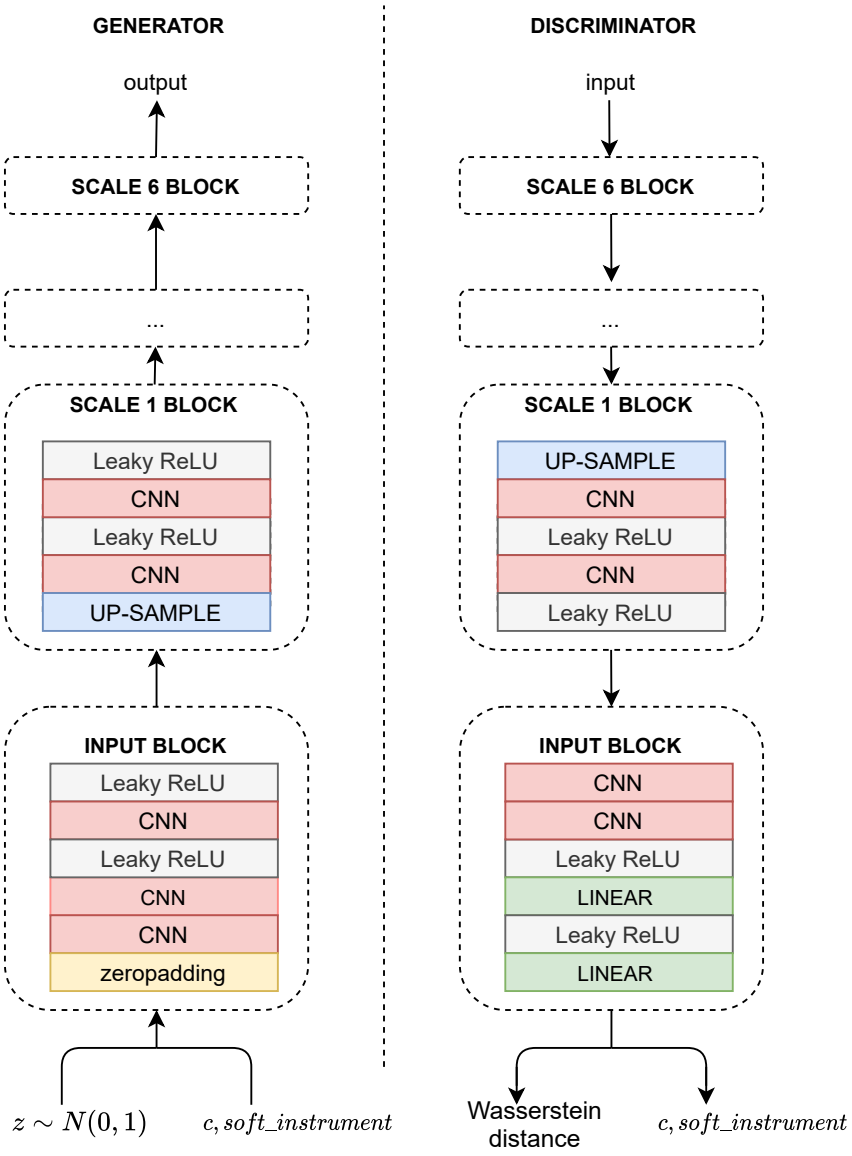


Figure 11: The architecture of DrumGAN. Image from [20]

block performs a zero-padding in the time and frequency dimensions, and is followed by two convolutional layers with ReLU non-linearity, in order to turn the 1D input vector into a 4D convolutional input. Then, each block is composed of one up-sampling step followed by two convolutional layers with filters of size (3, 3) and ReLUs. The number of feature maps decreases from low to high resolution as 256, 128, 128, 128, 64, 32.

In a reverse way, the discriminator is composed of convolutional downsampling blocks. At the end of the network, the discriminator estimates the Wasserstein distance [32] between the real and generated distributions, and predicts values for the 3 perceptual features.

Please note that at generation time, nothing constrains the soft-label vector c to a probability vector that sums to 1. This is the only type of data the generator has seen during training, but experience has showed that the generator can extrapolate to values higher than 1.

2.3.2 Training and Evaluation

DrumGAN was trained on a collection of 300000 drum samples. In the paper version of DrumGAN, the samples were shortened to a duration of one second and down-sampled to a sample rate of 16kHz. But the current version is trained with samples with a sample rate of 44.1 kHz. The model is trained on the real and imaginary components of the Short-Time Fourier Transform (STFT) (see Fig. 6), computed using a window size of 2048 samples and 75 % overlapping.

Following the PGAN training framework [25], the model was trained scale by scale, with a total of 1.1M iterations during 200k iterations for each scale. The optimizer used for the training was Adam [24] with a learning rate $\eta = 0.001$.

To evaluate the performance of the model, the authors use two quantitative metrics. First, the Inception score measures whether the generated images are easy to classify or not. Then, the Fréchet Audio Distance (FAD) [33] compares the statistics of real and generated data. These metrics are explained later in section 3.3.2.

2.4 Compressing Neural Networks

As we have seen, these deep generative models are very heavy, power consuming and take a lot of time to be trained. If we want to use the generative model on a mobile device or perform it in real-time, we need to make it lighter. A series of techniques have been developed for this purpose.

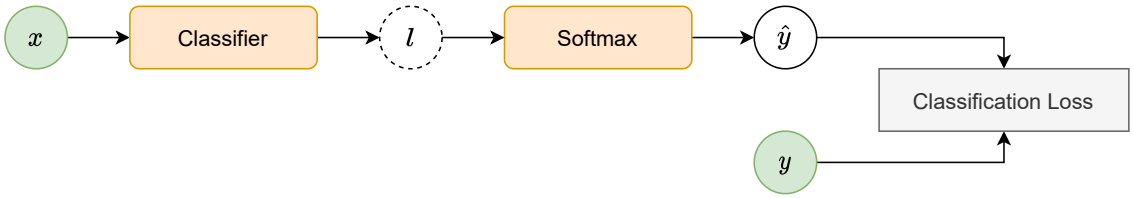


Figure 12: Standard training procedure for classifiers. (x, y) is an input data pair, l is the logits and $\hat{y}$ is the probability vector.

2.4.1 Knowledge Distillation

We will now introduce the basics of Knowledge Distillation (KD), developed in 2015 by Hinton [34]. The idea behind knowledge distillation is to re-use the knowledge acquired by a well-trained neural network called the *teacher*, to transfer it to into another smaller network called the *student*. In some cases, distilled models may be able to generalize better in addition to being significantly smaller [34].

We are now going to consider a simple handwritten digits classification task, with 10 classes (digits 0 to 9). The output of a classification network usually is an unnormalized score for each class (Fig. 12). This vector l, called the network's *logits*, then goes through a softmax layer that transforms it into a normalized probability vector y. The softmax operation is defined as follows:

$$y_i = \frac{e^{l_i/T}}{\sum_i e^{l_j/T}} \tag{7}$$

where T is called the temperature parameter and allows to control the shape of the normalized probability distribution. Then, the prediction y is compared to ground-truth labels, which are represented as one-hot vectors (also called *hard targets*), to produce the classification loss.

Because the softmax operation preserves ordering, the finally predicted class will be the one assigned to the highest score/probability, both in the logit and probability vectors. Nevertheless, the other classes also get a score, which is valuable information we can use. Indeed, when classifying an 8, class 3 might receive a high score, as both digits can look quite similar. This specific information is not available to the network in a standard training, as the ground-truth labels are represented as one-hot vectors.

