STARGAN-V2 COMPRESSION USING KNOWLEDGE DISTILLATION

Paras Kapoor

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montréal, Québec, Canada

March 2021

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Paras Kapoor**

Entitled: **StarGAN-v2 compression using knowledge distillation**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Charalambos Poullis

_____ Examiner
Dr. Yiming Xiao

_____ Examiner
Dr. Charalambos Poullis

_____ Thesis Supervisor
Dr. Tien D. Bui

Approved by _____
Dr. Leila Kosseim, Graduate Program Director

March 31, 2021 _____
Dr. Mourad Debbabi, Interim Dean
Gina Cody School of Engineering and Computer Science

Abstract

StarGAN-v2 compression using knowledge distillation

Paras Kapoor

Image-to-image translation is used in a broad variety of machine vision and computer graphics applications. These involve mapping grey-scale images to RGB images, deblurring of images, style transfer, transfiguring objects, to name a couple. In addressing complex image-to-image translation issues, Generative Adversarial Networks (GANs) are at the forefront. StarGAN-v2 is a state of the art method for multi-modal multi-domain image-to-image translation that produces different images from a single input image over multiple domains. However, at a parameter count of more than 50M, StarGAN-v2 has a computation bottleneck and consumes more than 60G MACs (Multiply-Accumulate Operations to calculate computation expense, 1 MAC = 2 FLOPs) to create one 256×256 image, preventing its widespread adoption. This thesis focuses on the task of compressing StarGAN-v2 using knowledge distillation. Using depthwise separable convolutional layers and reduced channels for intermediate layers, we develop efficient architectures for different StarGAN-v2 modules. In a GAN mini-max optimization environment, the efficient networks are trained with a combination of different distillation losses along with the original objective of StarGAN-v2. Without losing image quality, we reduce the size of the original framework by more than $20\times$ and the computation requirement by more than $5\times$. The feasibility of the proposed approach is demonstrated by experiments on CelebA-HQ and AFHQ datasets.

Acknowledgments

First of all, I would like to express my deepest appreciation to my supervisor, Dr. Tien D. Bui for leading me into the area of deep learning and image processing. He helped me pick the courses needed to build a strong basis for my deep learning and computer vision skills. He encouraged me to study the area of compression of deep learning and coordinated my research work. I truly appreciate his support during the course of my master's degree. In addition, I want to thank all people who have helped me with my study, research and thesis, including every member of my lab for their constructive feedback. Next, I would like to thank the instructors of my courses who taught me useful skills and knowledge, essential for my research work. Last but not least, I have to thank my parents for their sacrificial love and unceasing encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives an introduction to our research work. The purpose of our research is defined in section 1.1. Section 1.2 explains our contribution in the research. Finally, the framework of the thesis is described in section 1.3.

1.1 Problem Description

Over a couple of years, applications of Generative Adversarial Networks (GANs) [5], especially image-to-image translation, have seen incredible development. As the abundance of datasets for high-fidelity natural image synthesis has increased, deeper and complex neural networks have been proposed. However, the growth in the scale of the neural network and computational specifications, also make them difficult to use on embedded computers. StarGAN-v2 [6] is one such state of the art technique that has allowed different images to be translated across numerous domains in a single framework. Nonetheless, at a parameter count of more than 50M, StarGAN-v2 has a computational bottleneck, consuming more than 60G MACs to generate one 256×256 image, preventing its widespread adoption.

1.2 Contributions

The goal of this work is to reduce the size and computation requirements of StarGAN-v2 [6] leading to its widespread adoption to edge devices. Our contributions to the research

work in this thesis are as follows:

- We design efficient network modules of StarGAN-v2 using depthwise separable convolutional [7] layers.
- We develop a combination of various knowledge distillation [8] losses operating on different modules of StarGAN-v2 to be used along with the original objective in a GAN minimax optimization [5] setting.
- Without losing image quality, we reduce the size of the original StarGAN-v2 framework by more than 20× and the computation requirement by more than 5× on benchmark datasets.

1.3 Thesis Structure

The thesis is organized as follows. As needed to follow the rest of the work, Chapter 2 includes an introduction to CNNs [9], GANs [5], and StarGAN-v2 [6]. A review of recent work in the field of GAN compression and the motivation for our research can be found in the chapter 3. Chapter 4 offers a detailed explanation from a mathematical viewpoint of our proposed method of compressing StarGAN-v2 [6]. Chapter 5 presents the findings of different experiments carried out to investigate the network power design space and determine the efficiency of the image generation of our proposed system. Finally, the chapter 6 outlines our core observations and suggests potential future avenues for study. This thesis is intended for a reader with an understanding of GANs and compression of neural networks in general.

Chapter 2

Background

This chapter provides the requisite context to understand the remainder of the thesis. A few important mathematical concepts are detailed in section 2.1. Section 2.2 provides a brief introduction to convolution neural networks and different layers used to construct complex networks. Section 2.3 explains generative adversarial networks and minimax objective [5]. Section 2.4 describes all StarGAN-v2 [6] system components. Finally, Section 2.5 defines the necessary criteria for scoring the quality and diversity of images produced.

2.1 Mathematical Concepts

A few mathematical concepts for general understanding of the thesis are explained in the following sub-sections.

2.1.1 Vector Norms

A vector norm is used to calculate the size or length of a vector $\mathbf{x}$ of dimension N. In machine learning, norms are essential to formulate loss functions. L_1 and L_2 are the two most popular norm functions used to calculate magnitude of a vector. L_1 norm calculates the sum of absolute values of a vector as shown in Eq. 1. L_2 norm is calculated as the square root of the sum of the squared vector values as shown in Eq. 2.

$$L_1(\mathbf{x}) = \sum_{i=1}^{N} |x_i| \tag{1}$$

$$L_2(\mathbf{x}) = \left(\sum_{i=1}^{N} x_i^2 \right)^{\frac{1}{2}} \tag{2}$$

2.2 Convolutional Neural Network (CNNs)

Convolutional neural networks (CNNs) [9] are networks specialized in analyzing data with a grid-like topology, and are thus widely used in image recognition tasks. CNNs typically consists of convolution layers, fully connected layers and normalization layers explained in the following sub-sections.

2.2.1 Fully Connected (FC) Layer

Fully connected layers have connections from the previous layer with all the neurons and have connections from the next layer with all the neurons. An example of the 3 feed forward linear layers (input layer, hidden layer and output layer) network is seen in Figure 2.1. In shallow CNN models, FC layers are necessary, to cover the whole spatial dimension of an input image [10].

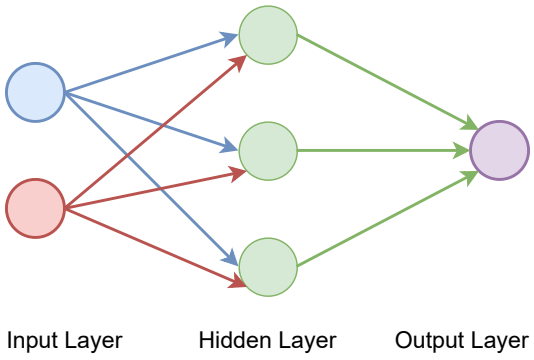


Figure 2.1: A Representation of Fully Connected Layer

2.2.2 Residual Layer

Residual layers [11] are essential to train very deep neural networks. The problem of vanishing gradient with an increasing number of layers is solved by residual connections, allowing information to flow from initial layers to final layers using skip connections as seen in Figure 2.2.

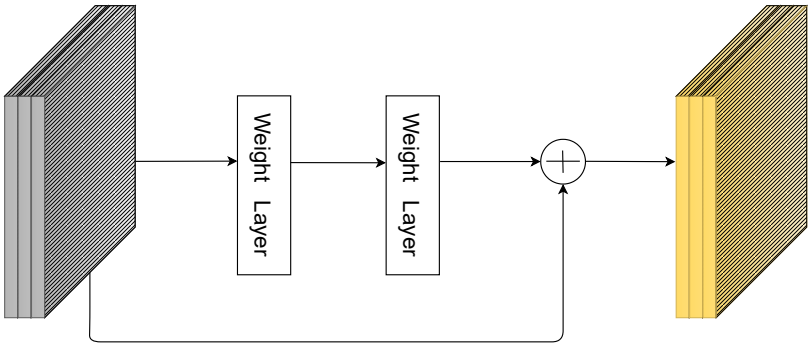


Figure 2.2: An illustration of Residual Layer

2.2.3 Regular Convolutional Layer

Convolutional layers are the core building blocks of CNNs, it helps in feature detection. In order to create an output feature map as seen in Figure 2.3, a standard convolution layer usually takes an input feature map, convolves multiple filters each with the same number of channels as input. Each filter slides over the input data, multiplying the element with the portion of the input it is currently on, and then summing up the input data.

2.2.4 Depthwise Separable and Pointwise Convolutional Layers

In light-weight models, the depthwise separable convolutional [7] and pointwise convolutional [12] layers are used extensively because they require a smaller number of parameters and dot products compared to regular convolutional layers. Each kernel of a filter in a depthwise separable convolutional layer, convolves separately with only 1 channel of the input feature map to give a single output. Pointwise convolutional layers consist of filters of

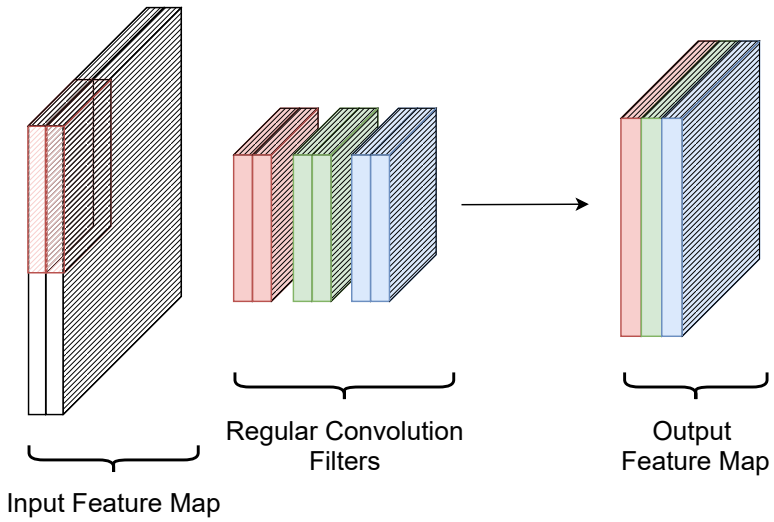


Figure 2.3: An Illustration of a Regular Convolutional Layer

shape 1×1 with depth equal to input feature map. As seen in Figure 2.4, these layers are commonly stacked together to obtain efficiency without significantly damaging performance.

