Distilling Generative Networks for Mobile and Embedded Platforms

Dhritiman Sagar
SCPD
Stanford University
dhrits@stanford.edu

Abstract

Since their inception in 2014, Generative Adversarial Networks(10) have achieved wide scale popularity in tasks which involve generation of new data or domain translation. However, despite the progress in GAN architectures and advanced training techniques, less attention has been devoted to the training of efficient generative architectures which may run in real-time on embedded and mobile devices. This work explores the training of a small image-to-image translation GANs by using knowledge distillation(12) from a much larger network.

1 Introduction and Motivation

Generative Adversarial Networks(10) have become the de-facto method for generating new domain specific data (optionally) conditioned on some prior. Despite their dominance and prevalence in the field of generative techniques, GANs tend to be extremely compute intensive and state-of-the-art GANs can easily involve tens to hundreds of millions of parameters. Such architectures are not suitable for inference on platforms which involve compute budgets or real-time constraints. The goal of this project is to explore GAN compression techniques on image-to-image translation tasks and produce a model which is capable of running in near-realtime on commodity mobile and embedded hardware which has become commonplace in the era of smartphones and the Internet of Things (IOT).

Despite their dominance and prevalence in the field of generative techniques, GANs tend to be extremely unstable to train. Compute intensive and state-of-the-art GANs can easily involve tens to hundreds of millions of parameters. Such architectures are unsuitable for inference on platforms which have conservative compute or real-time constraints. The goal of this project is to explore GAN compression techniques on image-to-image translation tasks and produce a model which is capable of running in real-time on commodity, mobile and embedded platforms. Successfully doing so will unlock creative applications in fields ranging from mobile augmented and virtual reality, wearables, embedded robotics and IOT creative cameras.

2 Related Work

2.1 Mobile Architectures

In the past five years, there's been a lot of research aimed at finding efficient neural architectures for mobile devices. The first of these was MobileNet(13) which made use of depthwise-separable-convolutions (a kind of convolution factoring technique) which achieved performance comparable with AlexNet(19) with an order of magnitude fewer parameters. MobileNet-V2(23) further built upon MobileNet and introduced inverse-residual blocks into the model backbone which allowed for increased accuracy. Most recently, the EfficientNet(27) family of architectures allows for configurable parameters which can trade off classification accuracy for computational budget. The FastDepth(28) paper is interesting because it develops a mobile-optimized-architecture for regressing a depth map from a monocular image. This is interesting for this project because the network architecture is essentially an image-to-image translation network (following the encoder-decoder pattern).

2.2 GAN Optimization

As recently as mid-2021, the MobileStyleGAN(5) paper attempts to come up with a convolutional architecture and technique which can perform high-fidelity image synthesis with 3.5x fewer parameters and 9.5x less compute compared to StyleGAN. Also in 2021, the Teachers do More than Teach(14) paper attempts to algorithmically prune a configurable student architecture for a generator network which is based on a very large inception-based teacher network. The teacher and student networks are then trained together using knowledge distillation(12). While this technique is the current state-of-the-art in distilling GANs, it requires training of a very large custom teacher super-network from scratch which may be beyond the computational budget for many in the community (and this class).

This work borrows ideas and inspiration from all of the above works, but attempts to do so using a very simple knowledge distillation approach **using a single NVIDIA RTX 2080 GPU with 8 GB of memory**. The goal of this work is to both simplify training procedure as well as network computational budget. Thus making the innovations of larger GANs accessible to those with more modest computational budgets. Additionally, the primary ablations focus on design of loss functions to help the student architectures learn better. Thus the overarching goal of this project is to push the limits of domain translation given very modest training and inference time budgets using training techniques.

3 Approach

3.1 Problem Statement

For this particular project, we will explore distillation of GANs by restricting the scope of the task to **distilling image-to-image translation models for human faces**. Face translation using GANs is a very popular application which has gained widespread appeal through apps like Snapchat, Instagram and FaceApp. Additionally, by restricting the translation-domain to human faces, we allow the smaller student generator more flexibility in its compute while maintaining a smaller, compute efficient size. The datasets used below will thus reflect our focus on human faces.

3.2 Data

For training the student generator architecture, a custom version of the CelebA(21) dataset will be used. However, as it will become clear later in this report, exclusive use of the CelebA dataset may slightly overfit the model to images that look like portraits. As such, the dataset is mixed with subsets of Flickr(9) faces dataset (especially for validation). One of the contributions of this project is the creation of this paired dataset for three different anime styles.