In response-based distillation training, in addition to the standard classification loss computed between the probability vector and the hard targets, the student network has to optimize a *distillation loss* between its logits and those of the teacher (Fig. 13). The distillation loss for response-based knowledge can be formulated as $L_{\text{Logits}}(x) = (l_{\text{teacher}}(x), l_{\text{student}}(x))$,

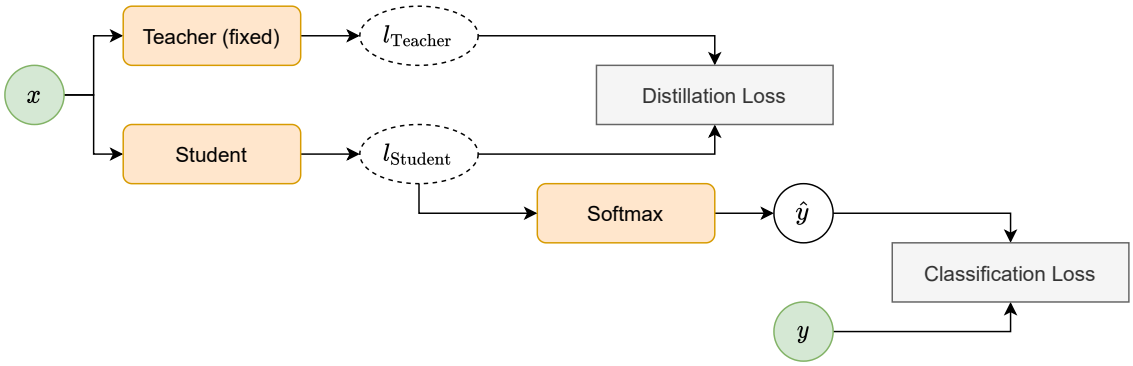


Figure 13: Training with knowledge distillation. The classification loss is completed with a distillation loss.

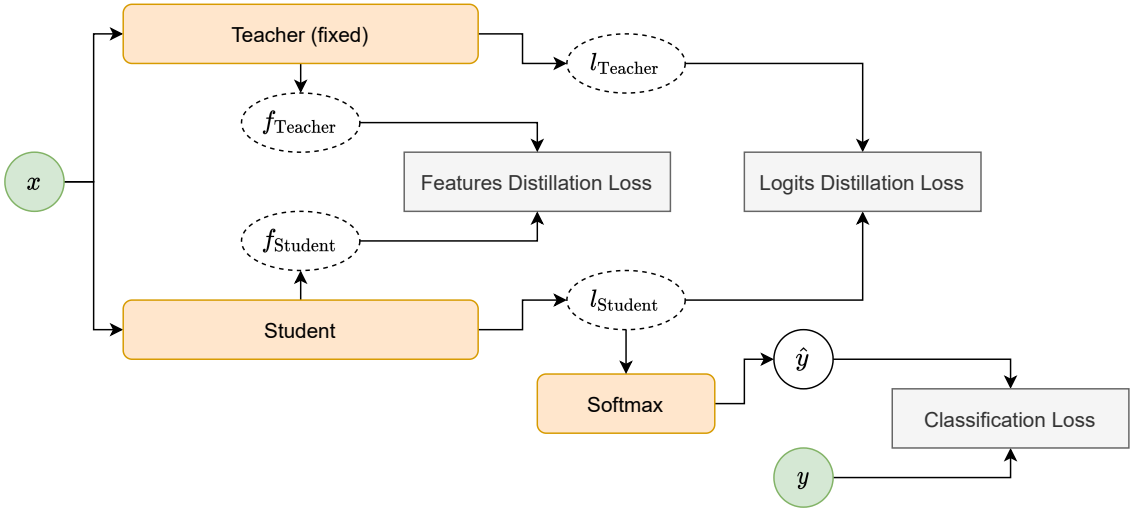


Figure 14: Feature-Based KD. A loss is computed between feature maps coming from the teacher and the student.

where l_{teacher} and l_{student} are the logits of the teacher and the student. Using a weighted average of these two loss functions, we are able to train a smaller student network to reach a similar accuracy to that obtained with a larger teacher model.

We have just presented content-based distillation but, as presented by Gou et al. in their survey [35], there exist different types of knowledge distillation. We want to talk about the *feature-based* KD. It is very similar to the content-based KD, but instead of using the output of the networks, we will rely on the feature maps, which are the outputs of the intermediate (hidden) layers of both networks. When designed smartly, one can ensure that some feature maps of the student have the same size as some feature maps of the teacher. We can then compute the feature distillation loss, formulated as $L_{\text{features}}(f_{\text{Teacher}}(x), f_{\text{Student}}(x))$, where, f_{Teacher} and f_{Student} are some feature maps of the intermediate layers of the teacher and the student.

In addition to choosing the right loss terms, the quality of knowledge acquisition and distillation from teacher to student is also determined by how to design the teacher and student networks. In knowledge distillation, how to select or design proper structures of teacher and student is a very important but difficult problem [36]. The model capacity gap between the large deep neural network and a small student neural network can degrade knowledge transfer. To effectively transfer knowledge to student networks, a variety of methods have been proposed for a controlled reduction of the model complexity [37]. In [38], they proposed a Residual Knowledge Distillation, which further distills the knowledge by introducing an assistant. Specifically, the student is trained to mimic the feature maps of teacher, and the assistant aids this process by learning the residual error between them.

2.4.2 Distillation of GANs

Recently, adversarial learning has received a great deal of attention due to its great success in generative networks [29]. Inspired by this, many adversarial knowledge distillation methods have been proposed.

In [39], the authors propose a method to compress GANs using knowledge distillation techniques. The teacher and student GANs use the original WGAN architecture. The authors experimented two methods; the first one uses MSE as the student training loss function using a pre-trained teacher generator, yielding the following optimization objective:

$$\min_{\theta} \mathbb{E}_{\mathbf{z}}[\|g_{teacher}(z) - g_{\theta}(z)\|^2] \tag{8}$$

The second experiment uses a joint loss function that combines regular GAN training (equation 6) with MSE loss. Specifically, the student is trained to optimize the following objective:

$$\min_{\theta} \max_{\phi} \Big[\mathbb{E}_{x\sim p_{data}}[\log(D_{\phi}(x))]+\mathbb{E}_{z\sim p_z}[\alpha \log(1-D_{\phi}(g_{\theta}(z)))+(1-\alpha)\|g_{teacher}(z)-g_{\theta}(z)\|^2]\Big] \tag{9}$$

where α controls the weight between the MSE loss and the regular GAN training. We must mention a blurry point here, as figures in the original paper mention relying on a teacher discriminator rather than training a student discriminator. This is in contradiction with the presence of an optimization over the discriminator parameters ϕ. Therefore, in the results section, we clarify how we have adapted these experiments to make them as clear as possible to the reader.

2.4.3 Other Compression techniques

There also exist two compressing techniques named pruning and quantization.

Pruning As proposed in [40, 41], it is possible to create smaller and more accurate models using another model compression technique called *pruning*. The method consists in removing weight connections in a network. The traditional technique called unstructured pruning acts on parameters deemed redundant or non-critical for network predictions by masking them. Usually, we consider that these non-critical parameters are a fixed ratio of the smallest weights. This method is however not suitable for embedded hardware because parameters are not removed but masked. Therefore the resulting model is not lighter nor less computationally-expensive.

A novel method called *structured pruning* or *trimming* addresses those memory and speed issues. Instead of masking individual weights in each layer of the neural network, this algorithm looks for ineffective entire units (neurons) to be removed from the model. As entire units are removed, considerable memory and inference time is saved because of the non-linear relationship between trimming and computation. This results in a full-fledged model compression, making it more embeddable. This approach has already proven to be successful when applied to MIR tasks (pitch estimation, chord estimation, drum transcription ...) on extremely compressed (up to a factor 100) state-of-the-art deep learning models without considerable loss of accuracy [42].

