MobileStyleGAN: A Lightweight Convolutional Neural Network for High-Fidelity Image Synthesis

Sergei Belousov
sergei.o.belousov@gmail.com

Abstract

In recent years, the use of Generative Adversarial Networks (GANs) has become very popular in generative image modeling. While style-based GAN architectures yield state-of-the-art results in high-fidelity image synthesis, computationally, they are highly complex. In our work, we focus on the performance optimization of style-based generative models. We analyze the most computationally hard parts of StyleGAN2, and propose changes in the generator network to make it possible to deploy style-based generative networks in the edge devices. We introduce MobileStyleGAN architecture, which has x3.5 fewer parameters and is x9.5 less computationally complex than StyleGAN2, while providing comparable quality.

1. Introduction

In recent years, high-fidelity image synthesis has significantly improved by through the use of Generative Adversarial Networks (GANs) [9]. Whereas early work such as DCGAN [27] could generate images having a resolution up to 64x64 pixels, modern networks such as BigGAN [3] and StyleGAN [20, 21, 19] allow the generation of photorealistic images with up to 512x512 and even 1024x1024 pixels. Although the quality of generative models has significantly improved, image generation still requires many computation resources. The high computational complexity makes it difficult to deploy state-of-the-art generative models to edge devices.

For example, the StyleGAN2 [21] network allows realistic face images 1024x1024 pixels in size with FID=2.84 for the FFHQ dataset. It, however, contains 28.27M parameters and has a computational complexity of 143.15GMAC.

We propose a new lightweight architecture, MobileStyleGAN, a high-resolution generative model for high-quality image generation. Taking as a baseline the original StyleGAN2 architecture, we revisit computationally hard parts of this network to create our own lightweight model that provides comparable quality (Figure 1). The whole network contains 8.01M parameters, has a computational complexity of 15.09 GMAC, and provides quality with FID=12.38 for the FFHQ dataset.

Our main contributions are:

- We introduce an end-to-end wavelet-based convolutional neural network for high-fidelity image synthesis.
- We introduce Depthwise Separable Modulated Convolution as a lightweight version of Modulated Convolution to decrease computational complexity.
- We introduce a revisited version of the demodulation mechanism applicable to graph optimizations such as operation fusion.
- We propose a pipeline based on knowledge distillation to train our network.

2. Related Work

2.1. StyleGAN

StyleGAN [20] is a modern generative model for high-resolution image generation. The key aspects of the StyleGAN network are:

- It uses progressive growing to increase the resolution gradually.
- It generates images from a fixed value tensor, as opposed to generating images from stochastically generated latent variables as in conventional GANs.
- The stochastically generated latent variables are used as style vectors through AdaIN [16] at each resolution after being nonlinearly transformed by an 8-layer neural network.

StyleGAN2 [21] improves upon StyleGAN by:

- Eliminating droplet modes by normalizing with estimated statistics instead of normalizing with actual statistics such as AdaIN.
- Reducing eye and tooth stagnation by using a hierarchical generator with skip connections instead of progressive growing.

Figure 1. (top) Images generated by StyleGAN2. (bottom) Images generated by MobileStyleGAN.

- Improving image quality by reducing PPL and smoothing the latent space.

StyleGAN2-ADA [19] makes StyleGAN applicable to tasks with limited data by using adaptive discriminator augmentations.

2.2. Model acceleration

A prominent research directions in deep learning focuses on accelerating to convolution neural networks (CNNs) using manual and automatic design of lightweight architectures.

Some works focused on designing efficient neural classification networks that can be used as a backbone for other tasks. Howard et al. [15, 14, 28] proposed efficient models called MobileNets for mobile and embedded vision applications. These ideas have been improved in many articles [29, 17, 8, 32, 26].

Other works focused on the design models of efficient generative models. Li et al. [23] proposed a generic framework for automatic optimization of the conditional GANs based on a distillation and neural architecture search [4]. Chang and Lu [5] proposed a distillation pipeline for compression of BigGAN. Liu et al. [24] proposed a lightweight GAN based on the attention mechanism.

2.3. Knowledge distillation

Hinton et al. [13] proposed the knowledge distillation method to train a small student network using a large teacher network. The main idea of the distillation is that the student network is trained to mimic the teacher network's behavior. Aguinaldo et al. [2] adopted knowledge distillation to accelerate unconditional GANs. Some other works related to GANs [30, 6, 23, 5] also used knowledge distillation as part of their pipeline.