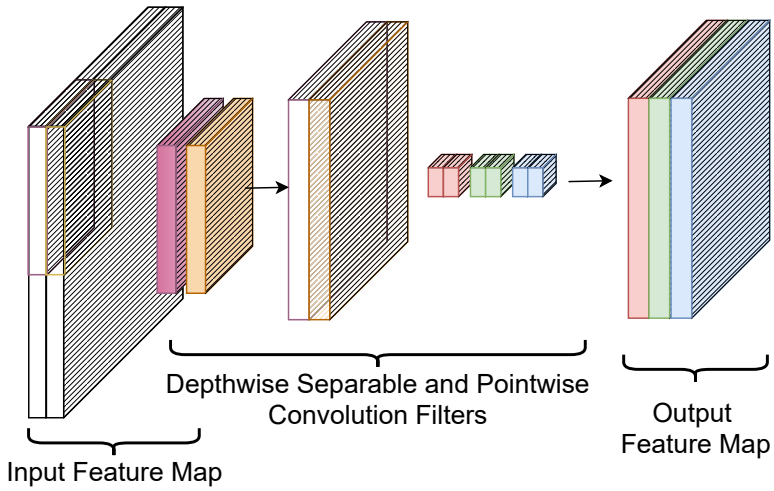
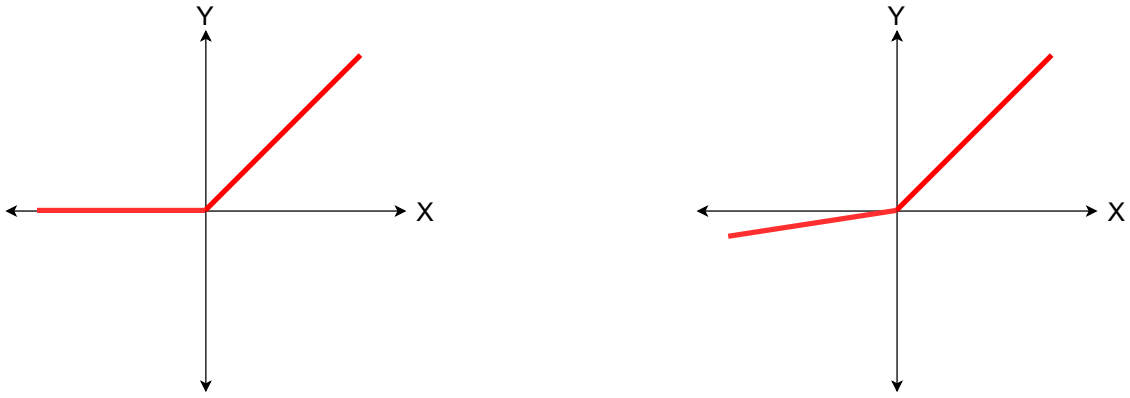


Figure 2.4: An Illustration of Depthwise Separable and Pointwise Convolutional Layers

2.2.5 ReLU and Leaky ReLU Layers

A ReLU [13] layer is composed of rectified linear unit nodes that are piece-wise linear function. For values greater than zero, the function is linear, but negative values are often

stated as zero, as seen in Figure 2.5. When training a neural network using backpropagation, it is a nonlinear function with a lot of the attractive properties of a linear activation function. However, neurons with high negative values are trapped at 0 in the ReLU layers, causing them to effectively die during training. Leaky ReLU [14] is an extention of ReLU with a non-zero gradient for negative input values as shown in Figure 2.5.



ReLU Activation Function Leaky ReLU Activation Function

Figure 2.5: Non-Linear Activation Functions

2.2.6 Instance Normalization (IN) and Adaptive Instance Normalization (AdaIN) Layers

Instance Normalization (IN) [15] is a representative method which was adopted during style transition to discard instance-specific contrast information from an image. IN performs style normalization by normalizing feature statistics, which have been found to hold an image's style details. For each channel and each sample, the mean and variance are separately measured across spatial dimensions. Adaptive instance normalization (AdaIN) [16] is a simple extension to IN. In order to fit the input type statistics, the AdaIN layer aligns the channel-wise mean and variance of the material image. Similar to IN, the statistics in AdaIN are computed across spatial locations.

2.2.7 Bilinear Upsampling Layer

In the context of image processing, upsampling is a technique for increasing the size of an image. When perfect image transformation with pixel matching is difficult, bilinear upsampling is used, so that pixels can be measured and assigned acceptable intensity values. It takes into account the nearest neighbourhood 2×2 of known pixel values surrounding the calculated position of the unknown pixel and then takes a weighted average of these 4 pixels to reach its actual, interpolated value.

2.3 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [5] are a family of generative networks that, through a two-player mini-max game, discover an underlying distribution behind a data generation mechanism. GANs are extensively used in applications involving image synthesis. As seen in Figure 2.6 GANs consists of two network modules a Generator (G) and a Discriminator (D) trained in an adversarial manner. Generator learns to take in a random input z and create a sample indistinguishable from the actual sample distribution. Discriminator is trained to decide whether a given sample came from the generator or from a collection of actual samples.

GANs are conditioned to find a Nash Equilibrium [5] in a minimax optimization setting as seen in Eq. (3).

$$\min_G \max_D [\mathbb{E}_{x \sim p_{data}}[log(D(x))] + \mathbb{E}_{z \sim p_Z}[log(1 - D(G(z)))]] \tag{3}$$

One such balance is where the generator models the actual distribution exactly, so that the discriminator does not do better than 50% percent. This is the balance that is anticipated, but the techniques of GANs also suffer from problems of convergence and mode falling. For a variety of reasons, a GAN may fail to converge during training. As a type of cat-and-mouse game of attempting to find a Nash equilibrium, the generator and discriminator will oscillate. The gradient will be very minimal if the discriminator becomes too good and

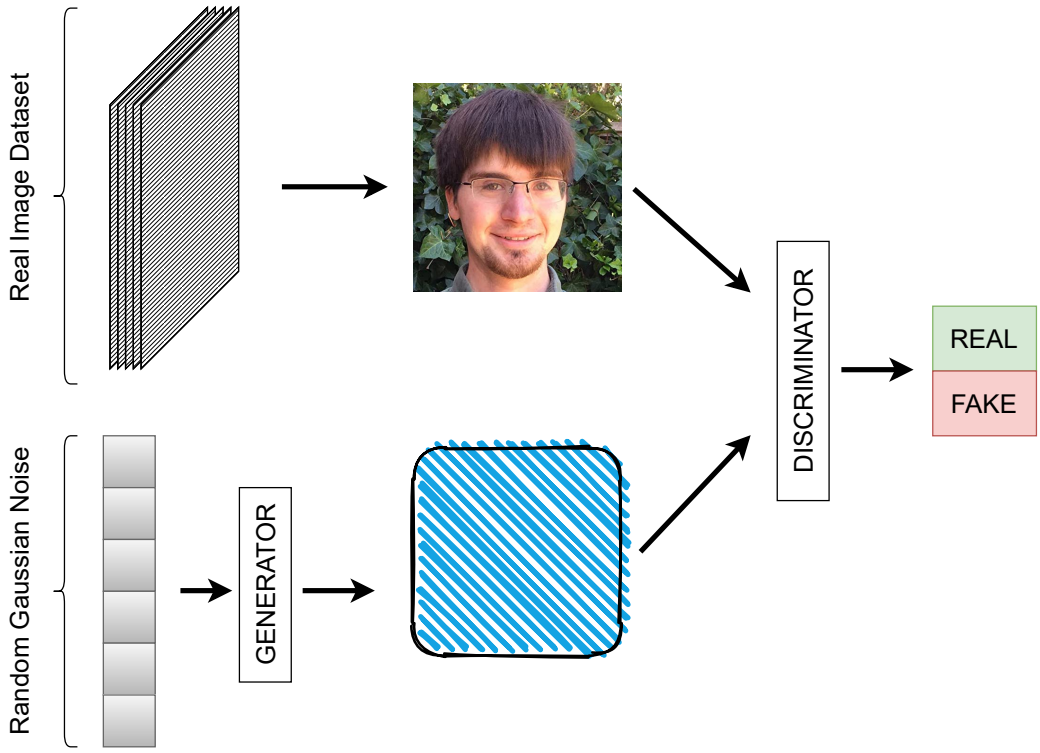


Figure 2.6: Flowchart of Generative Adversarial Network

too optimistic, then virtually little is discovered by the generator. Mode dropping happens when any of the modes of the goal distribution are not generated by the generator. This can occur when the generator overfits a particular mode, but can also occur when the generator gives up attempting to create more complicated examples. To stabilize GANs, different strategies and losses are applied, including a few described in the following sub-sections.

2.3.1 Exponential Moving Average (EMA)

Exponential Moving Average (EMA) [17, 18] increases the stability of GANs training by limiting cycles around the nash equilibrium with vanishing amplitude. The generator parameters are averaged over time outside the training loop as shown in Eq. (4). As β approaches 1, averaging effectively computes longer time windows.

$$\theta_{EMA}^{(t)} = \beta\theta_{EMA}^{(t-1)} + (1-\beta)\theta_{EMA}^{(t)} \tag{4}$$

2.3.2 Non Saturating Adversarial Loss

A succinct equation that intuitively illustrates the adversarial essence of the rivalry between the generator and the discriminator is the mini-max formulation. However, distinct loss functions for the generator and the discriminator are used in operation. A adjustment to the generator loss to solve the saturation issue is the Non-Saturating GAN Loss [5]. It is a slight shift involving the generator maximizing the log of the probabilities of discrimination for produced images instead of minimizing the log of the probabilities of inverted discrimination for generated images, as seen in Eq. (5).

$$\max_{G}[\log(D(G(z)))] \tag{5}$$

where G is the generator, D is the discriminator and z is the input noise.

2.3.3 Zero-Centered Gradient Penalty

A zero-centered gradient penalty [1] is used as a regularization to penalize the discriminator for deviating from the Nash equilibrium and addressing the problem of gradient explosion in discriminators. As seen in Eq. (6), the squared L_2 norm of gradients of the discriminator D with respect real input data x is penalized.

$$R_D(x) = E_{D(x)}[||\nabla D(x)||_2^2] \tag{6}$$

2.4 StarGAN-v2

StarGAN-v2, as seen in Figure 2.7, converts images from one domain to different images of several target domains.

2.4.1 Network Modules

Four network modules form the structure of StarGAN-v2. All the modules are multi-headed but the generator, with the number of heads equal to the number of domains.

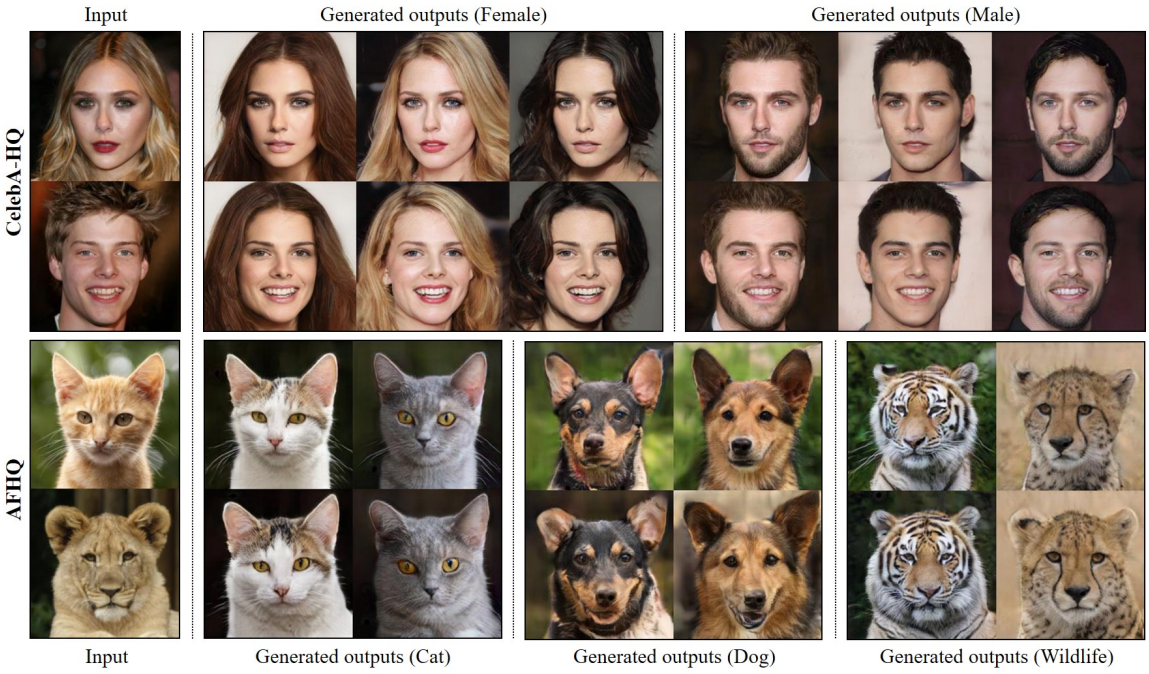


Figure 2.7: Diverse image synthesis using StarGAN-v2. Reprinted from [1].

Generator

Generator transforms an input image to an output image reflecting the input style code of a specific domain as shown in Figure 2.8.

Image Discriminator

For multiple domains seen in Figure 2.9, Image Discriminator recognizes the input image as genuine or fake.

Style Encoder

The style encoder extracts the style code from an input image and its domain label to allow the generator to synthesize reference-guided images, as seen in Figure 2.10.