Quantization Quantization is the process of approximating a neural network that uses floating-point numbers by a neural network with lower bit width weights [43]. The standard bit width for neural networks is 32 bits. We can quantize the model after the training (post-training quantization), but also during the training (quantization-aware training) [44]. In quantization, the goal is to reduce the precision of the neural network parameters θ and the intermediate activation maps to low precision, with minimal impact on the accuracy of the model.

Typically, float32 networks can be quantized to half-precision (float16 numbers, 16 bits resulting in 2x compression) or to integer-based precision (int8 numbers, 8bits resulting in 4x compression). However, the expected speedup is often smaller than the compression ratio, depending on the casting rules setup by the computer's CPU. For instance, according to Tensorflow's guide to quantization, most CPUs have to de-quantize the float16 values to float32, discarding the benefits of the quantization step.

To conclude, these compression techniques can also be combined to obtain further gains in size reduction, as in [45].

3 Experiments and results

After having studied generative models in audio and compression techniques, we implemented a basic classifier. We then tested different techniques of knowledge distillation on this toy model, following the paper [39]. We then switched to GAN models [29], and retrained DrumGAN [20] with the distillation techniques we tried earlier. This led us to different version of student DrumGAN that we quantitatively and qualitatively measured. This includes perceptive listening of the generated samples, but also the calculation of the Fréchet Audio Distance (FAD) [33]. At last, we compared the influence of some parameters on the inference/generation time. This would measure if the model could be embedded on light computers and run in real-time.

3.1 Preliminaries - Distillation of a basic classifier

To get familiar with distillation methods, we first implemented a classifier that we then distilled following the method of [39]. The toy datasets we used in this experience were MNIST and FashionMNIST. We started with a simple ANN with fully connected layers, and then tried with a CNN.

Following the architectures presented in [35], we built a teacher network and a corresponding student network. We want to study the effect of distillation methods, and impact of some basic parameters. The teacher was composed of four convolutional layers and two linear layers (Tab. 1). The number of parameters of the teacher was 312842. Compared to the teacher, the student was built with two times less convolutional layers. Our student eventually has 238986 parameters (Tab. 2).

We first trained our teacher following a standard classifier training (Fig. 12), with cross-entropy loss. This model eventually reached an accuracy of 86% on FashionMNIST. We then trained our student with three different approaches in order to compare the impact of different loss terms.

3.1.1 Student training with classification loss

The first step was to simply evaluate the impact of convolutional layers reduction on the model's accuracy. Hence, we started by training our student following the same training procedure as for the teacher, with cross-entropy loss (Fig. 12). The student reached an accuracy of 75%, showing the capacity loss implied by the missing convolutional layers.

Table 1: Our teacher architecture.

Operation	# Input Channels	# Output Channels	Kernel size	Padding	Output Size (channels, H, W)
	Image Input of size (1,28,28)				(1,28,28)
Conv2d	1	64	(3,3)	(1,1)	(64,28,28)
ReLU	-				(64,28,28)
MaxPool2d	64	64	(2,2)	-	(64,14,14)
Dropout	Dropout 0.2				(64,14,14)
Conv2d	64	64	(3,3)	(1,1)	(64,14,14)
ReLU	-				(64,14,14)
MaxPool2d	64	64	(2,2)	-	(64,7,7)
Conv2d	64	64	(3,3)	(1,1)	(64,7,7)
ReLU	-				(64,7,7)
Conv2d	64	64	(3,3)	(1,1)	(64,7,7)
ReLU	-				(64,7,7)
MaxPool2d	64	64	(2,2)	-	(64,7,7)
Linear	3136	64	-		64
Linear	64	10	-		10

Table 2: Our student architecture. Two convolutional layers have been removed, reducing the capacity of the model.

Operation	# Input Channels	# Output Channels	Kernel size	Padding	Output Size (channels, H, W)
	Image Input of size (1,28,28)				(1,28,28)
Conv2d	1	64	(3,3)	(1,1)	(64,28,28)
ReLU	-				(64,28,28)
MaxPool2d	64	64	(2,2)		(64,14,14)
Dropout	Dropout 0.2				(64,14,14)
Conv2d	64	64	(3,3)	(1,1)	(64,14,14)
ReLU	-				(64,14,14)
MaxPool2d	64	64	(2,2)		(64,7,7)
Linear	3136	64	-		64
Linear	64	10	-		10

3.1.2 Student training with Classification Loss and Logit Distillation Loss

On the second stage, we added the logit distillation loss, as illustrated on figure 13. We computed the Mean Squared Error (MSE) between the softmax of the teacher and the student, as follows:

$$L_{\text{logits}} = \text{MSE}(t, s) = \frac{1}{n}\sum_{i=1}^{n}(t_i - s_i)^2. \tag{10}$$

where we consider t and s as the output of the softmax layer for the teacher and the student respectively. They are defined as :

$$t_i = \frac{e^{l_{t,i}/T}}{\sum_j e^{l_{t,j}/T}}, s_i = \frac{e^{l_{s,i}/T}}{\sum_j e^{l_{s,j}/T}} \tag{11}$$

where l_t and l_s are the teacher and student logits. We computed the accuracy for different temperatures T, which appeared to have a noticeable effect, accordingly to some previous work [35]. Overall, the student model trained with $L_{\text{logits}} + L_{\text{classif}}$ reached a better accuracy than the one trained with L_{classif} only.

Temperature T	1	2	3	4	5
Prediction Accuracy (%)	81.2	81.1	79.2	80.6	81.4

Table 3: Accuracy scores for different values of the temperature T

3.1.3 Student training with Classification Loss, Logit Distillation Loss and Feature Distillation Loss

The third implementation was a training with the two previous losses, plus a feature distillation loss. The feature distillation loss, illustrated in figure 14, is a loss between the feature maps of the teachers and student. As presented earlier, the student has two convolutional layers, and the teacher four. So the idea was to extract the outputs of the second and fourth convolutional layers of the teacher (Tab. 1), which were designed to be the same size as the first and second layers of the student (Tab. 2). This allowed us to compute a mean square error between those outputs, as follows:

$$L_{\text{features}} = \text{MSE}(f_A, f_2) + \text{MSE}(f_B, f_4) \tag{12}$$

where f_A and f_B are the first and second convolutional layers of the student, and f_2 and f_4 are the second and fourth convolutional layers of the teacher. The final loss had thus three components:

$$L_{\text{total}} = L_{\text{classif}} + L_{\text{logits}} + L_{\text{features}} \tag{13}$$

This time, we reached an accuracy of 85% for the student, an improvement of 10% compared to the same model trained on classification only. This shows how impactful the distillation losses are. In addition, despite not doing better than the teacher, we have observed that the student reached 85% accuracy much faster than the teacher reached 86%.

3.2 DrumGAN Distillation

After having tested the benefit of the knowledge distillation technique on a toy example, the goal was to build a new DrumGAN based on the distillation method. We mainly rely on the distillation techniques of GANs developed in the articles [39] and [35]. Since we had to build a drum generator able to synthesize new samples, we only needed to re-build a generator, and no discriminator. For all following sections, some samples of generated audio and spectrograms can be listened and observed on the accompanying website[1].

3.2.1 Reducing the number of convolutional layers

In each block of the DrumGAN generator, there are 2 convolutional layers (Fig. 11). The idea was to consider the architecture with only one convolutional layer per block (see appendix A). We would eventually obtain the same output size, while reducing the number of parameters by 30% (Tab. 4).

	Number of Conv2d in each block	Number of parameters
Teacher	2	8.8 millions
Student	1	6.1 millions

Table 4: Size comparison between teacher and student model

The first experiment consisted in training the student with an MSE loss that minimizes the pixel level error between the images generated from the student and the teacher (Fig. 15).