2.4. Wavelet transform

Using wavelet-based methods with deep learning is not novel. Wavelets [10] have been applied to many computer vision tasks such as texture classification [7], image restoration [25],and super-resolution [31].

Han et al. [11] proposed Not-So-Big-GAN architecture based on two sub-networks: a generative network in low resolution and a super-resolution network for upsampling. The authors showed that wavelet-based sub-networks outperform pixel-based methods.

In comparison to previous works, we present an end-to-end wavelet-based CNN architecture for generative networks. We show that integrating wavelet-based methods into GANs allows more lightweight networks to be designed and provides more smoothed latent space.

3. MobileStyleGAN Architecture

Our proposed architecture, MobileStyleGAN, is based on previous works related to style-based generative models. MobileStyleGAN includes a mapping network and synthesis network, as in StyleGAN2. We adopt the mapping network from StyleGAN2, focusing on the design of a computationally efficient synthesis network.

We now first describe the key differences between our proposed architecture and the basic StyleGAN2. We then describe the distillation-based training procedure for the MobileStyleGAN network.

3.1. Image representation revisited

While StyleGAN2 works with pixel-based image representation and aims to directly predict the pixel values of the output image, in our work, we use a frequency-based representation of images. In this way, MobileStyleGAN seeks to predict the discrete wavelet transform (DWT) of the output image.

When applied to a 2d image, the DWT transforms the channel into four equally-sized channels having a lower spatial resolution and different frequency bands. The inverse discrete wavelet transform (IDWT) then reconstructs the pixel-based representation from the wavelet-domain (Figure 2).

This type of representation of images has several advantages, such as:

- Since wavelet-based image representation contains more structural information than pixel-based approaches, we can generate high-resolution images using low-resolution feature maps without losing accuracy:

$$|I - IDWT(DWT(I))|_{l1} < \epsilon \\ \epsilon \approx 1e-7 \quad (1)$$

- In our work, we use Haar wavelets [10] as a bank of filters for the DWT and IDWT. IDWT using Haar wavelets can be implemented efficiently without multiplication operations (Figure 2):

$$\begin{cases} A = LL + HL + LH + HH \\ B = LL - HL + LH - HH \\ C = LL + HL - LH - HH \\ D = LL - HL - LH + HH \end{cases} \quad (2)$$

- The generation of the high-frequency details of the image is a complex problem. While the latent space of StyleGAN is smoothed in low frequencies, it is rough in high frequencies. In contrast to pixel-based approaches, using frequency-based image representation, we can add regularization to the high-frequency components of the signal directly, which makes latent space smooth at both low and high frequencies.

3.2. Progressive growing revisited

StyleGAN2 uses a skip-generator to form the output image by explicitly summing RGB values from multiple resolutions of the same image. We found that when we predict images in the wavelet domain, the prediction head based on skip connections does not make an essential contribution to the quality of the generated image. Accordingly, to decrease computational complexity, we replace the skip-generator with a single prediction head from the network's last block. Predicting the target image from the intermediate

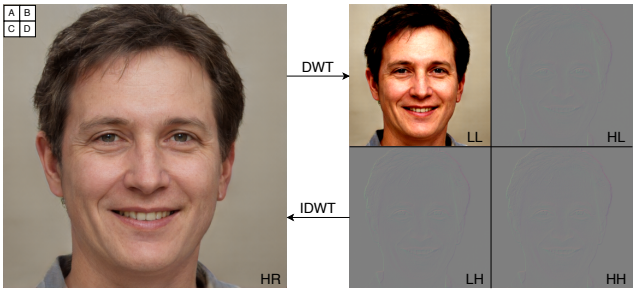

Figure 2. (left) The original RGB image. (right) DWT decomposition of the image. A, B, C, D are four pixels located in the top-left grid 2x2 of the HR image.