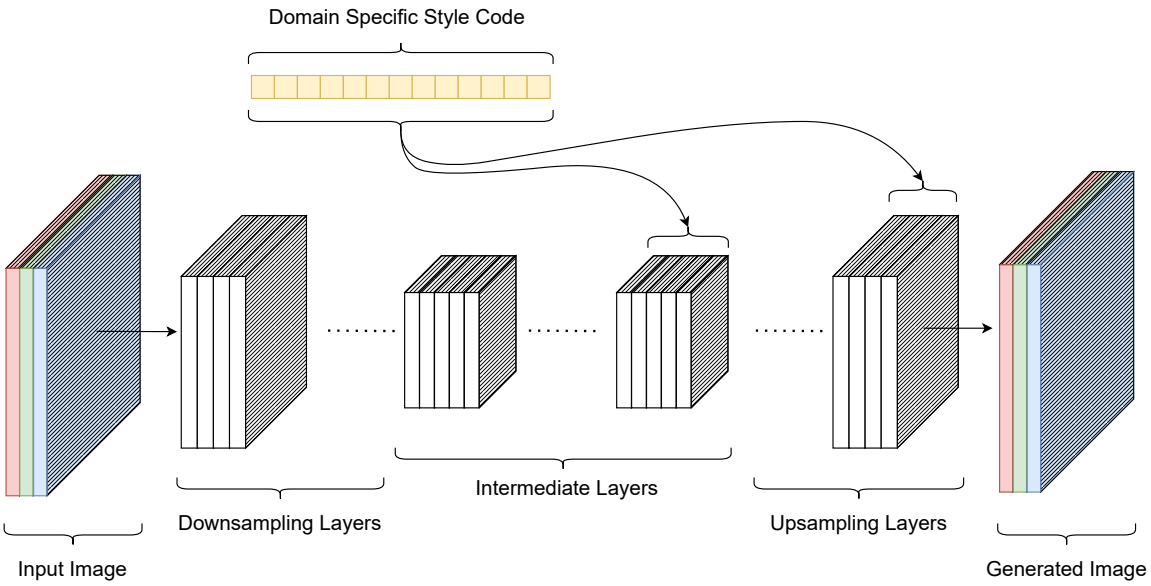


Figure 2.8: Generator Network flowchart

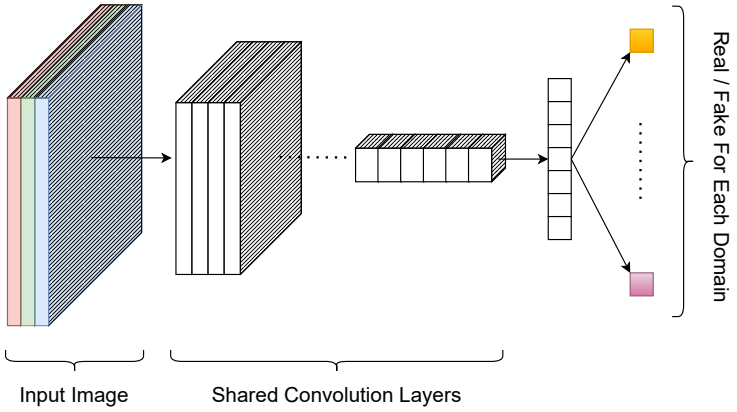


Figure 2.9: Image Discriminator Network flowchart

Mapping Network

Mapping network transforms latent code sampled from random Gaussian noise into multi-domain type codes, enabling the generator to synthesize latent images as seen in Figure 2.11.

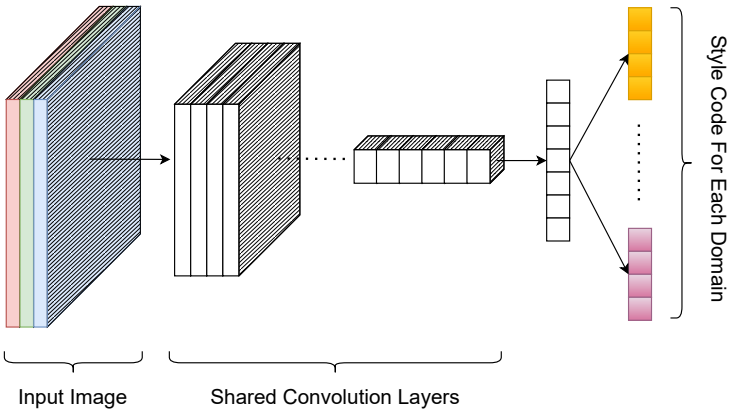


Figure 2.10: Style Encoder Network flowchart

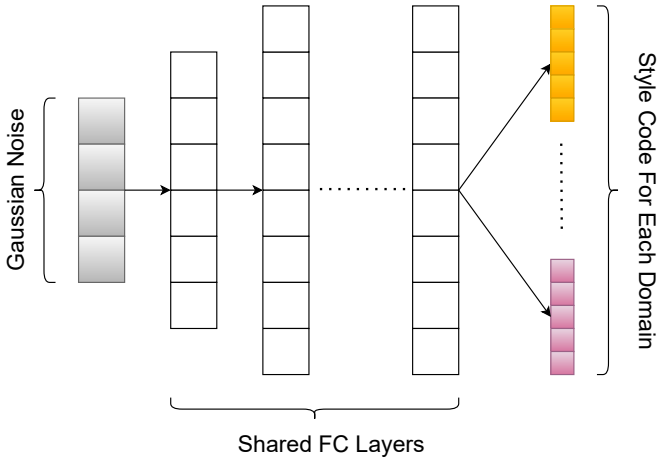


Figure 2.11: Mapping Network flowchart

2.4.2 Image Synthesis

StarGAN-v2 translates input image using reference image as shown in Figure 2.12 and also using latent code as shown in Figure 2.13.

2.4.3 Training Objectives

Let $\mathcal{X}$, $\mathcal{Y}$ be the sets of images and possible domains and $\mathcal{Z}$ be the set of all possible latent codes. Denoting the generator as G, style encoder as E, mapping network as F, and image discriminator as D. During training, an image $x \in \mathcal{X}$ with its original domain label $y \in \mathcal{Y}$, two latent codes $z_1 \in \mathcal{Z}$, $z_2 \in \mathcal{Z}$, and a target reference image $\tilde{x} \in \mathcal{X}$ with its target domain

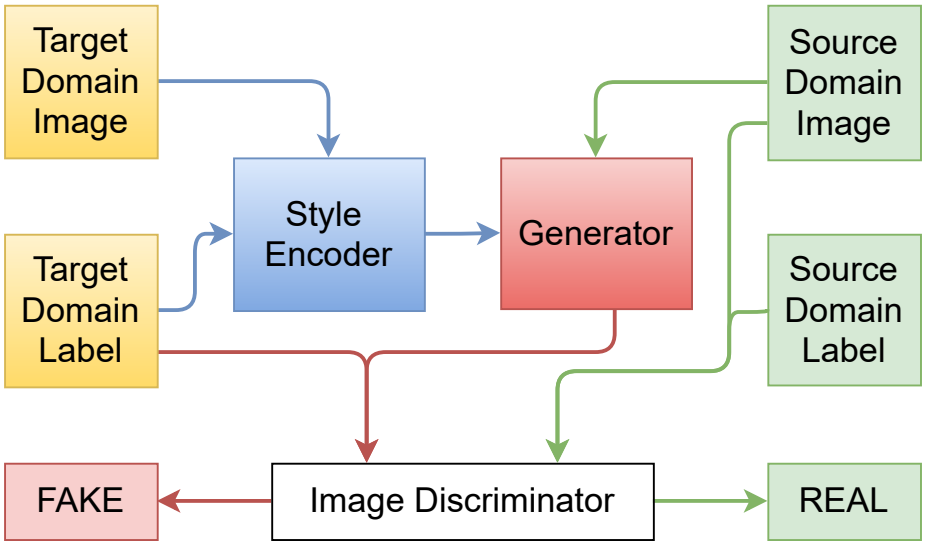


Figure 2.12: Flowchart of reference-guided image synthesis by generator and style encoder

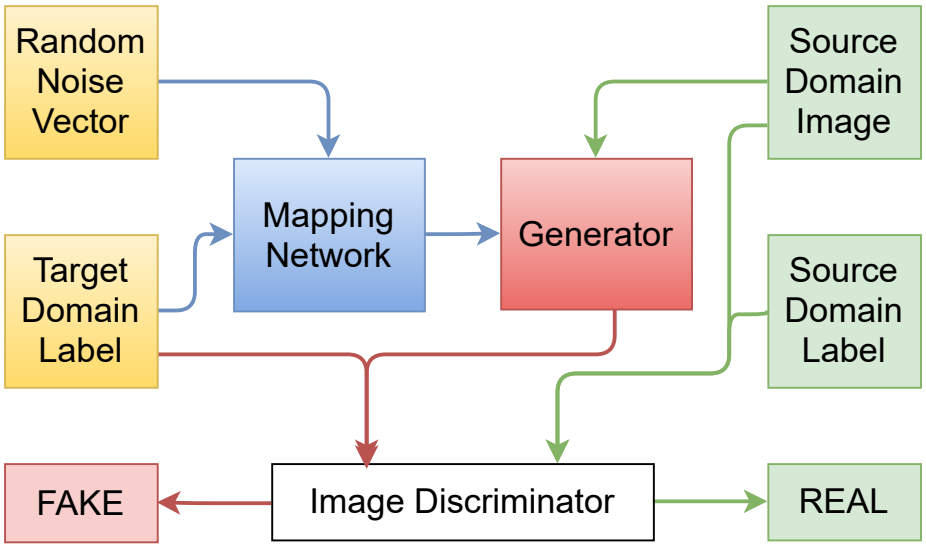


Figure 2.13: Flowchart of latent-guided image synthesis by generator and mapping network

label $\tilde{y} \in \mathcal{Y}$ are sampled in each iteration. Following target style codes are generated for both latent-guided and reference-guided synthesis.

$$s_{z_1} = F(z_1, \tilde{y}) \qquad s_{z_2} = F(z_2, y) \qquad s_{\tilde{x}} = E(\tilde{x}, \tilde{y}) \qquad s_x = E(x, y)$$

The framework is trained using the following objectives:

Adversarial Objective

Input image x and style codes $\tilde{s} \in s_{z_1}, s_{\tilde{x}}$ are taken by the generator to produce images in the target domain. The image discriminator D learns to determine the difference between their respective domain's actual and generated images as seen in Eq. (7). The mapping network and type encoder learn to have the style code $\tilde{s}$ that is likely to be found in the target domain $\tilde{y}$, and G learns to use and create photos that are indistinguishable from actual domain images $\tilde{y}$.

$$\mathcal{L}_{adv} = \mathbb{E}_{x,y}[\log(D_y(x))] + \mathbb{E}_{x,\tilde{y},z}[log(1 - D_{\tilde{y}}(G(x, \tilde{s})))] \tag{7}$$

Style reconstruction

Style reconstruction loss similar to the previous approaches [19, 20], forces the generator G to utilize the style code $\tilde{s} \in s_{z_1}, s_{\tilde{x}}$ when generating the image. It is an L_1 loss as seen in Eq. (8). At test time, the learned encoder E allows G to transform an input image, reflecting the style of a reference image.

$$\mathcal{L}_{sty} = \mathbb{E}_{x,\tilde{y},z}[||\tilde{s} - E_{\tilde{y}}(G(x, \tilde{s}))||_1] \tag{8}$$

Style diversification

The generator G is regularized with diversity sensitive loss [21, 20] to enable generator to produce diverse images. The style codes to be used are produced by mapping network F from two sampled latent codes z_1 and z_2. Maximizing the regularization term forces G, via an L_1 loss as seen in Eq. (9) to explore the image space and discover meaningful style features to generate diverse images.

$$\mathcal{L}_{ds} = \mathbb{E}_{x,\tilde{y},z_1,z_2}[||G(x, s_{z_1}) - G(x, s_{z_2})||_1] \tag{9}$$

Preserving source characteristics

The cycle consistency loss [22, 23, 24] is used to ensure that the generated image properly maintains the domain invariant (e.g. pose) characteristics of the source image via an L_1 loss as seen in Eq. (10). The generator G learns to maintain the original characteristics of the input image x by modifying its form diligently by allowing the generator G to recreate x with the approximate style code s_x.

$$\mathcal{L}_{cyc} = \mathbb{E}_{x,y,\tilde{y},z}[||x - G(G(x, \tilde{s}), s_x)||_1] \tag{10}$$

Full objective

The full objective of StarGAN-v2 is summarized in Eq. (12).

$$\mathcal{L}_{org} = \mathcal{L}_{adv} + \lambda_{sty}\mathcal{L}_{sty} - \lambda_{ds}\mathcal{L}_{ds} + \lambda_{cyc}\mathcal{L}_{cyc} \tag{11}$$

$$\implies \min_{G,E,F} \max_{D} [\mathcal{L}_{org}] \tag{12}$$

where λ_{sty}, λ_{ds}, and λ_{cyc} are hyperparameters for each loss term.