[1] `https://jeremybboy.github.io/compressed-drumgan/`

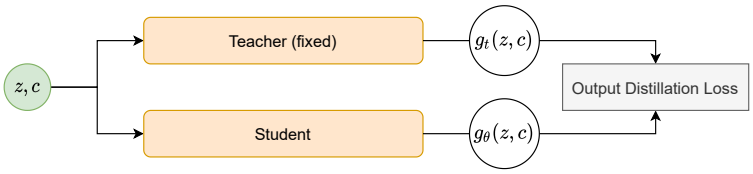


Figure 15: Initial distillation training scheme. The same input vector is fed into the teacher and the student. Then, we compute the MSE between the generations.

Therefore, we simply train the student by solving the optimization problem :

$$\min_{\theta} \mathbb{E}_{(\mathbf{z},\mathbf{c})}[\|g_t(z,c) - g_\theta(z,c)\|^2] \tag{14}$$

Concerning the input, as in the original DrumGAN paper, z is sampled from a 128-dimensional Gaussian distribution. However, as our teacher provides the training data here, there is no need for a dataset. In addition, as we want our student model to best replicate the teacher's results for all values of c, there is no restriction on the sampling strategy of c. We started by sampling c from a 3-dimensional Uniform distribution between 0 and 1.

Hence, DrumGAN takes as input a 131-dimensional vector and generates two tensors of size [1,1024,64] for the real and imaginary parts. Rather than computing the MSE on the real and imaginary parts directly, we match the literature and compute the MSE between the two outputs amplitudes defined as:

$$\begin{aligned} Amp_t &= log[\mathrm{Re}(g_t(z,c))^2 + \mathrm{Im}(g_t(z,c))^2] \\ Amp_s &= log[\mathrm{Re}(g_\theta(z,c))^2 + \mathrm{Im}(g_\theta(z,c))^2] \end{aligned} \tag{15}$$

We monitored the convergence of the student network based on the generated outputs and the loss trajectory. The student model was able to generate quite interesting samples that can be heard here [2]. The model learned an overall definition of drums, mixing the rich spectrograms of cymbals, but also a strong attack, an important component in kicks.

Four samples of teacher and student are presented in the figure (16). We remind here that the expected spectrogram from the teacher and the student are the same, because the input vector is the same. We can observe that the student generated spectrograms were blurry. In this training, we were asking the student generator to minimize the Euclidean distance, and more precisely the averaged ([46]) distance between the generated images of the teacher and

[2] `https://jeremybboy.github.io/compressed-drumgan/`

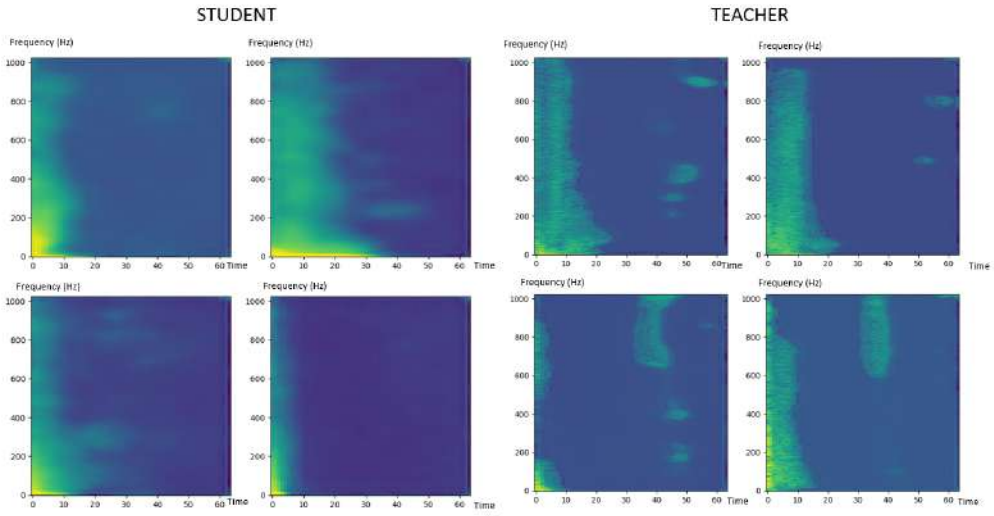


Figure 16: The generated magnitude spectrograms we see were obtained by feeding the trained models (Student on the left and Teacher on the right) with 4 latent vectors of size 131. The output size is a vector of size (4, 2, 1024, 48). The student and the teacher generate 4 matrices with real and imaginary parts of size (1024,48).

student. As a consequence, it tends to produce blurry results.

Following the experiments with the toy model, we then considered a training with a feature loss. This added loss had previously demonstrated a positive effect on the model performance.

3.2.2 Creating fake features

In a first time, we did not have access to the teacher features. Therefore, we decided to experiment the effect of adding "fake" teacher features. Those fake features were obtained by downsampling the final output of the teacher (size [2,1024,64]) at the corresponding size for every features outputs. Indeed, during a progressive growing training, feature maps go through *toRGB* layers, and should then resemble downsampled versions of real data. We replicated this in our distillation training (Fig. 17).

In order to learn not only low-level but also high-level information from the teacher generator, we merge the two above loss functions. Thus, we trained the student by solving the optimization problem :

$$\min_{\theta} \mathbb{E}_{(\mathbf{z},\mathbf{c})}[\|g_t(z,c) - g_\theta(z,c)\|^2 + \sum_{n=1}^{6} \|\mathrm{DS}_i(g_t(z,c)) - \mathrm{toRGB}_i(f_{s,i}(z,c,\theta)\|^2] \qquad (16)$$

where $f_{s,i}$ corresponds to the i^{th} features of the student, $toRGB_i$ corresponds to the i^{th} $toRGB$

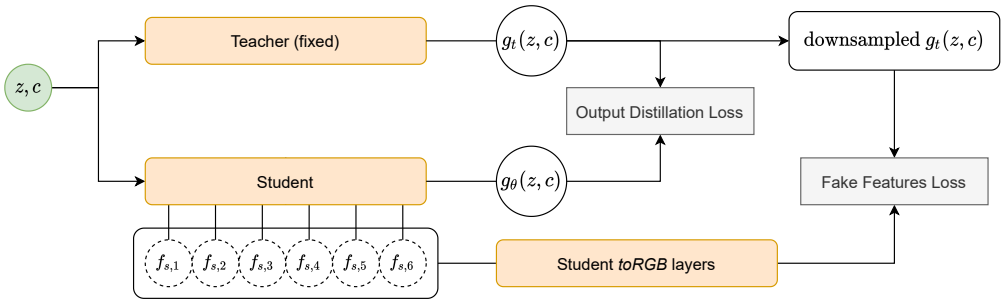


Figure 17: Our training with fake features. We use the *toRGB* layers in order to translate the student features in the data domain.

layer of the student, and $\mathrm{DS}_i(g_t(z,c))$ correspond to the downsampled version of the teacher's output that matches the size in scale i.

Unfortunately, we observed that this added loss didn't help the training. As mentioned on the CNNs section, the features maps usually allow to extract implicit knowledge. Intuitively, we can interpret that these fake teacher features didn't contain any knowledge, there were only interpolated images of the final output.

3.2.3 Add the Teacher features

Following the paper distillation of GAN [39], and our previous experiments on the CNN classifier, we then consider the actual teacher features. The intermediate layers provide rich information and allow the student model to acquire more knowledge in addition to the teacher outputs that we used in the previous experiments.