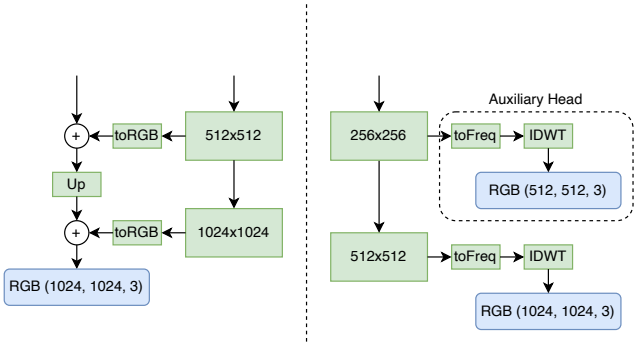

Figure 3. (left) Prediction head of StyleGAN2. (right) Prediction head of MobileStyleGAN.

blocks, however, is important in stabilizing image synthesis. We, therefore, add an auxiliary prediction head for each intermediate block to predict the target image according to its spatial resolution.

The difference between the StyleGAN2 and MobileStyleGAN prediction heads is shown in Figure 3.

3.3. Depthwise separable modulated convolution

Inspired by MobileNet [15], MobileStyleGAN is based on depthwise separable convolutions, a form of factorized convolutions that factorize a standard convolution into a 3x3 depthwise convolution and a 1x1 convolution called a pointwise convolution.

As described in [21], a modulated convolution consists of modulation, convolution, and normalization (left panel of Figure 4). Depthwise separable modulated convolution also comprises these parts (middle panel of Figure 4). We note, however, that while StyleGAN2 describes modulation/demodulation applied to weights, we apply them to input/output activations, respectively. This order of operations makes it easy to describe depthwise separable modulated convolution. Let us start by considering the effect of a modulation followed by a convolution. The modulation scales each input feature map of the convolution based on

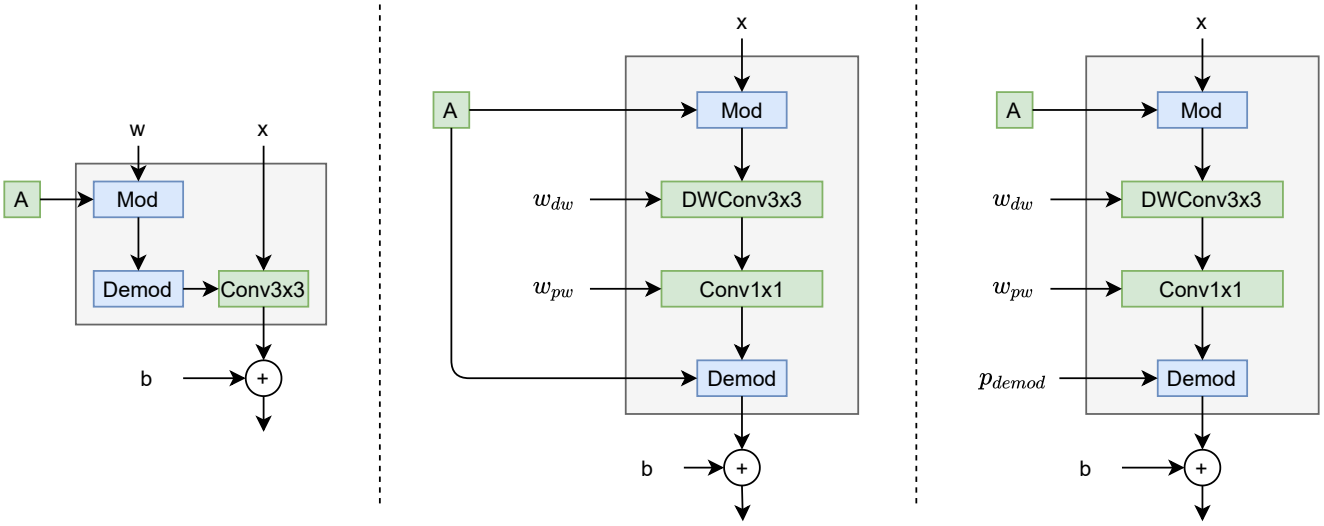

Figure 4. (left) Modulated convolution. (middle) Depthwise separable modulated convolution. (right) Depthwise separable modulated convolution with a trainable demodulation.

the incoming style:

$$x' = s * x \tag{3}$$

where x and x' are the original and modulated input activations, respectively, and s is the scale corresponding to the input feature maps. Then we apply 3x3 depthwise convolution and 1x1 pointwise convolution sequentially without any nonlinearity between them:

$$\begin{aligned} x'' &= w_{dw} * x' \\ x''' &= w_{pw} * x'' \end{aligned} \tag{4}$$