Since the idea is to use knowledge distillation to train a student network, a fully trained AnimeGANV2(7)(6)(20) model will be used to generate a large amount of **additional training data** for the student network using the CelebA dataset. AnimeGAN-V2 has recently gained popularity on social media through its Huggingface Spaces web demo. The combined (paired) data will then be used to train the student network. Generally speaking, smaller models are much harder to train and require a significant amount of training data to overcome inherent bias due to a smaller number of parameters. There are a lot of face-based datasets and additional paired-data can easily be generated by the use of a larger pre-trained model. As a stretch goal, this work attempts to train multiple styles although the primary focus will be on the paprika anime style.

For this project, three custom paired datasets were generated using a pre-trained AnimeGAN-V2(20) model on the paprika, webtoon and face portrait v2 styles.

3.3 Baseline and Expected Results

A student model with significantly fewer parameters will never have the learning capacity of a larger teacher network. However, one of the primary objectives of this work is to as closely approximate the teacher network as possible. In order to accomplish this, the fidelity of the images generated by the teacher and student networks needs to be measured and compared. Evaluation of the quality of GAN generated results continues to be an active area of research, however, there are many methods to quantitatively measure the distance between two distributions of data given enough samples for each distribution.

For this work, the **Frechet Inception Distance (FID score)** is used to measure the distance between samples generated from the teacher network and the student network. This distance measure is calculated by computing the mean and covariance of each generated dataset as follows:

$$||\mu_1 - \mu_2||_2^2 + tr(\Sigma_1 + \Sigma_2 - 2tr(\Sigma_1\Sigma_2)^{\frac{1}{2}})$$

The theoretical baseline is the FID score between two random sample datasets generated from the teacher AnimeGAN-V2 generator. In an ideal world, it should be the same as as the FID score between a sample generated from the generator and a sample generated from the teacher and a sample generated from the student (if the student is a perfect approximation of the teacher). In other words, given sample datasets GT1, GT2 and GT3 generated from the teacher network and the dataset S generated using the student, the following approximation should hold.

$$FID(GT, S) \approx FID(GT1, GT2)$$

Of course, in practice, we will not be able to find such an approximation. So alternatively, we want to find parameters of a student network which minimizes the absolute value between the two FID scores. If ϕ represents the parameters of the student network then:

$$\arg\min_{\phi} |FID(GT1, GT2) - FID(GT, S)|$$

Unfortunately, the above metric is not differentiable. As such, we'll train our algorithm with a combination of loss functions which preserve structural and perceptual identity (Please see algorithms section).

3.4 Evaluation

For evaluating the student network, there are 3 key areas which need to be measured: inference speed, computational complexity and quality of generated results. The following high-level metrics will be used to measure these areas:

1. **MACs for computational complexity**: The number of multiply-accumulate (MAC) operations in the network will be used as a measure of its computational complexity.
2. **Model size**: The size of the mobile GAN is another measure of computational complexity. **It is expected that the mobile network will be an order of magnitude smaller than 8MB (the size of AnimeGAN-v2**.
3. **Inference time/Frames-per-Second**: Inference time for translating an image to a selfie is a direct measure of the speed of the network. Ideally, we can measure this inference speed on a mobile/embedded chipset.
4. **FID Score as a measure of quality**: Frechet Inception Distance(11) (FID) score will be used as a measure of the quality of the images generated by the mobile network.

4 Technical Approach

4.1 Algorithm

The overall algorithm for distilling a mobile-optimized Generator is as follows:

1. Generate a large paired dataset by making use of the existing CelebA(21) and Flickr(9) datasets together with a pretrained Anime GAN-V2(20) network.
2. Train the smaller network on the generated paired data. The architecture of the network will be based on other mobile architectures for image-to-image translation tasks like mobile depth inference(28) and Segmentation(3). Other than training on the final activations of the teacher network, we also (optionally) attempt to match the intermediate activations of the teacher network (see Loss Functions section below).
3. Find an appropriate learning rate to begin training by running an LR Finder(25)(24). In this approach, the learning rate is quickly increased between two limits while training on the data and the learning rate which produces the steepest gradient in loss is used to train the network.
4. Train the network using the Adam(17) optimizer till the overall loss function converges or plateaus. Choice of Adam optimizer is due to its prior success in Generative Adversarial Networks.