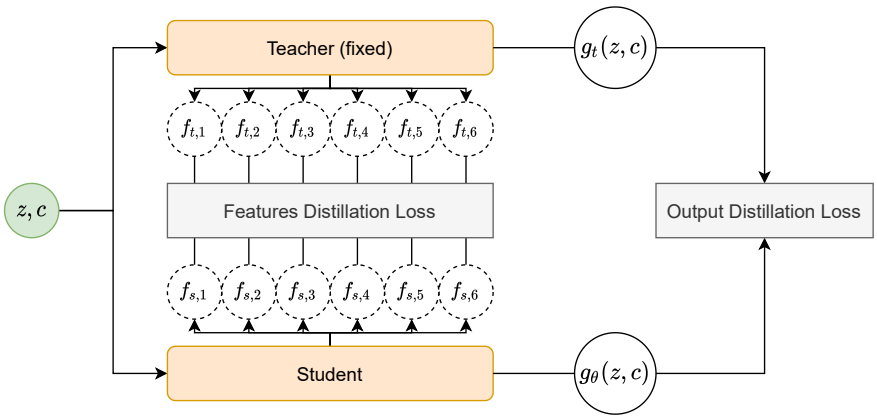


Figure 18: Student training with the added feature loss. Teacher and student are composed of 6 blocks (see A), from which we extract the feature map outputs and compute a MSE Loss.

We used the MSE as the feature distillation loss:

$$L_{\text{features}=} \sum_{n=1}^{6} \|f_{t,i}(z,c) - f_{s,i}(z,c,\theta)\|^2 \tag{17}$$

Therefore, optimizing problem of the proposed method was :

$$\min_{\theta} \mathbb{E}_{(\mathbf{z},\mathbf{c})} \left[\|g_t(z,c) - g_\theta(z,c)\|^2 + \sum_{n=1}^{6} \|f_{t,i}(z,c) - f_{s,i}(z,c,\theta)\|^2 \right] \tag{18}$$

This feature loss really helped the student. Firstly, we obtained nicer generated samples, with less artifacts[3]. Moreover, we obtained a better score in the Fréchet Audio Distance, as we can observe in the table 5. Eventually, the feature loss seems to have stabilized the training, as we see in the figure 19.

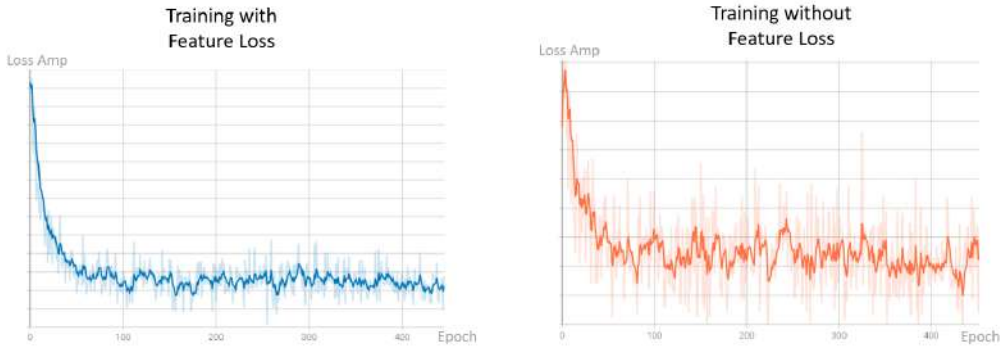


Figure 19: Comparison of the losses over epochs. On the left, the training with the feature loss. On the left, the training of the first student model, without the feature loss. They were extracted from Tensorboard, with the same smoothing scale of 0.75.

3.2.4 Add a discriminator loss

In order to bring more guidance to the generator, we tried to add another penalty ; the Wassertein distance. This metric is detailed in the section 2.2.3. When doing an adversarial training, the generator keeps proposing new images that tend to be more and more realistic. But this realism is due to the discriminator model which always back propagate information to the generator, by telling if the images are true or fake. In order to reproduce this, we added a trained discriminator that would evaluate the generations of the student generator. By adding as well this adversarial loss from a discriminator teacher, we could gain a bit more

[3] `https://jeremybboy.github.io/compressed-drumgan/`

precision (sharpness) in the generated spectrograms. Here, we train the student by solving the optimization problem :

$$\min_{\theta} \mathbb{E}_{(\mathbf{z},\mathbf{c})}\Big[\|g_{teacher}(z) - g_\theta(z)\|^2 + \sum_{n=1}^{6}\|f_{t,i}(z,c) - f_{s,i}(z,c,\theta)\|^2 + W(p_t||p_s)\Big] \tag{19}$$

where p_t and p_s are respectively teacher and student generation distributions.

This added loss consequently helped the student. We obtained more realistic drums than with the previous model[4]. However, a large proportion of the samples generated were sorts of Kicks. This phenomenon can be interpreted as a mode collapse [31]. GANs can sometimes suffer from the limitation of generating samples with little representative of the population. The model fails to generalize to the all variety of possible distributions and samples and almost only generate distributions of kicks. Doing that allows it to minimize its loss function. That may highlight that the student realized that energy is important in the low frequency for all three kinds of drums (Fig. 20). Indeed, we can observe from the spectrograms that almost all the generations have a strong value in the low frequencies. Thus the model may have decided to generate all the samples with a strong value in the low frequency the reduce the overall loss.

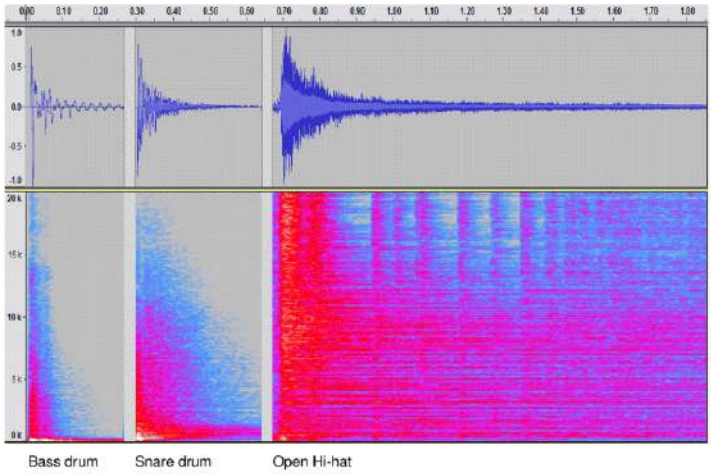


Figure 20: Drums spectral comparison. We can observe that the three drums have a strong energy in low frequencies. Moreover, we observe that the bass drum/kick spectrogram is the poorest in terms of harmonics and thus the easiest to learn.

[4] `https://jeremybboy.github.io/compressed-drumgan/`

3.2.5 Use a real Dataset

At first, no dataset was really needed. Indeed, the trained teacher generator was used as a data generator. The student goal was to generate, from any input vector, the exact same image as the teacher. As described earlier, the input vector used here is the concatenation of a Gaussian random vector of size 128 and a conditional vector of size 3. The last will give information on how much "Kick", "Cymbal" and "Hats" need to be felt in the generated sample. This triplet is called the soft-label c. In order to improve the results, we then thought that it could be relevant to use real values for c, rather than sampling it from a uniform distribution. Indeed, the initial training of DrumGAN was made with a dataset of 300 000 samples and every sample had its specific soft-label that was extracted with feature extractors. Thus, we recreated a realistic dataset of soft-labels that would be used as input for the student generator.

In terms of generation variety and quality, this last model is the most interesting one[5].

The distilled model we obtain was able to generate some interesting samples, with more variety than the previous models (Fig. 21). We can also easily see that the student performed better in approaching the expected spectrogram of its teacher.