Now we apply demodulation to remove the effect of s from the statistics of the output feature maps. Due to the linearity of the convolution operator, the result of the sequentially applied depthwise and pixelwise convolutions are equal to the result of the applied dense convolution:

$$\begin{aligned} &w_{dense} = w_{dw} * w_{pw} \\ &w_{pw} * (w_{dw} * x') = (w_{pw} * w_{dw}) * x' = w_{dense} * x' \end{aligned} \tag{5}$$

In this way, demodulation coefficients can be calculated as:

$$\begin{aligned} w' &= w_{pw} * w_{dw} \\ demod_j &= \frac{1}{\sqrt{\sum_{i,k} (s_i * w'_{i,j,k})^2 + \epsilon}} \end{aligned} \tag{6}$$

where i and j and k enumerate the input/output feature maps and spatial footprint of the convolution, respectively. To demodulate the output feature maps, we apply:

$$x_{out} = demod * x''' \tag{7}$$

3.4. Demodulation fusion

Batch normalization fusion is a popular technique to decrease the computational complexity of convolutional networks at the inference time. This technique relies on the fact that we can merge two linear operations into one. The demodulation mechanism is similar to batch normalization, but it is not a linear operation at the inference time. Following Equation 6, where weights are fixed, the demodulation is the function of style. To make the demodulation constant, we replace style coefficients with trainable parameters (right panel of Figure 4):

$$\begin{aligned} w' &= w_{pw} * w_{dw} \\ demod_j &= \frac{1}{\sqrt{\sum_{i,k} (p_{demod} * w'_{i,j,k})^2 + \epsilon}} \end{aligned} \tag{8}$$

Thus, demodulation becomes the constant at the inference time and can be merged into the pixelwise convolution weights. We found that this technique does not adversely affect the quality of the generated image.

3.5. Upscale revisited

While the StyleGAN2 building block uses ConvTranspose (left panel of Figure 5) to upscale input feature maps, following Section 3.1, we use IDWT as an upscale function in the building block of MobileStyleGAN (right panel of Figure 5). Since IDWT does not include trainable parameters, we add extra depthwise separable modulated convolution after the IDWT layer.

The complete building block structures for StyleGAN2 and MobileStyleGAN are shown in Figure 6.

4. Training framework

As in previous works [5, 23], our training framework is based on the knowledge distillation technique [13]. Given StyleGAN2 [21] as the teacher network, we train MobileStyleGAN to mimic its functionality. The overall frame-

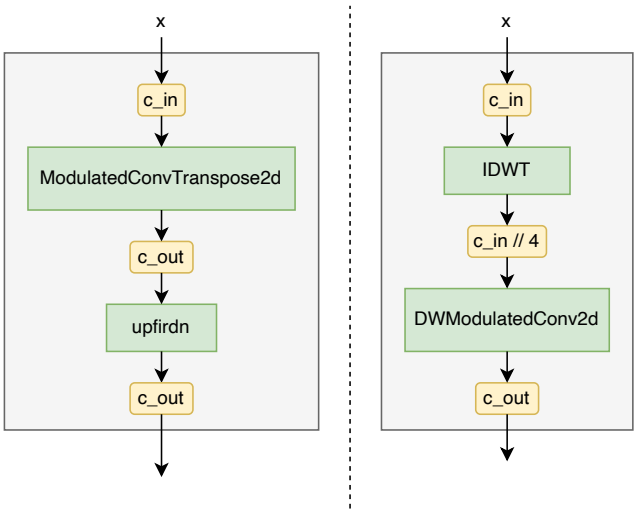

Figure 5. (left) StyleGAN2 upscale block. (right) MobileStyleGAN upscale block. Yellow rectangles on arrows show the number of channels in related feature maps.

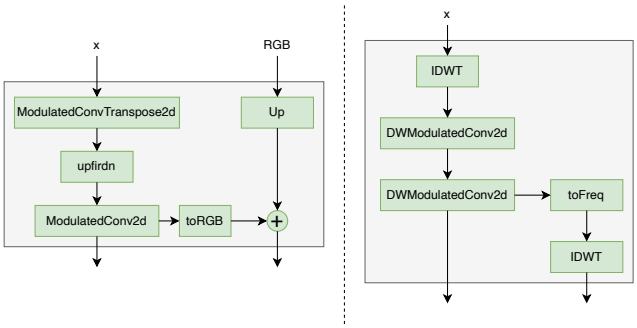

Figure 6. (left) StyleGAN2 building block. (right) MobileStyleGAN building block.

work is illustrated in Figure 7. In this section, we discuss the main parts of our training framework.