2.5 Metrics

Quantitative metrics are essentials to objectively measure the quality and diversity of images. Two important metrics are explained in the following sub-sections.

2.5.1 Frećhet Inception Distance (FID)

Frećhet Inception Distance (FID) [25] measures the discrepancy between two sets of images. A lower FID indicates better quality images. Conversely, a higher score indicates a lower quality image and the relationship may be linear. Assuming the feature embeddings follow a multidimensional Gaussian distribution, mean and covariance matrices are obtained for

both real (m_r, C_r) and synthetic data (m, C). FID score is calculated using the Eq. (13) where $Tr()$ refers to sum of diagonal elements of a matrix.

$$FID = ||m - m_r||_2^2 + Tr(C + C_r - 2(CC_r)^{\frac{1}{2}}) \tag{13}$$

2.5.2 Learned Perceptual Image Patch Similarity (LPIPS)

Learned similarity of perceptual image patch (LPIPS) [2] tests the diversity of images produced using the L_1 distance between features extracted from AlexNet [26] pretrained from ImageNet. In the channel dimension, deep embedding of each image patch is normalised and then averaged over the spatial dimension and over all layers. Higher distance means more diverse images and lower means similar images.

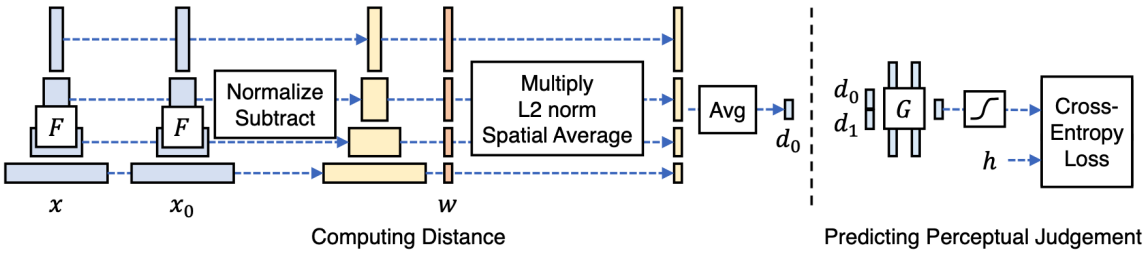


Figure 2.14: Calculation of Learned Perceptual Image Patch Similarity. Reprinted from [2].

2.6 Summary

In this chapter, the components that build up a convolution neural network, mechanics of GANs particularly StarGAN-v2, and metrics to objectively evaluate GANs were discussed. In the next chapter, recent techniques developed to compress GANs and the motivation of our research work will be explained.

Chapter 3

Related Work

This segment addresses recent work undertaken to compress and speed-up GANs. The section 3.1 offers information on important techniques required by recent compression techniques for GANs. Section 3.2 describes NAS and KD based compression methods combined into a single target. The section 3.3 explains strategies that use pruning and KD to achieve successful networks. Only techniques based on KD are stated in the section 3.4. Finally, the section 3.5 presents descriptions of the limitations of the present research work for our objective.

3.1 Prerequisite Techniques

Following are some techniques essential to understand recently developed methods for compressing GANs:

3.1.1 Knowledge Distillation (KD)

Knowledge distillation (KD) [8] is a way of transmitting information that allows a single network with a relatively limited number of parameters to benefit from a set of networks or a network with a large number of parameters. The larger model is known as the instructor and the smaller model is known as the pupil. The aim of economically deploying a valid model is accomplished through training a large data model, leveraging its better capacity

to acquire succinct representations of information, and then distilling that knowledge into the smaller model, which would not be able to absorb it on its own, through training it to learn the large model's soft performance. A student can learn to adapt the instructor to every activation layer.

3.1.2 Neural Architecture Search (NAS)

Neural architecture search (NAS) [27] is a technique for automating complex deep neural network design. The networks designed by NAS are on par or outperform hand-designed architectures. NAS is closely related to hyperparameter optimization and is a subfield of automated machine learning (AutoML) [28]. Strategy to steps for NAS can be categorized according to the search space, search strategy and performance estimation strategy used:

(1) The search space defines all the potential architectures.

(2) The search strategy defines the approach to explore the search space.

(3) The performance estimation strategy evaluates the performance of a possible architecture from its design without constructing and training it.

3.1.3 Neural Network Pruning

The process of decreasing the size of a network by deleting parameters is neural network pruning [29]. In this algorithm, the network is first trained to convergence. Afterwards, each parameter or structural element in the network is issued a score, and the network is pruned based on these scores. Pruning lowers the network's accuracy, so it is trained to recover better (known as fine-tuning). The pruning and fine-tuning method is also iterated several times, progressively decreasing the size of the network. There are two types of pruning strategies, structured pruning and unstructured pruning. Unstructured pruning generates a sparse neural network, by setting individual weights to zero, whereas structured pruning eliminates entire neurons, filters, or channels.

3.2 NAS and KD Based Techniques

Following methods use KD and NAS together to compress GANs:

3.2.1 GAN Compression

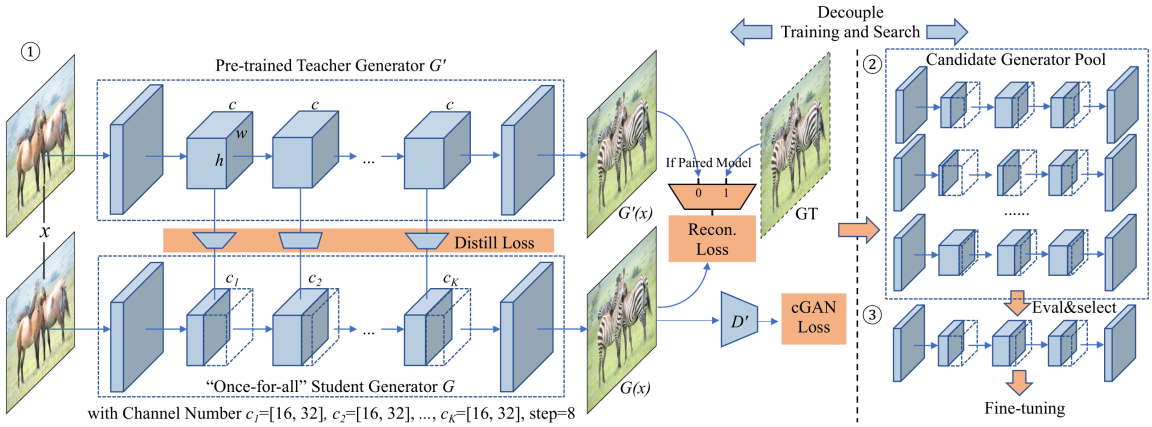


Figure 3.1: A figure demonstrating GAN Compression method. Reprinted from [3].

A general-purpose tool for compressing conditional GANs is GAN Compression [3]. It decreases the computation of regularly used conditional GAN models like pix2pix [30], CycleGAN [24], and GauGAN [31] by $9-21\times$ while retaining visual fidelity. It utilizes neural architecture search (NAS) to obtain a compressed model through weight sharing and knowledge distillation (KD) of intermediate representations of the original model. Following are the steps involved:

(1) A smaller once-for-all [32] student generator G is distilled with a pre-trained instructor generator $\hat{G}$, comprising all feasible channel numbers by weight sharing.

(2) At each training phase, different channel numbers for G are sampled such that all channel numbers can be served by one generator.

(3) Many sub-generators are isolated from the once-for-all generator with various channel numbers and their output is measured.

(4) Finally, given the compression ratio target and output target (FID or mAP) and fine-tuned to achieve the final compressed model, the best sub-generator is selected.

3.2.2 AutoGAN Distiller

AutoGAN Distiller (AGD) [33] is a differentiable NAS and KD-based system that customizes the search space for tasks with various properties. It searches for categories of operators in addition to channel widths, and tests a super-resolution task in addition. Through knowledge distillation, the search is driven by the original GAN model, thus satisfying the compression. AGD is fully autonomous, standalone (i.e. no qualified discriminators are required) and broadly applicable to different models of GAN.

Operator search is performed for the following operators:

- Conv 1×1, Conv 3×3
- Residual Block ("ResBlock") (2 layers of Conv 3×3, with a skip connection)
- Depthwise Block ("DwsBlock") (Conv 1×1 + DepthwiseConv 3×3 + Conv 1×1, with a skip connection)

The operator for each layer is searched in a differentiable manner [34, 35]. The design parameter α_i is used for the i-th layer to decide the operator for the current layer, and the α_i softmax value denotes the chance of selecting this operator. For any iteration of the scanning process, all candidate operators are activated and the output is the weighted sum of all operators calculated by all α softmax values.

To create a slimmer generator, width search is performed. Each kernel of convolution with a maximum width is called a superkernel. The scan is then done for the ϕ expansion ratio to use just a subset of the superkernel input/output dimensions. $\phi \in [\frac{1}{3}, \frac{1}{2}, \frac{3}{4}, \frac{5}{6}, 1]$ and the architecture parameter γ_i controls the probability of choosing each expansion ratio in the i-th layer. For estimated differential sampling of ϕ and γi, Gumbel-Softmax is used. Each time, only one most likely expansion ratio is enabled during the searching process, which saves both memory and computing costs. The training objective of AGD framework is shown in Eq. (14).

$$\min_{G,\alpha,\gamma} \frac{1}{N} \sum_{i=1}^{N} d(G(x_i, \alpha, \gamma), G_0(x_i)) + \lambda F(\alpha, \gamma) \tag{14}$$

Here, $d(.,.)$ is a distance metric for the knowledge distillation between the compact generator G and the pre-trained one G_0, F is the computational budget determined by the network architecture, α and γ are the architecture parameters controlling the operator and width of each layer, respectively, and λ is the trade-off parameter. Note that both $d(.,.)$ and F are functions of α and γ.

3.3 Neural Network Pruning and KD Based Techniques

Following methods use KD and neural network pruning together to compress GANs:

3.3.1 GAN Slimming (GS)

GAN Slimming (GS) [36] is a coherent optimization system combining the distillation of information, channel pruning, and quantization for GAN compression. With limited visual output loss, GS compresses CartoonGAN [37], a state-of-the-art style transfer network, by up to 47×.

$$\min_{G} \max_{D} L_{GAN} \tag{15}$$

$$L_{GAN} = \mathbb{E}_{y \in Y}[log(D(y))] + \mathbb{E}_{x \in X}[1 - log(D(G(x)))] \tag{16}$$

$$L_{dist}(W, \gamma) = \mathbb{E}[d(G(x; W, \gamma), G_0(x))] \tag{17}$$

$$L_{CP}(\gamma) = ||\gamma||_1 \tag{18}$$

GS integrates the typical minimax optimization question in GAN where D is the discriminator jointly trained by minimax optimization with efficient generator G as shown in Eqs.

(15) and (16). A model distillation loss term L_{dist} as shown in Eq. (17) is used to enforce the small generator G to mimic the behaviour of original large generator G_0 where $d(.,.)$ is Perceptual loss [38]. G is slimmed from G_0, through channel pruning and quantization. For channel pruning, approach similar to [39] with L_1 norm on the trainable scale parameters in the normalization layers to encourage channel sparsity as shown in Eq. (18). By default, both activation and kernel weights are quantized uniformly to 8-bit (i.e., m = n = 8).