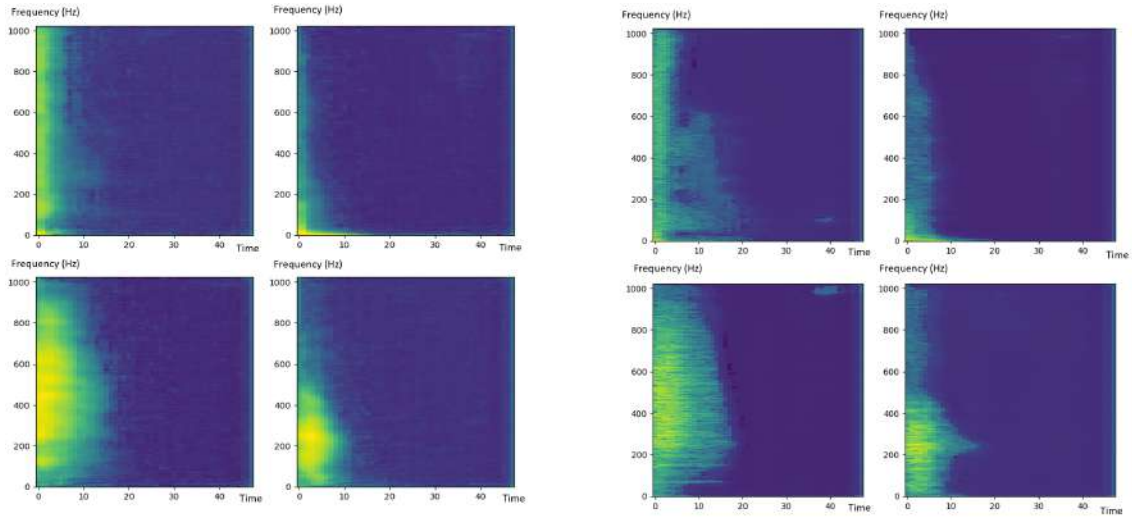


Figure 21: On the left: generations of the student. On the right : generations of the teacher

[5] https://jeremybboy.github.io/compressed-drumgan/

3.3 Evaluation Metrics

Objective evaluation of generated audio is not an easy task. Still, specific metrics and techniques make it possible to obtain quite relevant qualitative and quantitative evaluations.

3.3.1 Qualitative Evaluation

Many different approaches have been used to evaluate generative models and they all have some bias [47]. With GANs, the goal is to create a model able to generate new audios. So we can't compare it to some existing inputs. One intuitive metric of performance can be obtained by having human annotators judging the perceptive quality of audio samples. One could ask a group of people to rate examples of real and generated audios. This is a labor-intensive exercise, although costs can be lowered by using a crowd sourcing platform, and efficiency can be increased by using a web interface. A major downside of the approach is that the performance of human judges is not fixed and can improve over time. This is especially the case if they are given feedback, such as clues on how to detect real and generated samples. It is still harder to fool a discriminator than a human, fortunately. Still, the first perceptual evaluation can be done by evaluating different samples from the student models we trained during the experiments in [6].

3.3.2 Quantitative Evaluation

Besides perceptual evaluation of the generations of the models, there are specific numerical scores used to summarize the quality of generated audios. The two most common GAN evaluation measures are Inception Score (IS) and Fréchet Inception Distance (FID).

Inception score IS uses an Inception model classifier to measures if the generated images are or not easy to classify. If a generated dataset can be easily organized, it means that it can be considered as a good dataset. It also measures the diversity of generations, by calculation the distributions of classes in the generated dataset. However, Inception score does not consider real images at all, and so cannot measure how well the generator approximates the real distribution.

Fréchet Audio Distance The Fréchet Inception Distance (FID) measures the distance between the feature vectors (or embeddings) of generated images and real images. The idea

[6] `https://jeremybboy.github.io/compressed-drumgan/`

is to extract the feature vectors of real and generated images from a model pre-trained for classification, thus yielding meaningful features. A low FID means that the distributions of features are close. Let's consider the distribution of two models p_f and p_r. We calculate the feature wise mean of the feature vectors (m_f and m_r) and the covariance of the feature vectors (C_f and C_r) to obtain a Gaussian (maximum entropy distribution). We then compute the FID formula:

$$F((m_f, C_f), (m_r, C_r)) = ||m_f - m_r||_2^2 + Tr(C_f + C_r - 2((C_f C_r)^{\frac{1}{2}}))$$

When the distributions are 1-dimensional, the FID formula is equivalent to :

$$F(p_f, p_r)) = ||m_f - m_r||_2^2 + ||C_f - C_r||_2^2$$

The Fréchet Audio Distance (FAD) was introduced in [33] and is based on the FID that we just presented. Like the FID, The FAD is interesting because is allows to score an evaluation set without a ground truth reference image. Indeed, we cannot compare a pixel to pixel generated image and the real image, because the generated image is unique. If we cannot compare images, we can still compare embeddings generated from fakes images of GANs and real images. Similar to FID, the embeddings are obtained from a pre-trained model called VGGish [48]. This model derives from the VGG image recognition architecture, and has been trained on Youtube video as an audio classifier with 3000 classes.

The input we give to the model is log-mel features extracted from the magnitude spectrogram over 1s of audio. Even if the audios usually lasts longer than 1s, the model has proven efficient enough. The feature maps layer are closer to output nodes that correspond to real-world images of spectrograms such as a specific timbre of Rhodes or a guitar bass, and further from the shallow layers near the input image, as we see in figure (22). As a result, they tend to mimic human perception of similarity in sounds.

Ablation study To better evaluate the performance of our trained models, and measure the contribution of the components we used for our knowledge distillation, we computed the FAD of the different student models. We recall that the lower the FAD, the smaller the distance between distributions of real and generated data. The student n°3 is the simplest one, where we only computed the MSE loss between the generations of the teacher and student outputs [39]. With the student n°2, we then added a feature loss, computing the MSE between the teacher and student feature maps. This loss had a serious impact since the FAD dropped from 18.52 to 6.91. With the student n°1, we eventually added a teacher

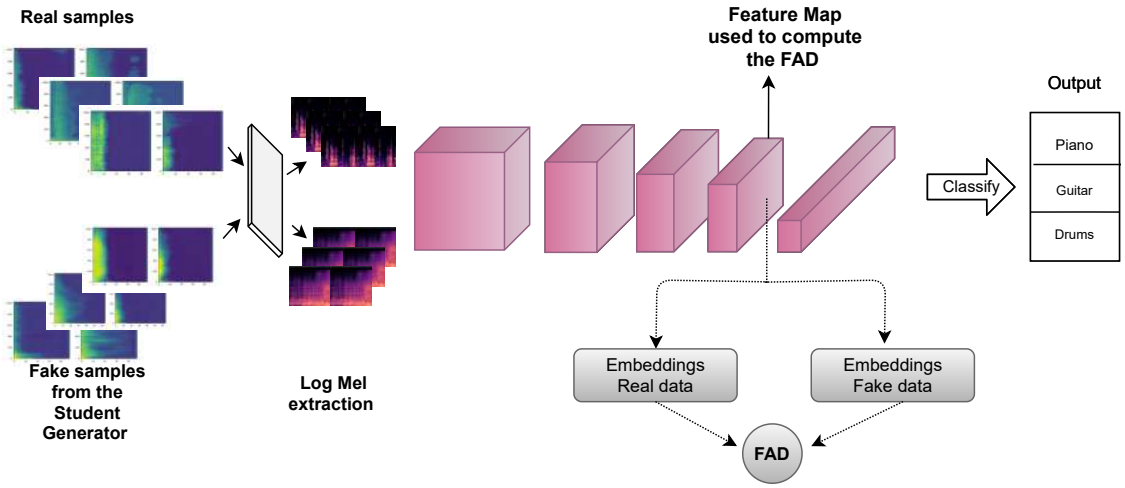


Figure 22: Extraction of embeddings from the VGGish network. On the left, the real and the generated samples. After having extracted the Log Mel features of the samples, we pass them in the VGGish network (pink). We use the last convolutional feature map to obtain embeddings for real and generated data. We then compute the FAD between those two embeddings.