4.2 Loss Functions

The design of the loss function is one of the most important degrees of freedom for this project. The student network must be trained in a way that maximizes the similarity of its output with the teacher network while having approximately 1/5th the size. As such we need to choose loss functions which not-just naively maximize the per-pixel similarity, but also match the perceived style of the teacher outputs. As such, we define a multi-part loss function.

1. **Smooth L1 Loss:** To compare the ground-truth images to the generated images, we will use of the **Smoothed L1 Loss(1) function**. This loss function can be described as follows:

   $$l(a,x) = \begin{cases} 0.5(a-x)^2/\beta, & \text{if } |a-x| < \beta \\ |a-x| - 0.5\beta, & \text{otherwise} \end{cases}$$

   This loss function behaves like the L2 loss for small values and L1 loss otherwise. By using L1 loss for large differences, we avoid outliers in our teacher activations from unduly influencing the student network parameters.
2. **Sobel Edge Loss:** Based on experiments (see Experiments section), it was not enough to simply use a pixel-wise loss like above. To match the sharper anime-like edges in the teacher distribution, we had to additionally devise a loss function which matched the edges between teacher and student outputs. For this, we use the Kornia(8) computer-vision library to generate the edge gradient image of the teacher and student outputs and then take the L1 loss between the two gradient images.

   $$SobelLoss(X,Y) = L1(Sobel(X), Sobel(Y))$$

3. **Perceptual Loss**: Perceptual Loss(15) was first devised for the task of real-time style transfer. It makes use of the learning ability of a pre-trained VGG network enforce constrains on style and content between two images. For this task, we make use of the style component of perceptual loss. We pass both the student and teacher outputs through a pretrained VGG network. Then for layer l, the style loss is defined as:

   $$G^l_{ij}(I) = \textstyle\sum_k A^l_{ik}(I) A^l_{jk}(I)$$
   $$L^l_{style} = \tfrac{1}{M} \textstyle\sum_{ij} (G^l_{ij}(X) - G^l_{ij}(Y))^2$$

4. As a stretch-goal, the student network will be trained using GKA loss(14) function between the intermediate layers of the student and teacher networks. While it would be budgetarily infeasible to use the exact approach used in (14), it would be good to explore the benefits provided by just this loss function. The GKA loss function forces the intermediate activations of the teacher and student networks to be similar. Given a matrix $X \in \mathbb{R}^{n \times p1}$ and $Y \in \mathbb{R}^{n \times p2}$, the kernel alignment(18) is calculated as follows:

   $$KA(X,Y) = \frac{\|Y^T X\|_F^2}{\|X^T X\|_F \|Y^T Y\|_F}$$

Based on the above GKA(14) is defined as follows:

$$GKA(X,Y) = KA(\rho(X), \rho(Y))$$

Note that even the use of kernel-alignment based loss would require utilization of a lot of GPU memory, thereby restricting the size of the batch we can use. Another drawback of this loss function is that we will have to ensure that the spatial dimensions of student network match those of the teacher network for layer correspondence.

5. The final loss function is described as follows:

   $$AnimeLoss(X,Y) = w_1 SmoothL1(X,Y) + w_2 EdgeLoss(X,Y) + w_3 L_{style}(X,Y) + w_4 GKA(X,Y)$$

   The coefficients w_1, w_2, w_3 and w_4 are determined through experimentation to keep the magnitude of the different components of roughly the same order of magnitude.

Thus the distillation algorithm converts the Generative task to a supervised learning task using a pre-trained generator network.

4.3 Network Architecture

Table 1: Student Network Architecture

Input	Operator	n
$256^2 \times 3$	ConvBNRelu	1
$128^2 \times 16$	InvertedResidual	1
$128^2 \times 8$	InvertedResidual	1
$64^2 \times 16$	InvertedResidual	1
$32^2 \times 16$	InvertedResidual	3
$16^2 \times 32$	InvertedResidual	4
$16^2 \times 48$	InvertedResidual	3
$8^2 \times 80$	InvertedResidual	2
$8^2 \times 80$	ConvBNRelu + NNUpscale	1
$16^2 \times 48$	ConvBNRelu + NNUpscale	1
$32^2 \times 16$	ConvBNRelu + NNUpscale	1
$64^2 \times 16$	ConvBNRelu + NNUpscale	1
$128^2 \times 8$	ConvBNRelu + NNUpscale	1
$256^2 \times 3$	ConvBNRelu	1

Table 1 describes the architecture of the student network in detail. The architecture of the student network is based on the commonly used encoder-decoder pattern. In particular, the encoder for the network is based on a pre-trained MobileNet-V2(23) and the decoder is configurable with a choice between:

1. 5 modules of convolutions + 2x nearest neighbor up-samplings. This is based on a segmentation architecture open-sourced by Snapchat(3).
2. 5 modules of depthwise-separable-convolutions + 2x nearest neighbor up-samlings. This architecture is based on the FastDepth(28) mobile depth inference architecture.