3.3.2 Co-Evolutionary Compression

Co-evolutionary compression for unpaired image translation [4] obtains compressed generator with less than $\frac{1}{4}$ parameters compared to the original one while maintaining the image translation performance. In fact, generators are encoded as two populations for two image domains and optimized to iteratively investigate the most relevant convolution filters, as seen in Figure 3.2. Fitness of each individual is calculated using the number of parameters, a discriminator-aware regularization in Eq. (19), and the cycle consistency in Eq. (20) into a joint loss as shown in Eq. (21). Where G_1 and G_2 are the compressed generators, D_1 is the discriminator in the original network, $||\quad||_2$ is L_2 norm and $\mathcal{N}()$ is the total number of parameters.

$$L_{DisA} = \frac{1}{m}\sum_{i=1}^{m} ||D_1(G_1(x_i)) - D_1(\hat{G}_1(x_i))||_2^2 \tag{19}$$

$$L_{cyc} = \frac{1}{m}\sum_{i=1}^{m} ||G_2(\hat{G}_1(x_i)) - x_i||_2^2 \tag{20}$$

$$\hat{G}_1 = \arg\min_{G_1} \mathcal{N}(G_1) + \gamma(L_{DisA} + L_{cyc}) \tag{21}$$

Two populations are kept in the unpaired image conversion task for the two generator networks, respectively. In order to obtain portable architectures of adequate performance, these two communities are alternatively updated by leveraging the best individuals in the previous iteration.

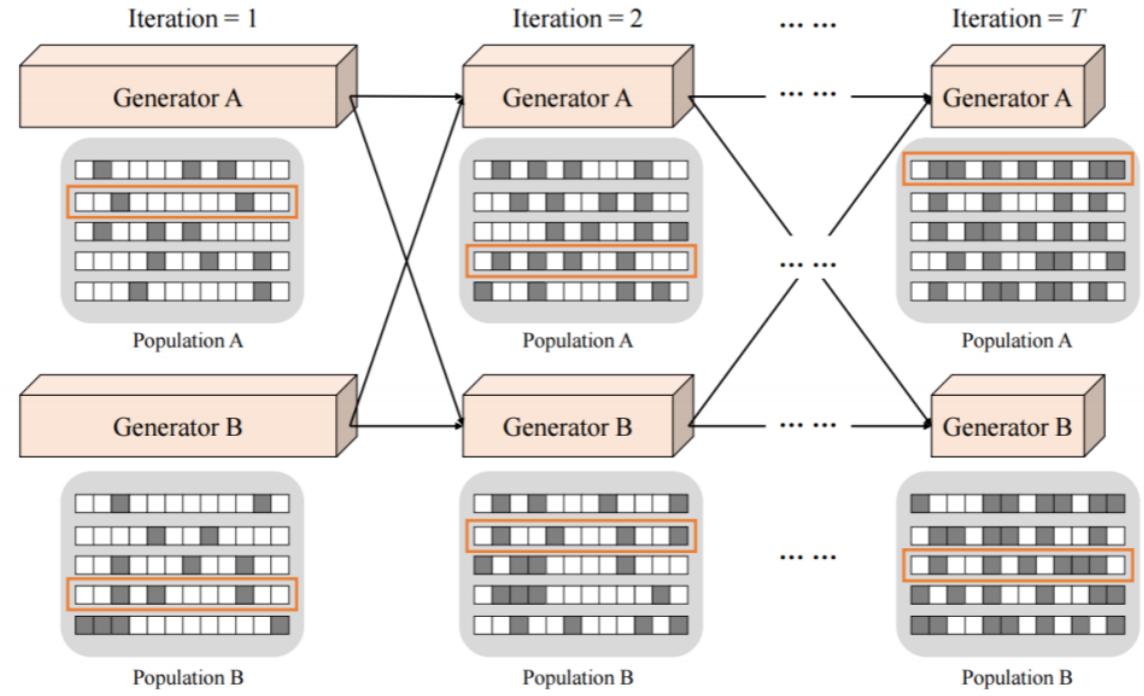


Figure 3.2: The diagram of the Co-Evolutionary method for learning efficient generators. Reprinted from [4].

3.4 KD Based Techniques

Following methods use KD to compress GANs:

3.4.1 Distilling portable GANs

Distilling portable GANs for image translation [40] uses various knowledge distillation losses to train a student generator (G_S) of fewer parameters from the original heavy teacher generator (G_T). L_1 loss is used to transfer low-level information from G_T to G_S over generated images as shown in Eq. (22). G_S acquires high-level information from G_T via L_{perc} loss as shown in Eq. (23) where D_T is the teacher discriminator. An adversarial learning process is established to optimize G_S and D_S as seen in Eq. (24). The images from G_T are used as real samples as seen in Eq. (25), which allows G_S to mimic real images as well as the images generated by G_T. To promote the capability of G_S, L_{tri} is included to measure the L_1 distances between real images, and images generated by G_S and G_T as

shown in Eq. (26). All the losses are combined with pre-defined hyper-parameters into a single knowledge distillation objective as seen in Eq. (27) where x, y are real input data and labels.

$$L_1(G_S) = \frac{1}{n}\sum_{i=1}^{n} ||G_T(x_i) - G_S(x_i)||_1^2 \tag{22}$$

$$L_{perc}(G_S) = \frac{1}{n}\sum_{i=1}^{n} ||\hat{D_T}(G_T(x_i)) - \hat{D_T}(G_S(x_i))||_1^2 \tag{23}$$

$$L_{GAN}(G_S, D_S) = \mathbb{E}_{x,y}[\log(D_S(x, y)] + \mathbb{E}_x[\log(1 - D_S(x, G_S(x)))] \tag{24}$$

$$L_{G_T}(D_S) = \frac{1}{n}\sum_{i=1}^{n} D_S(G_T(x_i), \textbf{True}) \tag{25}$$

$$L_{tri}(D_S) = \frac{1}{n}\sum_{i=1}^{n} \Big[||\hat{D_S}(y_i) - \hat{D_S}(G_T(x_i))||_1 - ||\hat{D_S}(y_i) - \hat{D_S}(G_S(x_i))||_1\Big] \tag{26}$$

$$\begin{aligned} L_{KD}(G_S, D_S) = L_{GAN}(G, D) + \beta_1 L_1(G_S) + \gamma_1 L_{perc}(G_S) + \\ \beta_2 L_{G_T}(D_S) + \gamma_2 L_{tri}(D_S) \end{aligned} \tag{27}$$

3.4.2 Compressing GANs

Compressing GANs based on knowledge distillation [41] uses mean square error as shown in Eq. (28) between the images generated from the student generator G_S and the teacher generator G_T as an additional loss to GAN adversarial training of Eq. (29) where D_S is the student discriminator. Both losses are combined into a joint loss using a hyper-parameter α as shown in Eq. (30). It can compress teacher GANs at ratios $1669:1$, $58:1$ and $87:1$ on MNIST, CIFAR-10, and Celeba-A datasets.

$$L_{MSE}(G_S) = \frac{1}{n}\sum_{i=1}^{n} ||G_T(x_i) - G_S(x_i)||_2^2 \tag{28}$$

$$L_{GAN} = \mathbb{E}_{x \in p_{data}}[log(D_S(x))] + \mathbb{E}_{z \in \mathcal{N}(0,1)}[1 - log(D_S(G_S(z)))] \tag{29}$$

$$\min_{G_S} \max_{D_S} L = \min_{G_S} \left[\max_{D_S} L_{GAN} + \alpha L_{MSE}(G_S) \right] \tag{30}$$

3.5 Inapplicable to StarGAN-v2

While recently developed approaches can provide very high compression and speed-up ratios with minor performance loss, for the following reasons, they are not specifically applicable to StarGAN-v2:

(1) Current techniques compress only a single generator network, while StarGAN-v2 is a multi-network system.

(2) Most approaches operate on deterministic generator networks, while StarGAN-v2 uses random Gaussian noise to produce diverse images from a single source image.

3.6 Summary

In this chapter, recent approaches to compress GANs using KD, NAS and neural network pruning techniques were mentioned. Then, the motivation of our research objective was explained. Our proposed method for compressing StarGAN-v2 will be mentioned in the next chapter.

Chapter 4

Method

This chapter provides our proposed algorithm in detail. We look at all of the architectural changes needed to obtain efficient models in section 4.1. Section 4.2 implements a style discriminator network that is essential for the transfer of teacher knowledge to the student mapping network. In order to guide the effective student model towards convergence, section 4.3 implements all the required knowledge distillation losses. Finally, we explain the complete aim of our training system in section 4.4.

4.1 Efficient Architecture Development

Knowledge distillation is most successful when there is a high degree of correlation between teacher and student model architectures. Therefore, we use the original generator, discriminator, style encoder,and mapping network as our baseline student architectures. Since our primary aim is to obtain memory efficient networks for resource constrained environments, we focus on altering the baseline models with efficient layers and reduced channels.

4.1.1 Residual Blocks (ResBlk)

Residual Blocks (ResBlk) [11] are used extensively in StarGAN-v2 architectures. Figure 4.1 displays all the layers of a typical residual block used for downsampling the input activation maps, with the regular convolutional layers being computationally expensive.

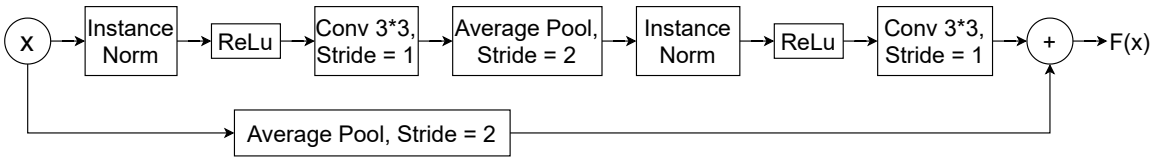


Figure 4.1: An illustration of intermediate layers of a Residual Blocks used for encoding

4.1.2 Mobile Residual Blocks (MobileResBlk)

Mobile Residual Blocks (MobileResBlk) contain efficient depthwise convolution and point-wise convolution [7] layers. The former is a spatial convolution while the latter is a channel wise convolution that operates on the channel dimensions. MobileResBlks are used in place of all the ResBlks for the encoding parts of all the modules of StarGAN-v2.

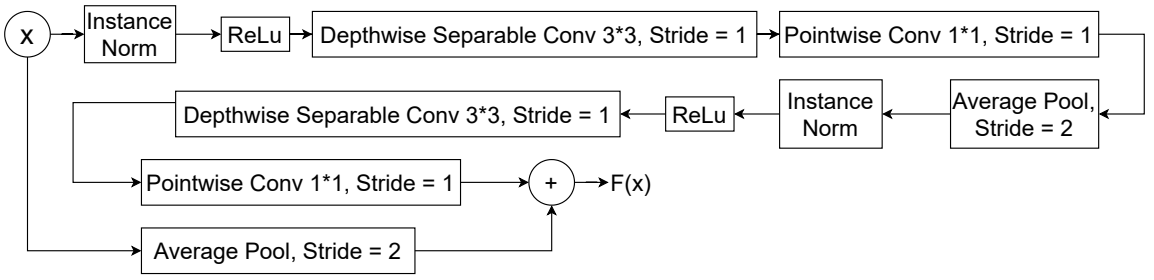


Figure 4.2: An illustration of intermediate layers of a Mobile Residual Blocks used for encoding

4.1.3 Generator

Generator contains multiple residual blocks for encoding and decoding. The network is defined to have a maximum channel width for each layer. Table 4.1 displays an example of original generator architecture with 512 maximum number of channels [6]. All the residual blocks used for encoding with are replaced with mobile residual blocks. The maximum number of channels is obtained empirically on standard dataset performance. Table 4.2 displays the proposed generator architecture with 128 maximum number of channels for any intermediate convolution layer.