Models	Loss type	Fréchet Audio Distance
Teacher	*WGAN Losses*	*2.11*
Student 1	MSE outputs + Features Loss + Classification Loss	4.94
Student 2	MSE outputs + Features Loss	6.91
Student 3	MSE on outputs	18.52

Table 5: Fréchet Audio Distance of the different models. We used 5000 drums samples to calculate the background embeddings. We then calculated the embeddings of the different trained models generations and eventually computed the FADs.

discriminator loss. This loss had a positive effect on the model, making the FAD drop to 4.91. Still, it is reasonable to say from the FAD results in table 5 that the influence of this loss was not as important as the feature loss.

3.3.3 Test of the compressed model on small computers

Even if the generations by the new models we trained were not as relevant as the ones of the original DrumGAN, we wanted to test and run them on different computers. This would allow us to know in what extent a compressed model could be embedded on tiny devices. We used a computer with a graphic processor unit (GPU Nvidia GTX1080), a laptop with a i7 CPU (16gb RAM), and a Raspberry Pi 4 (4gb RAM). We measure the time for a trained generator model to generate 1 sample, and a batch of 64 samples (Fig. 6). Calculations

Model	GPU		CPU		Raspberry	
Nb of samples	1 sample	64 samples	1 sample	64 samples	1 sample	64 samples
Teacher	598 ms	785 ms	195 ms	5308 ms	6156 ms	X
Student	526 ms	662 ms	147	2932 ms	3081 ms	X

Table 6: Inference time on different devices.

on Raspberry are still in process. We already observe an improvement with the GPU. The student is more than 10 % faster in terms of generation time for both 1 and 64 samples.

We observe that the CPU time for 1 generation is faster than the GPU time. That's because CPUs are designed with fewer processor cores that have higher clock speeds (in our CPU, up to 4.2 Ghz) than the ones found on GPUs (in our GPU, up to 1,7 GHz), allowing them to complete series of tasks very quickly. GPUs, on the other hand, have much greater number of cores and are designed for a different purpose. When it comes to multiple generations, we observe a strong gap between CPU and GPU. Indeed, it takes almost 3 seconds for the student to generate 64 samples with a CPU, when it took 662 ms with a GPU. GPUs generate more quickly than a CPU because of its parallel processing architecture, which allows it to perform multiple calculations simultaneously. Coming back to teacher-student comparison, we observe more than 2 seconds difference between the generation time student and teacher. This confirms that compressing the architecture of a neural network has a relevant impact on inference time.

The Raspberry was not able to pass 64 samples at the same time. Torch.conv2d throws runtime error if invoked with large kernels. Indeed, Pytorch versions for ARM are specific, and they use an acceleration package for neural networks called NNPACK. This library was originally created to optimize models like Darknet without using a GPU. It is useful for embedded devices using ARM CPUs.

4 Discussion and Conclusion

Discussion In this internship, we studied compressing techniques aimed at reducing power consumption and inference time of generative models in audio. Due to time constraints, we focused our experiments on Knowledge distillation (KD) [34]. We first re-implemented two knowledge distillation techniques presented in [39], and then trained a student DrumGAN generator. In conclusion, we obtained a smaller DrumGAN, able to generate quite interesting sounds, especially kick drums samples. The inference time was also reduced, which confirmed the potential benefit of such compressing method for our project. We conducted an ablation study to evaluate the performance of our trained model, and measure the contribution of the components we used for our knowledge distillation. Our experiments confirmed the impact of the feature loss and concluded that it really helped the training. However, the final student model we obtained was not able to compete with his teacher in terms of perceived audio quality[7] nor quantitative performance [33]. Also, the time gained seems not to be enough in order to develop the visualization discussed in the introduction.

A noticeable problem we have faced during the end of this internship is that DrumGAN's research framework was not really designed for the kind of production-time optimization we wanted to perform. This made the last round of experiments (progressive distillation and quantization) quite tricky.

In terms of knowledge distillation, a future experiment would consist in implementing a progressive growing distillation training. This would imply a classical training of PG-WGAN, plus an added distillation loss from a teacher network that would guide the student and make the training faster. This could even allow to further compress the model in order to gain significant time. We have tried to setup such a training during the last month of this internship but given the complexity of the existing code base we used, it turned out to be more difficult than expected.

It would also be interesting to test the other techniques that were introduced in the state of the art. We could implement a quantization-aware training [43] and evaluate the resulting quantized model performance. The structured pruning technique [45] would have to be adapted to the progressive growing training [25].

Finally, deep learning frameworks for on-device inference have recently been developed, e.g. Tensorflow Lite. Compared to Pytorch or Tensorflow, they are optimized for light models and embedding on light CPUs like smartphones. They are widely supported and can be found

[7] `https://jeremybboy.github.io/compressed-drumgan/`

in many frameworks, tools, and hardware. Many AI projects including an audio classifier[8] have been developed on Raspberry using Tensorflow lite. Hence, we could try to convert our Pytorch-based model to Tensorflow lite. This could be done using a standardized format for neural networks called Open Neural Network Exchange (ONNX), that also contains an inference accelerator, called ONNX Runtime, that must be worth trying.

An interesting consideration that I have been discussing with the other researchers in the lab is the alternative of using instance of GPU in the cloud, in order to run the model with no inference latency, and almost no latency at all given existing internet speeds. Strong and heavy models can be run on the tiniest device, as long as it can be connected to internet and call the API to receive the inference result from the server. But it makes the hardware dependant on Internet access. Even if our coming connected devices era suggests that internet access will be widely accessible, this is not that much portable. This could even be a major issue as some studios decide not to setup an internet connection to minimize the risk of hackers attacks and music leaks.

References

[1] Cheng-Zhi Anna Huang, Hendrik Vincent Koops, Ed Newton-Rex, Monica Dinculescu, and Carrie J Cai. Ai song contest: Human-ai co-creation in songwriting. *arXiv preprint arXiv:2010.05388*, 2020.

[2] Rebecca Fiebrink and Perry R Cook. The wekinator: a system for real-time, interactive machine learning in music. In *Proceedings of The Eleventh International Society for Music Information Retrieval Conference (ISMIR 2010)(Utrecht)*, volume 3, 2010.

[3] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *International Conference on Machine Learning*, pages 1362–1371. PMLR, 2017.

[4] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

[5] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. Ddsp: Differentiable digital signal processing, 2020.

[8] `https://www.tensorflow.org/lite/tutorials/model_maker_audio_classification`

[6] Emilia Gómez, Merlijn Blaauw, Jordi Bonada, Pritish Chandna, and Helena Cuesta. Deep learning for singing processing: Achievements, challenges and impact on singers and listeners. *arXiv preprint arXiv:1807.03046*, 2018.

[7] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[8] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

[9] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[10] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

[11] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[12] Sean Vasquez and Mike Lewis. Melnet: A generative model for audio in the frequency domain. *arXiv preprint arXiv:1906.01083*, 2019.

[13] Philippe Esling, Axel Chemla-Romeu-Santos, and Adrien Bitton. Generative timbre spaces: regularizing variational auto-encoders with perceptual metrics, 2018.

[14] Cyran Aouameur, Philippe Esling, and Gaëtan Hadjeres. Neural drum machine : An interactive system for real-time synthesis of drum sounds, 2019.

[15] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018.

[16] Adam Roberts, Jesse Engel, Yotam Mann, Jon Gillick, Claire Kayacik, Signe Nørly, Monica Dinculescu, Carey Radebaugh, Curtis Hawthorne, and Douglas Eck. Magenta studio: Augmenting creativity with deep learning in ableton live. 2019.