4.1. Data preparation

Given the original StyleGAN2 generator, we can transform unpaired learning into the paired setting. To do so, we prepare triplet data $\{style,\ noise,\ I_{teacher}\}$, where $style$ is the output of the mapping network for given noise vector z, $noise$ is the noise shared between the teacher and student networks, and $I_{teacher}$ is the output of the teacher network for the given style.

As described in Section 3.2, each block of MobileStyleGAN predicts the output image for its spatial size. Therefore, instead of $I_{teacher}$, we use $I_{teacher}^{pyramid}$ as a ground-truth. $I_{teacher}^{pyramid}$ is the image pyramid built from $I_{teacher}$. Accordingly, our trained data are represented as the triplet data $\{style,\ noise,\ I_{teacher}^{pyramid}\}$.

To prevent overfitting, we do not use a preprocessed dataset. Instead, we generate data on the fly during a learning procedure.

To decrease memory consumption during the learning procedure, we use only artificial samples generated by StyleGAN2 and no real data.

4.2. Training objective

We now elaborate on the proposed objectives of the knowledge distillation.

Pixel-Level Distillation Loss (Figure 8). Since MobileStyleGAN aims to predict the target image in the wavelet domain, the naive method to mimic the functionality of StyleGAN2 minimizes the pixel-level distance between the wavelet transform of the image generated by StyleGAN2 and the output of MobileStyleGAN. Also, we added a regularization term that minimizes the pixel-level distance between our ground truth and the predicted image in the pixel-based domain. We found that this term allows us to train different frequencies in concert with each other. As described in Section 3.2, our network predicts output images at each spatial size. Accordingly, pixel-based distillation loss was applied at each scale.

Formally, let

$$L_{pix}(F_s, I_t) = \sum_i \left(||F_s^i - DWT(I_t^i)||_1 + ||IDWT(F_s^i) - I_t^i||_1\right) \tag{9}$$

where F_s^i is the image in the wavelet domain predicted by the i-th block of the student network, I_t^i is the ground-truth image in the pixel-based domain in the corresponding spatial size.

Perceptual Loss. The pixel-level loss described previously does not capture perceptual differences between output and ground-truth images. To solve this problem, we use perceptual loss as an objective. Our perceptual loss is based on VGG16 features and implemented as described in [18]. We applied perceptual loss only to the output image generated by MobileStyleGAN.

Formally, let

$$L_{perc}(I_s, I_t) = \sum_l ||VGG16(I_s^{256\text{x}256})_l - VGG16(I_t^{256\text{x}256})_l||_2 \tag{10}$$

where $VGG16(...)_l$ are intermediate features from the corresponding layer $l \in [relu1\_2,\ relu2\_2,\ relu3\_3,\ relu4\_3]$ of the VGG16, $I_s^{256\text{x}256}$ is the output image predicted by the student network (resized to 256x256), where $I_t^{256\text{x}256}$ is the output image predicted by the teacher network (resized to 256x256).

GAN Loss. Using only pixel-level and perceptual losses leads to generation of blurred images. To sharpen the generated images, we incorporate a discriminator network in our pipeline. We adopt the GAN loss for the generator:

$$L_g(style, noise) = BCE(D_T(G_{student}(style, noise)), 1) \tag{11}$$

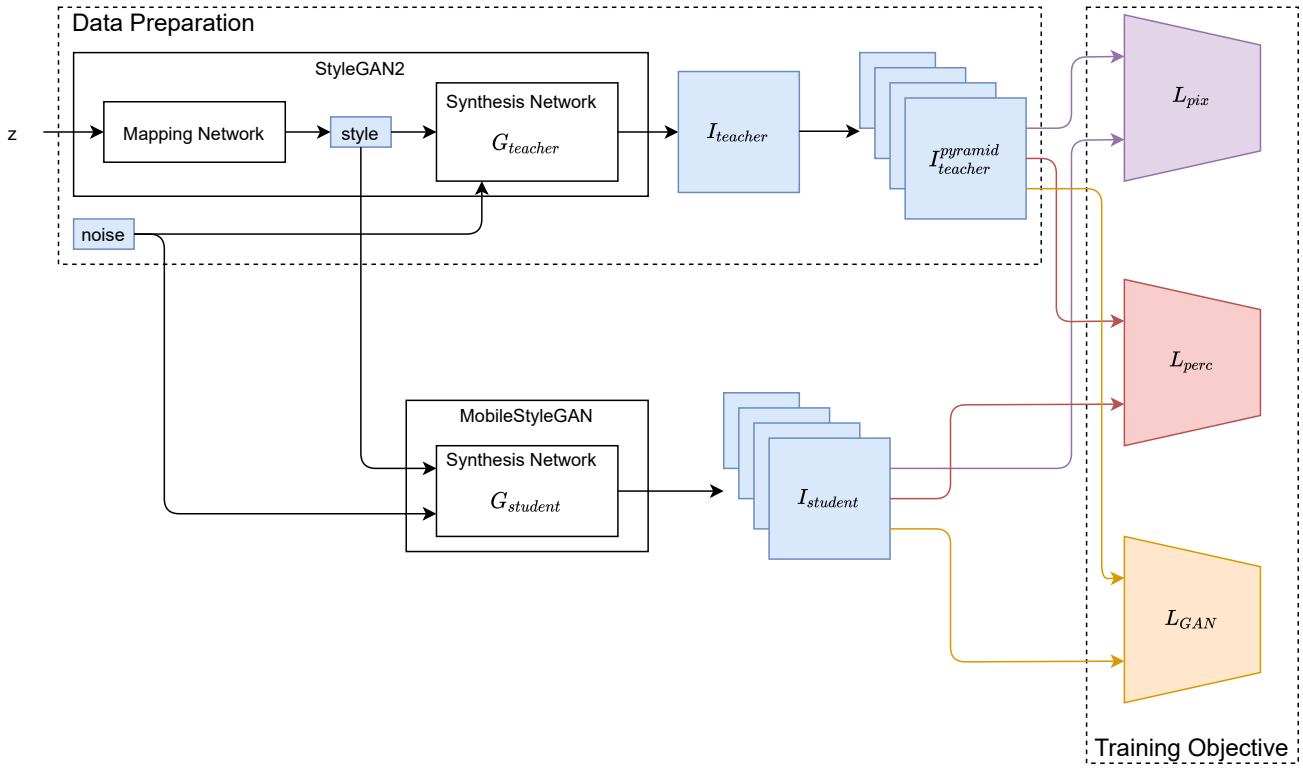

Figure 7. Training framework.

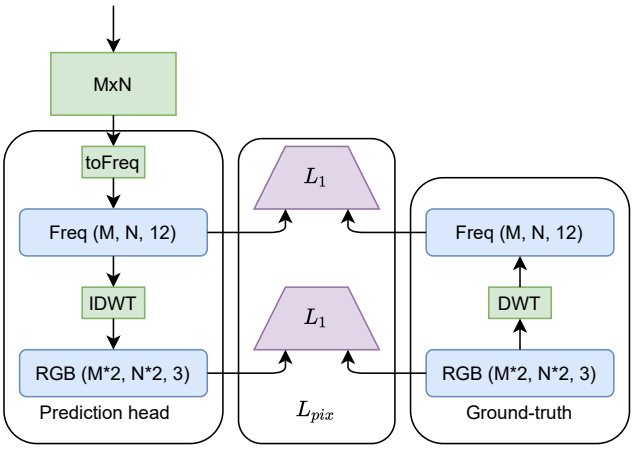

Figure 8. Pixel-Level Distillation Loss.

and for the discriminator network:

$$L_d(style, noise) = BCE(D_T(G_{teacher}(style, noise)), 1) + BCE(D_T(G_{student}(style, noise)), 0) \quad (12)$$

where BCE is the binary cross-entropy loss, $D_T(...)$ is the discriminator network with differentiable augmentations, $G_{student}(...)/G_{teacher}(...)$ is the student/teacher network, and (style, noise) is the input paired data. The discriminator has the same topology as in [20].

Full objective. Finally, we define the full objective as:

$$\begin{aligned} L_{student} &= \lambda_1 * L_{pix} + \lambda_2 * L_{perc} + \lambda_3 * L_g \\ L_{discriminator} &= L_d \end{aligned} \quad (13)$$

where hyperparameters $\lambda_1,\ \lambda_2,\ \lambda_3$ control the importance of each term.

5. Experiments

To train our MobileStyleGAN network we use the StyleGAN2 teacher network trained on the FFHQ [20] dataset.