LAYER	RESAMPLE	NORM	OUTPUT SHAPE	PARAMETERS
RGB Image	-	-	$256 \times 256 \times 3$	0
Conv1 $\times$ 1	-	-	$256 \times 256 \times 64$	1792
ResBlk	AvgPool	IN	$128 \times 128 \times 128$	119232
ResBlk	AvgPool	IN	$64 \times 64 \times 256$	476032
ResBlk	AvgPool	IN	$32 \times 32 \times 512$	1902336
ResBlk	AvgPool	IN	$16 \times 16 \times 512$	4721664
ResBlk	-	IN	$16 \times 16 \times 512$	4721664
ResBlk	-	IN	$16 \times 16 \times 512$	4721664
ResBlk	-	AdaIN	$16 \times 16 \times 512$	4852736
ResBlk	-	AdaIN	$16 \times 16 \times 512$	4852736
ResBlk	Upsample	AdaIN	$32 \times 32 \times 512$	4852736
ResBlk	Upsample	AdaIN	$64 \times 64 \times 256$	2000896
ResBlk	Upsample	AdaIN	$128 \times 128 \times 128$	525312
ResBlk	Upsample	AdaIN	$256 \times 256 \times 64$	143872
Conv1 $\times$ 1	-	-	$256 \times 256 \times 3$	323
			Total	33892995

Table 4.1: Original Generator architecture with 512 as maximum channel width

LAYER	RESAMPLE	NORM	OUTPUT SHAPE	PARAMETERS
RGB Image	-	-	$256 \times 256 \times 3$	0
Conv1 $\times$ 1	-	-	$256 \times 256 \times 64$	1792
MobileResBlk	AvgPool	IN	$128 \times 128 \times 128$	22208
MobileResBlk	AvgPool	IN	$64 \times 64 \times 128$	36096
MobileResBlk	AvgPool	IN	$32 \times 32 \times 128$	36096
MobileResBlk	AvgPool	IN	$16 \times 16 \times 128$	36096
MobileResBlk	-	IN	$16 \times 16 \times 128$	36096
MobileResBlk	-	IN	$16 \times 16 \times 128$	36096
ResBlk	-	AdaIN	$16 \times 16 \times 128$	328448
ResBlk	-	AdaIN	$16 \times 16 \times 128$	328448
ResBlk	Upsample	AdaIN	$32 \times 32 \times 128$	328448
ResBlk	Upsample	AdaIN	$64 \times 64 \times 128$	328448
ResBlk	Upsample	AdaIN	$128 \times 128 \times 128$	328448
ResBlk	Upsample	AdaIN	$256 \times 256 \times 64$	143872
Conv1 $\times$ 1	-	-	$256 \times 256 \times 3$	323
			Total	1990915

Table 4.2: Proposed Generator architecture with 128 as maximum channel width

4.1.4 Style Encoder

The style encoder includes numerous residual blocks and FC layers that are shared between all branches of the domain. Table 4.3 demonstrates the original style encoder network with a maximum 512 channel count on every hidden layer. All residual blocks are replaced

with mobile residual blocks, and the maximum channel count is obtained empirically on standard dataset performance. Table 4.4 shows the 128 maximum number of channels in our proposed style encoder architecture.

LAYER	RESAMPLE	NORM	OUTPUT SHAPE	PARAMETERS
RGB Image	-	-	$256 \times 256 \times 3$	0
Conv1 $\times$ 1	-	-	$256 \times 256 \times 64$	1792
ResBlk	AvgPool	-	$128 \times 128 \times 128$	118976
ResBlk	AvgPool	-	$64 \times 64 \times 256$	475520
ResBlk	AvgPool	-	$32 \times 32 \times 512$	1901312
ResBlk	AvgPool	-	$16 \times 16 \times 512$	4719616
ResBlk	AvgPool	-	$8 \times 8 \times 512$	4719616
ResBlk	AvgPool	-	$4 \times 4 \times 512$	4719616
LReLU	-	-	$4 \times 4 \times 512$	0
Conv4 $\times$ 4	-	-	$1 \times 1 \times 512$	4194816
LReLU	-	-	$1 \times 1 \times 512$	0
ReShape	-	-	512	0
Linear$\times$3	-	-	3×64	98496
			Total	20949760

Table 4.3: Original Style Encoder architecture with 512 as maximum channel width

LAYER	RESAMPLE	NORM	OUTPUT SHAPE	PARAMETERS
RGB Image	-	-	$256 \times 256 \times 3$	0
Conv1 $\times$ 1	-	-	$256 \times 256 \times 64$	1792
MobileResBlk	AvgPool	-	$128 \times 128 \times 128$	21952
MobileResBlk	AvgPool	-	$64 \times 64 \times 128$	35584
MobileResBlk	AvgPool	-	$32 \times 32 \times 128$	35584
MobileResBlk	AvgPool	-	$16 \times 16 \times 128$	35584
MobileResBlk	AvgPool	-	$8 \times 8 \times 128$	35584
MobileResBlk	AvgPool	-	$4 \times 4 \times 128$	35584
LReLU	-	-	$4 \times 4 \times 128$	0
Conv4 $\times$ 4	-	-	$1 \times 1 \times 128$	4194816
LReLU	-	-	$1 \times 1 \times 128$	0
ReShape	-	-	128	0
Linear$\times$3	-	-	3×64	262272
			Total	488704

Table 4.4: Proposed Style Encoder architecture with 128 as maximum channel width

4.1.5 Mapping Network

Mapping network comprises several FC layers shared between all domain branches with ReLU non-linearity. In the intermediate layers, the network has a limit on the overall number of hidden modules. Table 4.5 displays the original 512 maximum hidden node architecture. We get a similar architecture with empirically obtained maximum hidden node count on standard dataset efficiency. Table 4.6 shows our proposed network, decreasing the maximum number of hidden nodes to 128.

TYPE	LAYER	ACTIVATION	OUTPUT SHAPE	PARAMETERS
Shared	Latent z	-	16	0
Shared	Linear	ReLU	512	8704
Shared	Linear	ReLU	512	262656
Shared	Linear	ReLU	512	262656
Shared	Linear	ReLU	512	262656
Unshared	Linear	ReLU	512	262656
Unshared	Linear	ReLU	512	262656
Unshared	Linear	ReLU	512	262656
Unshared	Linear	-	64	32832
			Total	1617472

Table 4.5: Original Mapping Network architecture with 512 as maximum hidden nodes

TYPE	LAYER	ACTIVATION	OUTPUT SHAPE	PARAMETERS
Shared	Latent z	-	16	0
Shared	Linear	ReLU	128	2176
Shared	Linear	ReLU	128	16512
Shared	Linear	ReLU	128	16512
Shared	Linear	ReLU	128	16512
Unshared	Linear	ReLU	128	16512
Unshared	Linear	ReLU	128	16512
Unshared	Linear	ReLU	128	16512
Unshared	Linear	-	64	8256
			Total	109504

Table 4.6: Proposed Mapping Network architecture with 128 as maximum hidden nodes

4.2 Style Discriminator

The mapping network creates style codes from latent codes based on random Gaussian noise. We introduce the style discriminator as seen in Figure 4.3 in order to transfer knowledge from teacher to student mapping network via adversarial learning [42]. It is a multi-task network which contains multiple linear output branches, one for each domain. As shown in Figure 4.4, each branch of the style discriminator learns a binary classification that determines if the style code is created by adversarial loss by the teacher mapping network or the student mapping network of its domain. The style discriminator capacity is close to the student mapping network. For non-linear activation Leaky-ReLU layers are used in intermediate layers.

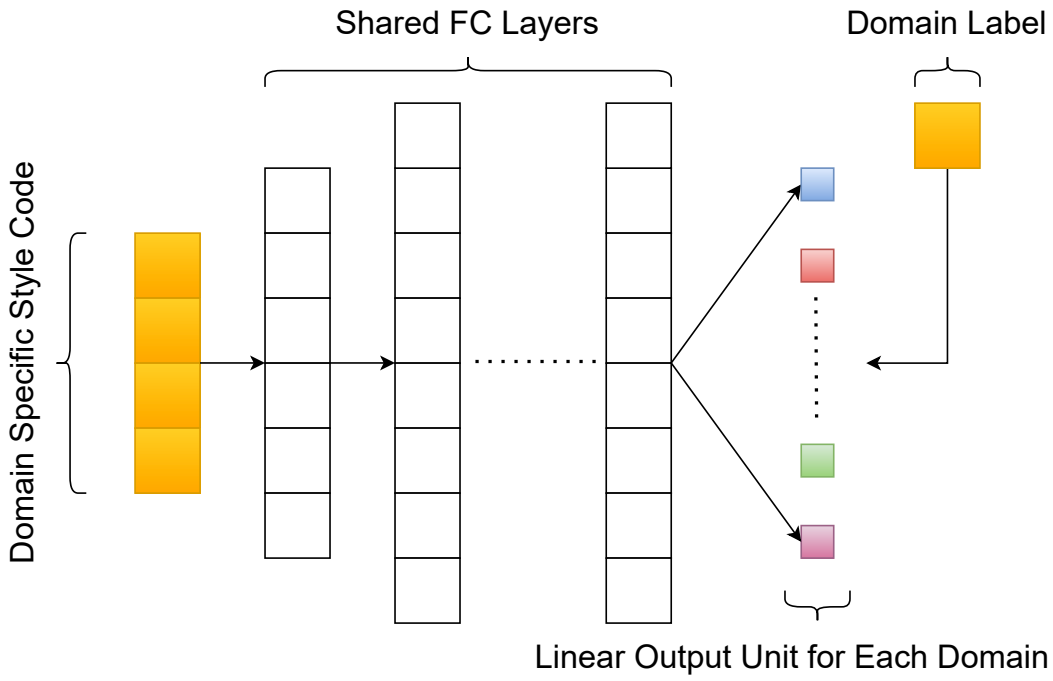


Figure 4.3: Workflow of Style Discriminator. The input domain specific style code is classified as originated from teacher or student mapping network.

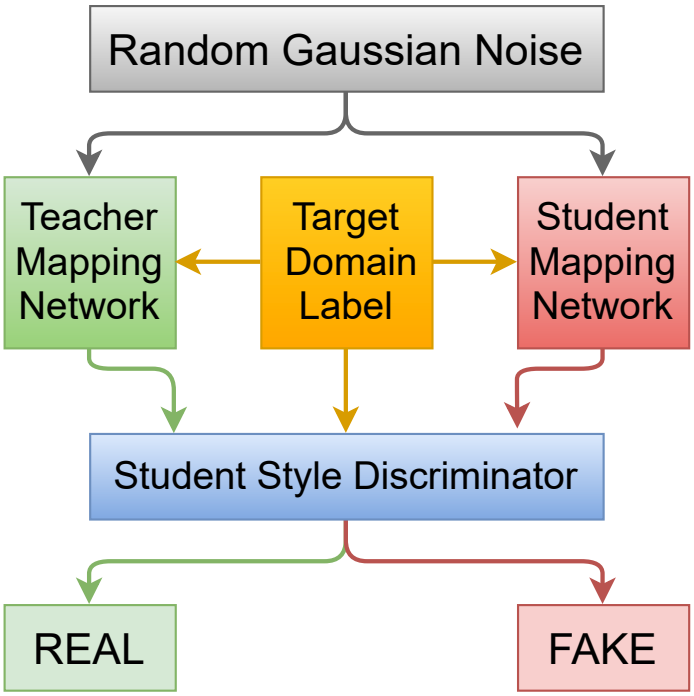


Figure 4.4: Flowchart of style code distillation in adversarial setting.

4.3 Distillation Losses

Inspired by the success of knowledge distillation in recent works [6, 33, 36, 4, 40], we add multiple model distillation losses to enforce student modules to mimic the behaviour of original networks of StarGAN-v2 framework. We do not use a pre-trained teacher discriminator in our training.

Denoting our pre-trained teacher generator as G_T, style encoder as E_T, mapping network as F_T and similarly, student generator as G_S, style encoder as E_S, mapping network as F_S, image discriminator as D, and style discriminator as D^S.