[17] Stefan Lattner and Maarten Grachten. High-level control of drum track generation using learned patterns of rhythmic interaction. In *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 35–39, 2019.

[18] Maarten Grachten, Stefan Lattner, and Emmanuel Deruty. Bassnet: A variational gated autoencoder for conditional generation of bass guitar tracks with learned interactive control. *Applied Sciences*, 10(18):6627, 2020.

[19] Théis Bazin and Gaëtan Hadjeres. Nonoto: A model-agnostic web interface for interactive music composition by inpainting. *arXiv preprint arXiv:1907.10380*, 2019.

[20] Javier Nistal, Stephan Lattner, and Gaël Richard. Drumgan: Synthesis of drum sounds with timbral feature conditioning using generative adversarial networks. *arXiv preprint arXiv:2008.12073*, 2020.

[21] Christopher M Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[23] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[26] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*, 2020.

[27] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[28] Javier Nistal, Stefan Lattner, and Gael Richard. Comparing representations for audio synthesis using generative adversarial networks. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 161–165. IEEE, 2021.

[29] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[30] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242, 2016.

[31] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans) challenges, solutions, and future directions. *ACM Computing Surveys (CSUR)*, 54(3):1–42, 2021.

[32] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[33] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. Fr\'echet audio distance: A metric for evaluating music enhancement algorithms. *arXiv preprint arXiv:1812.08466*, 2018.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[35] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, pages 1–31, 2021.

[36] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[37] Hui Wang, Hanbin Zhao, Xi Li, and Xu Tan. Progressive blockwise knowledge distillation for neural network acceleration. In *IJCAI*, pages 2769–2775, 2018.

[38] Mengya Gao, Yujun Wang, and Liang Wan. Residual error based knowledge distillation. *Neurocomputing*, 433:154–161, 2021.

[39] Angeline Aguinaldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. Compressing gans using knowledge distillation. *arXiv preprint arXiv:1902.00159*, 2019.

[40] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[41] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[42] Philippe Esling, Theis Bazin, Adrien Bitton, Tristan Carsault, and Ninon Devis. Ultra-light deep mir by trimming lottery tickets, 2020.

[43] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[44] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[45] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[46] Hanting Chen, Yunhe Wang, Han Shu, Changyuan Wen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Distilling portable generative adversarial networks for image translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3585–3592, 2020.

[47] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[48] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

Appendices

A Model parameters configurations

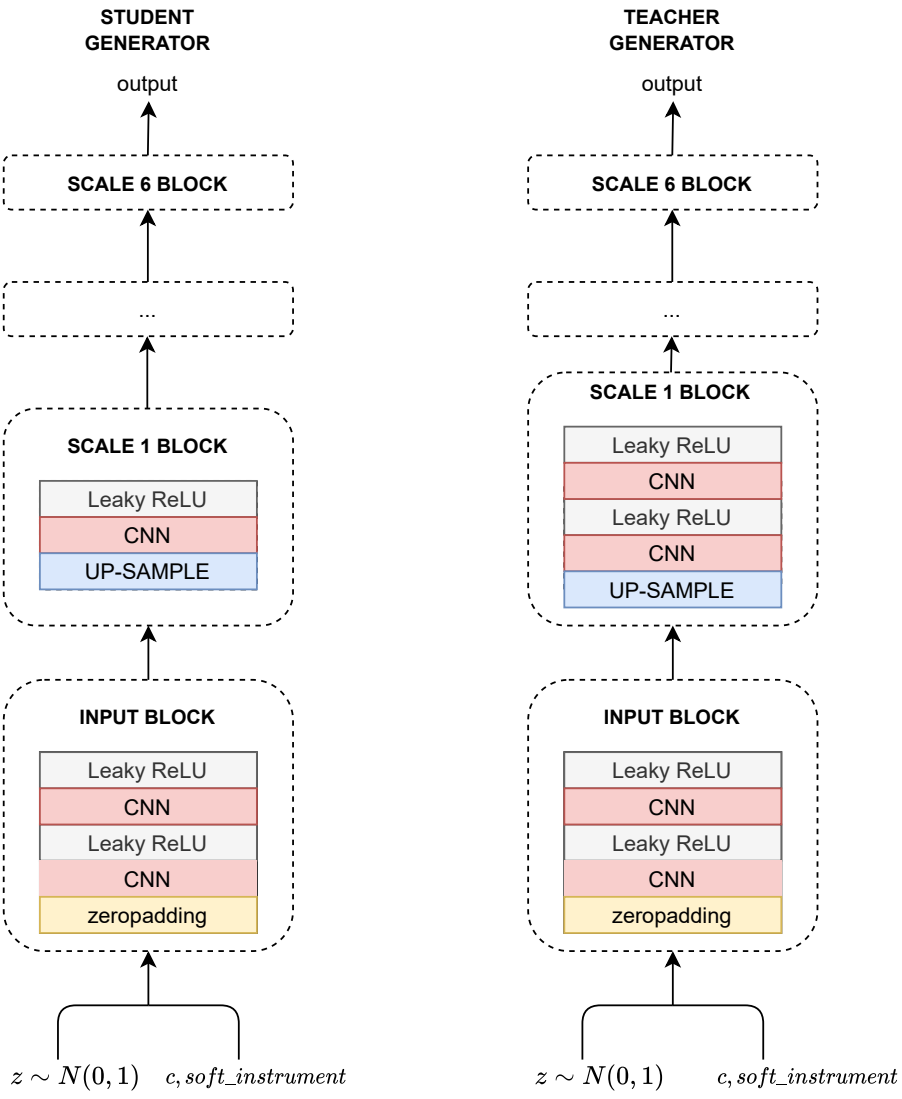


Figure 23: Student and teacher architecture

Table 7: DrumGAN student architecture. As we can observe, each block of the student is composed of Up-sampling, Conv2d and Leaky ReLU layers. The teacher has two Conv2d and LeykyReLU layers on each block, which makes is more complex.

Module List	Operation	IN Channel	OUT Channel	Kernel size	Padding	Output Size (channels, H, W)
Latent vector (131)	Unsqueeze	131	131			(131,1)
	Unsqueeze	131	131			(131,1,1)
Input Block	ZeroPad2d	-			(3,2,16,15)	(256,32,6)
	Conv2d	131	131	(33,7)	(16,3)	
	Conv2d	131	256	(3,3)	(1,1)	
	LeakyReLU	-				
	Conv2d	256	256	(3,3)	(1,1)	
	LeakyReLU	-				
Block 1	Up-Sample	Scale factor = (1,1), mode nearest				(256,32,6)
	Conv2d	256	256	(3,3)	(1,1)	
	LeakyReLU	-				
Block 2	Up-Sample	Scale factor = (2,2), mode nearest				(128,64,12)
	Conv2d	256	128	(3,3)	(1,1)	
	LeakyReLU	-				
Block 3	Up-Sample	Scale factor = (2,2), mode nearest				(128,128,24)
	Conv2d	128	128	(3,3)	(1,1)	
	LeakyReLU	-				
Block 4	Up-Sample	Scale factor = (2,2), mode nearest				(128,256,48)
	Conv2d	128	128	(3,3)	(1,1)	
	LeakyReLU	-				
Block 5	Up-Sample	Scale factor = (2,2), mode nearest				(64,512,48)
	Conv2d	128	64	(3,3)	(1,1)	
	LeakyReLU	-				
Block 6	Up-Sample	Scale factor = (2,2), mode nearest				(32,1024,48)
	Conv2d	64	32	(3,3)	(1,1)	
	LeakyReLU	-				
ToRGB	Conv2d	32	2	(1,1)	(1,1)	(2,1024,48)