5.1. Training

MobileStyleGAN is trained using the Adam algorithm [22] with $\beta 1 = 0.9,\ \beta 2 = 0.999$. The generator and discriminator learning rates are both set to constant 5e-4. We use affine transforms and cutout as differentiable augmentations at the input of the discriminator. Hyperparameters of the objective function set are fixed at $\lambda_1 = 1.0,\ \lambda_2 = 1.0,\ \lambda_3 = 0.1$. We update the generator and discriminator at each optimization step. Training takes about three days on 4 x NVIDIA 2080Ti GPU with batch_size=8.

Network	MParams	GMACs	FID
StyleGAN2	28.27	143.15	2.84
MobileStyleGAN	8.01	15.09	12.38

Table 1. Main results. Comparison between StyleGAN and MobileStyleGAN for the FFHQ dataset, 1024x1024.

Network	Engine	Time (sec.)
StyleGAN2	PyTorch	4.3
MobileStyleGAN	PyTorch	1.2
MobileStyleGAN	OpenVINO	0.16

Table 2. Inference time on CPU (Intel(R) Core(TM) i5-8279U).

5.2. Results

Table 1 shows the result of our evaluation of MobileStyleGAN on the FFHQ datase. We compare the number of parameters, the computational cost and the Frechet inception distance (FID) [12] of MobileStyleGAN and the teacher network (StyleGAN2).

Also, we evaluate inference time for StyleGAN2 and MobileStyleGAN on CPU. We use a laptop with Intel(R) Core(TM) i5-8279U CPU for our experiments. Table 2 shows the result of inference time evaluation. As we noticed previously, some architectural tricks that we used in MobileStyleGAN make available graph optimizations such as constant folding, etc. We get better performance when we use special inference engines such as OpenVINO [1] that automatically apply graph optimization.

6. Conclusion

In this work, we approached the problem of high-fidelity image synthesis, suitable for deployment on edge devices. We proposed the new style-based lightweight generative network and training pipeline based on knowledge distillation. We have made our training code publicly available (`https://github.com/bes-dev/MobileStyleGAN.pytorch`), along with a simple python library for fast random face synthesis on the CPU (`https://github.com/bes-dev/random_face`). The accompanying videos can be found on YouTube (`https://www.youtube.com/playlist?list=PLstKhmdpWBtwsvq_27ALmPbf_mBLmk0uI`).

Some techniques may further improve performance and accuracy, such as quantization and pruning. We left them for future research.

References

[1] Openvino toolkit. `https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html`.

[2] Angeline Aguinaldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. Compressing gans using knowledge distillation, 2019.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019.

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment, 2020.

[5] Ting-Yun Chang and Chi-Jen Lu. Tinygan: Distilling biggan for conditional image generation, 2020.

[6] Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. Autogan-distiller: Searching to compress generative adversarial networks, 2020.

[7] Shin Fujieda, Kohei Takayama, and Toshiya Hachisuka. Wavelet convolutional neural networks for texture classification, 2017.

[8] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design, 2018.

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

[10] A. Haar. On the theory of orthogonal function systems. *) * *) (first communication.). 1909.

[11] Seungwook Han, Akash Srivastava, Cole Hurwitz, Prasanna Sattigeri, and David D. Cox. not-so-biggan: Generating high-fidelity images on small compute with wavelet-based super-resolution, 2020.

[12] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[16] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization, 2017.

[17] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

[18] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[19] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Proc. NeurIPS*, 2020.

[20] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[21] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. CVPR*, 2020.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[23] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. Gan compression: Efficient architectures for interactive conditional gans, 2020.

[24] Bingchen Liu, Yizhe Zhu, Kunpeng Song, and Ahmed Elgammal. Towards faster and stabilized gan training for high-fidelity few-shot image synthesis, 2021.

[25] Pengju Liu, Hongzhi Zhang, Kai Zhang, Liang Lin, and Wangmeng Zuo. Multi-level wavelet-cnn for image restoration, 2018.

[26] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018.

[27] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[28] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[29] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[30] Haotao Wang, Shupeng Gui, Haichuan Yang, Ji Liu, and Zhangyang Wang. Gan slimming: All-in-one gan compression by a unified optimization framework, 2020.

[31] Qi Zhang, Huafeng Wang, and Sichen Yang. Image super-resolution using a wavelet-based generative adversarial network, 2019.

[32] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017.