Let $\mathcal{X}$, $\mathcal{Y}$ be the sets of images and possible domains and $\mathcal{Z}$ be the set of all possible latent codes. During training, we randomly sample an image $x \in \mathcal{X}$ with its original domain label $y \in \mathcal{Y}$, two latent codes $z_1 \in \mathcal{Z}$, $z_2 \in \mathcal{Z}$, and a target reference image $\tilde{x} \in \mathcal{X}$ with its target domain label $\tilde{y} \in \mathcal{Y}$. We generate target style codes using teacher networks:

$$s_{z_1} = F_T(z_1, \tilde{y}) \qquad s_{z_2} = F_T(z_2, y) \qquad s_{\tilde{x}} = E_T(\tilde{x}, \tilde{y}) \qquad s_x = E_T(x, y)$$

4.3.1 Image Adversarial Losses

We generate images from G_T and G_S using s_{z_1}, $s_{\tilde{x}}$, and x. The image discriminator D learns to assess the divergence between images generated by G_T and G_S. D maximizes the divergence, while G_S minimizes it as shown in Eqs. (31) and (32). In this way, G_S learns to mimic G_T implicitly.

$$\mathcal{L}^1_{G_S,D} = \mathbb{E}_{x,z_1,\tilde{y}}[\log(D_{\tilde{y}}(G_T(x, s_{z_1})) + log(1 - D_{\tilde{y}}(G_S(x, s_{z_1})))] \tag{31}$$

$$\mathcal{L}^2_{G_S,D} = \mathbb{E}_{x,\tilde{x},\tilde{y}}[\log(D_{\tilde{y}}(G_T(x, s_{\tilde{x}})) + log(1 - D_{\tilde{y}}(G_S(x, s_{\tilde{x}})))] \tag{32}$$

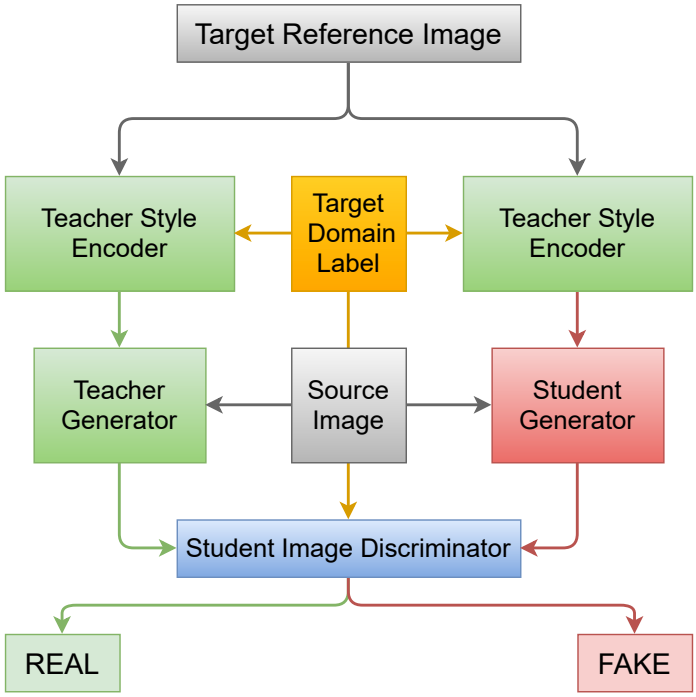


Figure 4.5: Flowchart of reference-guided image synthesis by teacher and student.

4.3.2 Style Adversarial Losses

We generate style codes from F_S using z_1, z_2, y, and $\tilde{y}$. The style discriminator D^S learns to assess the divergence between style codes generated by F_T and F_S. D^S maximizes the

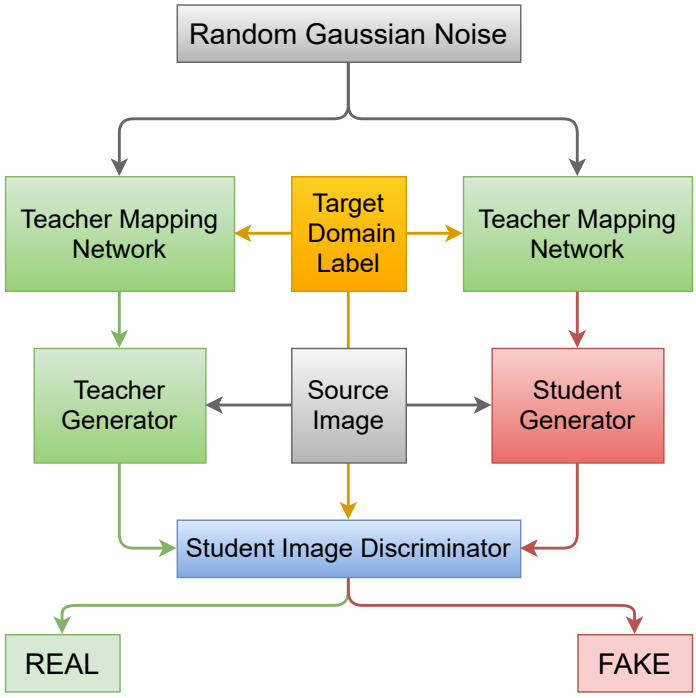


Figure 4.6: Flowchart of latent-guided image synthesis by teacher and student.

divergence, while F_S minimizes it as shown in Eqs. (33) and (34). In this way, F_S learns to mimic F_T implicitly.

$$\mathcal{L}^1_{F_S,D^S} = \mathbb{E}_{z_1,\tilde{y}}[\log(D^S_{\tilde{y}}(s_{z_1})) + log(1 - D^S_{\tilde{y}}(F_S(z_1,\tilde{y})))] \tag{33}$$

$$\mathcal{L}^2_{F_S,D^S} = \mathbb{E}_{z_2,y}[\log(D^S_{y}(s_{z_2})) + log(1 - D^S_{y}(F_S(z_2,y)))] \tag{34}$$

4.3.3 Style Utilization Losses

The student style encoder E_S learns to extract style code similar to target style codes over the images generated by G_S using s_{z_1}, $s_{\tilde{x}}$, and x. Meanwhile, G_S learns to utilize target style codes via L_1 loss. This objective is similar to previous approaches [19, 20] as shown in Eqs. (35) and (36).

$$\mathcal{L}^1_{G_S,E_S} = \mathbb{E}_{x,z_1,\tilde{y}}||s_{z_1} - E_S(G_S(x,s_{z_1}),\tilde{y})||_1 \tag{35}$$

$$\mathcal{L}^2_{G_S,E_S} = \mathbb{E}_{x,\tilde{x},\tilde{y}}||s_{\tilde{x}} - E_S(G_S(x, s_{\tilde{x}}), \tilde{y})||_1 \tag{36}$$

4.3.4 Source Attributes Preservation Losses

The student generator learns to preserve the domain invariant characteristics of input image x while generating images using s_{z_1} and $s_{\tilde{x}}$. We employ the cycle consistency loss [22, 23, 24] by reconstructing the source image from generated images using s_x via L_1 loss as shown in Eqs. (37) and (38).

$$\mathcal{L}^1_{G_S} = \mathbb{E}_{x,z_1,\tilde{y}}||x - G_S(G_S(x, s_{z_1}), s_x)||_1 \tag{37}$$

$$\mathcal{L}^2_{G_S} = \mathbb{E}_{x,\tilde{x},\tilde{y}}||x - G_S(G_S(x, s_{\tilde{x}}), s_x)||_1 \tag{38}$$

4.4 Full Objective

Our full objective utilizes the original training objective loss $\mathcal{L}_{org}$ from Eq. (12) along with our distillation losses combined into minimax optimization setting as shown in Eq. (39).

$$\begin{aligned}\min_{G_S,E_S,F_S} \quad \max_{D^S,D} [&\mathcal{L}_{org} + \mathcal{L}^1_{G_S,D} + \mathcal{L}^2_{G_S,D} + \mathcal{L}^1_{F_S,D^S} + \\ &\mathcal{L}^2_{F_S,D^S} + \mathcal{L}^1_{G_S,E_S} + \mathcal{L}^2_{G_S,E_S} + \mathcal{L}^1_{G_S} + \mathcal{L}^2_{G_S}]\end{aligned} \tag{39}$$

4.5 Summary

In this chapter, in our proposed approach, architectural adaptation to StarGAN-v2 networks for efficiency and loss function for distillation was explained. Experimental analyses of our research method and comparisons with the original StarGAN-v2 networks will be done in the next chapter.

Chapter 5

Experiments

This chapter contains information about the experiments performed to test the approach suggested. The datasets and measurement metrics used for analysis are defined in sections 5.1 and 5.2. The specifics related to training are given in the 5.3 section. Finally, qualitative and quantitative distinctions are made between the proposed methodology and the initial framework in section 5.4.

5.1 Datasets

For our assessments, we use the datasets **CelebA-HQ** [18] and **AFHQ** [6]. As seen in the Figure 5.1, the CelebA-HQ dataset is split into two male and female realms. As seen in Figure 5.2, the AFHQ dataset comprises three cat, dog, and wildlife domains. As seen in Table 5.1, we observe the train and validation splits. As originally used in StarGAN-v2, all images are resized to 256×256.

	CelebA-HQ		**AFHQ**		
	Male	Female	Cat	Dog	Wildlife
Train	10057	17943	5153	4739	4738
Val	1000	1000	500	500	500

Table 5.1: Train-Validation splits for CelebA-HQ and AFHQ datasets.

Figure 5.1: Sample Training Images of CelebA-HQ

Figure 5.2: Sample Training Images of AFHQ

5.2 Evaluation Metrics

To measure the visual quality and the diversity of images produced, FID and LPIPS are used. For comparison, the computational requirements and size of networks is also computed. For latent-guided synthesis, each test image from a source domain is translated into a target domain using 10 latent vectors sampled randomly from a regular Gaussian distribution. Similarly, for reference-guided synthesis each test image from the source domain is transformed using 10 reference images randomly sampled from the target domain test collection.

5.2.1 FID

The FID is calculated between the translated images and training images in the target domain. For each pair of image domains (e.g. female, male for CelebA-HQ), the FID values are measured and recorded the average value separately for reference-guided synthesis and latent-guided synthesis.

5.2.2 LPIPS

The average pair distance is determined between all outputs produced from the same input (i.e. 45 pairs). Finally, for reference-guided synthesis and latent-guided synthesis, the LPIPS is computed separately by means of averages over all test images.

5.2.3 MACs and Parameter Count

The total number of parameters combined in the Generator, Style Encoder, and Mapping Network are listed as the framework size. Similarly, by aggregating MACs count of a single forward pass of each module, the total computing cost for the system is calculated.

5.3 Implementation Details

Details regarding the training parameters, data augmentation and network parameters is given below:

5.3.1 Training Parameters

Each experiment was run on a Tesla P100 32GB GPU containing Cuda (10.0), Python (3.8.5), Numpy (1.19.1) and Pytorch (1.6.0). A batch size of 8 is used and training is performed for $100K$ iterations with checkpoints stored after each $10K$ iterations in each experiment. Each training cycle takes approximately 3 days. Adam optimizer [43] is used with $\beta_1 = 0$ and $\beta_2 = 0.99$. For training without distillation, the learning rates for G, D and E are set to 10^{-4}, while that of F is set to 10^{-6}, whereas with distillation the learning rates for all the networks is set to 10^{-4}. All the experiments are performed with a fixed random seed value 777.

5.3.2 Data Augmentation

Following data augmentation techniques are applied on the images of both source and target domains:

- Images are horizontally flipped with a probability of 0.5.
- A random crop of Size $\in [0.8, 1.0]$ and Aspect Ratio $\in [0.9, 1.1]$ is extracted from original image and resized to 256×256.

5.3.3 Loss Parameters

For CelebA-HQ, $\lambda_{sty} = 1$, $\lambda_{ds} = 1$, and $\lambda_{cyc} = 1$, and for AFHQ, $\lambda_{sty} = 1$, $\lambda_{ds} = 2$, and $\lambda_{cyc} = 1$ are used. To stabilize the training, the weight λ_{ds} is linearly decayed to zero over the $100K$ iterations. A non-saturating adversarial loss with zero-centered gradient penalty is used for training.

5.3.4 Network Parameters

The weights of all the networks are initialized with He [44] initialization and all biases are set to zero, except for the biases associated with the scaling vectors of AdaIN that are set to one. Exponential Moving Averages (EMA) is applied on all the modules excluding the Discriminator's parameters. Lengths are set at 16 and 64 for the latent code and style

code. Pretrained networks with maximum hidden layer channels set to 512 are used as teacher networks. In order to control the capacity of the whole framework α is defined as a single global parameter for the maximum number of channels in convolution layers and the maximum number of hidden nodes in linear layers for all the networks. Experiments are done with different values of $\alpha \in [64, 96, 128, 160]$ and the score of best scoring checkpoint is reported after evaluation of all the checkpoints for each α.

5.4 Results

In Tables 5.2 and 5.3 a summary of evaluation scores on test images by using our proposed method with different values of α on both CelebA-HQ and AFHQ datasets is reported. At $\alpha = 64$ and 96 we can see a huge drop in performance whereas the scores at $\alpha = 128$ and $\alpha = 160$ are close to that of the original method. We obtain a higher compression rate at $\alpha = 128$ compared to $\alpha = 160$, thus achieving a better balance of compression and performance. Figures 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 and 5.10 display images generated by the original and our proposed method at $\alpha = \mathbf{128}$, for reference-guided and latent-guided image synthesis on test images. At $\alpha = \mathbf{128}$, our method compresses original StarGAN-v2 framework by more than $\mathbf{20}\times$ in size and by more than $\mathbf{5}\times$ in MACs and produces images with high quality and diverse styles across all domains on both CelebA-HQ and AFHQ datasets as good as the original StarGAN-v2.

5.5 Summary

In this chapter, details related to standard datasets and training parameters used have been listed. Then the performancee of networks trained using our proposed method at various capacities was compared with original networks. In the next chapter, the potential scope of our research work will be provided.

Dataset	Method	Latent-Guided Synthesis		Reference-Guided Synthesis	
		FID	**LPIPS**	**FID**	**LPIPS**
AFHQ	Original	16.09	0.451	19.73	0.431
	Ours ($\alpha = 64$)	26.65	0.492	30.49	0.448
	Ours ($\alpha = 96$)	23.52	0.446	24.21	0.425
	Ours ($\alpha = \mathbf{128}$)	**20.69**	**0.437**	**21.60**	**0.415**
	Ours ($\alpha = 160$)	20.95	0.421	21.32	0.408
CelebA-HQ	Original	13.76	0.451	23.88	0.388
	Ours ($\alpha = 64$)	20.69	0.423	27.17	0.381
	Ours ($\alpha = 96$)	19.30	0.425	26.23	0.382
	Ours ($\alpha = \mathbf{128}$)	**18.41**	**0.417**	**25.18**	**0.385**
	Ours ($\alpha = 160$)	18.91	0.419	26.24	0.380

Table 5.2: Quantitative evaluation of our proposed method on validation images. FID(the lower the better) and LPIPS (the higher the better) for both latent and reference guided synthesis are reported.

Dataset	Method	Size (M) (E+G+F)	MACs (G) (E+G+F)
AFHQ	Original	58.10	65.30
	Ours ($\alpha = 64$)	0.81 (71×)	5.37 (12×)
	Ours ($\alpha = 96$)	1.62 (35×)	8.54 (7×)
	Ours ($\alpha = \mathbf{128}$)	2.71 (**21**×)	11.53 (**5**×)
	Ours ($\alpha = 160$)	3.94 (14×)	13.47 (4×)
CelebA-HQ	Original	66.82	62.03
	Ours ($\alpha = 64$)	0.89 (75×)	5.35 (11×)
	Ours ($\alpha = 96$)	1.79 (37×)	8.09 (7×)
	Ours ($\alpha = \mathbf{128}$)	3.00 (**22**×)	10.92 (**5**×)
	Ours ($\alpha = 160$)	4.40 (15×)	12.49 (4×)

Table 5.3: Individual size and MACs of generator (G), style encoder (E), and mapping network (F) combined are reported for networks trained with our proposed method.

Figure 5.3: Reference-guided image synthesis using **original** networks of AFHQ test images.

Figure 5.4: Reference-guided image synthesis using **proposed (α=128)** networks of AFHQ test images.

Figure 5.5: Reference-guided image synthesis using **original** networks of CelebA-HQ test images.

Figure 5.6: Reference-guided image synthesis using **proposed** (α=**128**) networks of CelebA-HQ test images.

Figure 5.7: Latent-guided image synthesis using **original** networks of AFHQ test images.

Figure 5.8: Latent-guided image synthesis using **proposed (α=128)** networks of AFHQ test images.

Figure 5.9: Latent-guided image synthesis using **original** networks of CelebA-HQ test images.

Figure 5.10: Latent-guided image synthesis using **proposed (α=128)** networks of CelebA-HQ test images.

Chapter 6

Conclusions and Future Work

This chapter will give the conclusion and future work of this thesis.

6.1 Conclusions

In this thesis we examined the problem of StarGAN-v2 deployment in resource constraint environments. We explored recent techniques on compressing GANs and defined the need to develop a novel approach to obtain light-weight StarGAN-v2 networks. This work contributes to a method of compressing a multi-network framework. We also introduced style discriminator by drawing ideas on distilling knowledge via adversarial loss. We also introduced efficient architecture for different modules of StarGAN-v2. These models are intended to be memory efficient as well as applicable to different datasets. Finally, we looked at the image translation performance of student models trained using our proposed knowledge distillation based method.

The core contribution of our work is in defining an end-to-end training algorithm to distill the knowledge from a multi-network framework. Experimentally, we observed that at a compression factor of approximately $\mathbf{20}\times$ in size and $\mathbf{5}\times$ in MACs, a reasonable balance between compression and performance is obtained. Additionally, we examined the performance of networks at different network capacities and observed that the FID score improvements reduces upon increasing the capacity, a potential drawback to our approach.

6.2 Future Work

Concerning the future work, we plan on extending this work to explore following major ideas:

(1) Include neural architecture search as a part of optimization to obtain a more competitive compression factor without reducing performance.

(2) Explore techniques to filter the knowledge from teacher networks. Not all images and style codes generated from teacher networks are of reasonable quality to be distilled.

Bibliography

[1] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for gans do actually converge?," in *International Conference on Machine Learning*, 2018. https://arxiv.org/pdf/1801.04406.pdf.

[2] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018. https://arxiv.org/pdf/1801.03924.pdf.

[3] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "Gan compression: Efficient architectures for interactive conditional gans," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5284–5294, 2020. https://arxiv.org/pdf/2003.08936.pdf.

[4] H. Shu, Y. Wang, X. Jia, K. Han, H. Chen, C. Xu, Q. Tian, and C. Xu, "Co-evolutionary compression for unpaired image translation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3235–3244, 2019. https://arxiv.org/pdf/1907.10804.pdf.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014. https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[6] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, "Stargan v2: Diverse image synthesis for multiple domains," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2020. https://openaccess.thecvf.com/content_CVPR_2020/papers/Choi_StarGAN_v2_Diverse_Image_Synthesis_for_Multiple_Domains_CVPR_2020_paper.pdf.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. https://arxiv.org/pdf/1704.04861.pdf.

[8] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015. https://arxiv.org/pdf/1503.02531.pdf.

[9] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, 1990. https://papers.nips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[10] S. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," *Neurocomputing*, 2020. https://arxiv.org/pdf/1902.02771.pdf.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. https://arxiv.org/pdf/1512.03385.pdf.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015. https://arxiv.org/pdf/1409.4842.pdf.

[13] V. Nair and E. G. Hinton, "Rectified linear units improve restricted boltzmann machines," *International Conference on Machine Learning*, 2010. https://www.cs.toronto.edu/%7Efritz/absps/reluICML.pdf.

[14] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," Citeseer, 2013. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.693.1422&rep=rep1&type=pdf.

[15] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016. https://arxiv.org/pdf/1607.08022.pdf.

[16] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017. https://arxiv.org/pdf/1703.06868.pdf.

[17] Y. Yazıcı, C.-S. Foo, S. Winkler, K.-H. Yap, G. Piliouras, and V. Chandrasekhar, "The unusual effectiveness of averaging in gan training," *arXiv preprint arXiv:1806.04498*, 2018. https://arxiv.org/pdf/1806.04498.pdf.

[18] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017. https://arxiv.org/pdf/1710.10196.pdf.

[19] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," in *European conference on computer vision*, 2018. https://openaccess.thecvf.com/content_ECCV_2018/papers/Xun_Huang_Multimodal_Unsupervised_Image-to-image_ECCV_2018_paper.pdf.

[20] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman, "Toward multimodal image-to-image translation," *arXiv preprint arXiv:1711.11586*, 2017. https://arxiv.org/pdf/1711.11586.pdf.

[21] Q. Mao, H.-Y. Lee, H.-Y. Tseng, S. Ma, and M.-H. Yang, "Mode seeking generative adversarial networks for diverse image synthesis," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019. https://arxiv.org/pdf/1903.05628.pdf.

[22] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018. https://arxiv.org/pdf/1711.09020.pdf.

[23] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," in *International Conference on Machine Learning*, 2017. https://arxiv.org/pdf/1703.05192.pdf.

[24] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017. https://arxiv.org/pdf/1703.10593.pdf.

[25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *arXiv preprint arXiv:1706.08500*, 2017. https://arxiv.org/pdf/1706.08500.pdf.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012. https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[27] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016. https://arxiv.org/pdf/1611.01578.pdf.

[28] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, 2021. https://arxiv.org/pdf/1908.00709.pdf.

[29] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," *arXiv preprint arXiv:2003.03033*, 2020. https://arxiv.org/pdf/2003.03033.pdf.

[30] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017. https://arxiv.org/pdf/1611.07004.pdf.

[31] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, "Semantic image synthesis with spatially-adaptive normalization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019. https://arxiv.org/pdf/1903.07291.pdf.

[32] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019. https://arxiv.org/pdf/1908.09791.pdf.

[33] Y. Fu, W. Chen, H. Wang, H. Li, Y. Lin, and Z. Wang, "Autogan-distiller: Searching to compress generative adversarial networks," *arXiv preprint arXiv:2006.08198*, 2020. https://arxiv.org/pdf/2006.08198.pdf.

[34] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018. https://arxiv.org/pdf/1806.09055.pdf.

[35] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019. https://arxiv.org/pdf/1812.03443.pdf.

[36] H. Wang, S. Gui, H. Yang, J. Liu, and Z. Wang, "Gan slimming: All-in-one gan compression by a unified optimization framework," in *European Conference on Computer Vision*, pp. 54–73, 2020. https://arxiv.org/pdf/2008.11062.pdf.

[37] Y. Chen, Y. Lai, and Y. Liu, "Cartoongan: Generative adversarial networks for photo cartoonization," in *Proceedings of the IEEE conference on computer vision and pattern*

recognition, 2018. https://openaccess.thecvf.com/content_cvpr_2018/papers/Chen_CartoonGAN_Generative_Adversarial_CVPR_2018_paper.pdf.

[38] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European conference on computer vision*, 2016. https://arxiv.org/pdf/1603.08155.pdf.

[39] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017. https://arxiv.org/pdf/1708.06519.pdf.

[40] H. Chen, Y. Wang, H. Shu, C. Wen, C. Xu, B. Shi, C. Xu, and C. Xu, "Distilling portable generative adversarial networks for image translation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3585–3592, 2020. https://arxiv.org/pdf/2003.03519.pdf.

[41] A. Aguinaldo, P.-Y. Chiang, A. Gain, A. Patil, K. Pearson, and S. Feizi, "Compressing gans using knowledge distillation," *arXiv preprint arXiv:1902.00159*, 2019. https://arxiv.org/pdf/1902.00159.pdf.

[42] V. Belagiannis, A. Farshad, and F. Galasso, "Adversarial network compression," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018. https://arxiv.org/pdf/1803.10750.pdf.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. https://arxiv.org/pdf/1412.6980.pdf.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015. https://arxiv.org/pdf/1502.01852.pdf.