The corresponding and symmetric layers of the encoder and decoder parts of the network are connected via **additive skip connections** to adequately convey higher level feature information to the decoder. **This student architecture has a size of 1.6 MB (vs 8MB of the original Teacher AnimeGAN-V2 generator)**. This is an 8x reduction in size.

5 Experiments

The sections below describe the experiments and training methodology. All source code for experiments can be found at this link. The custom dataset generated for this project can be downloaded from S3 at this link. Before downloading, please note that the dataset is 16GB in size.

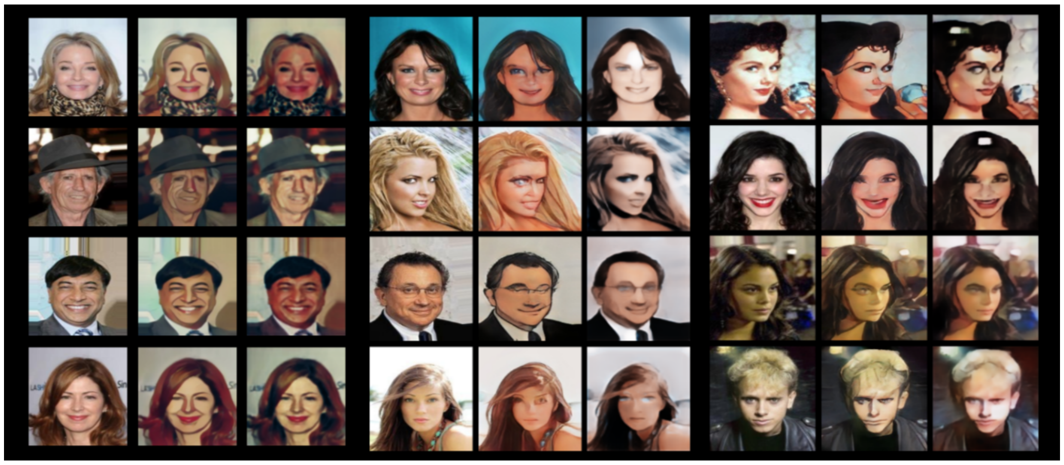

Figure 1: Comparing teacher model vs student model for paprika, webtoon and face v2 styles respectively. Each grid of 4x3 consists of the input image as the first column, followed by the teacher network output column followed by student network output column. The paprika style in the first grid has the best performance because most hyperparameters were tuned for this style during majority of the project.

5.1 Training Methodology

The first step in experimentation was to generate a dataset using the pre-trained AnimeGAN-V2(20) network as well as the CelebA(21) and Flickr(9) datasets. This paired dataset consists of > 216K paired images. Instead of generating the activations of the teacher network on-the-fly, we cache the outputs to disk in order to preserve GPU memory while training. Training the student network on this generated dataset is equivalent to forcing student activations to align with the larger teacher network activations.

In order to quickly evaluate some of the network's hyperparameters, a **subset of the paired dataset consisting of 16K images** was used for evaluating network performance. The hyperparameters resulting from this experiment were used to train the student architecture on three different anime styles. Each experiment was run for 300,000 batches (100 epochs through the data) till the loss converged. It is worth noting however that majority of the hyperparameter tuning was performed for the Paprika dataset (and thus it has the best overall performance). The other two styles were trained as stretch goals.

The best learning rate for training was determined using an LR Finder(25)(24). Other hyperparameters were fixed based on initial experimentation performed on the subset of data mentioned above. The primary training was done with this set of parameters. Also during this initial experimentation, it became clear that use of GKA loss with the larger teacher network would severely stress the GPU memory and force the adoption of a very small batch size which hurt performance. Thus instead of GKA loss, a perceptual loss(15) was used instead.

5.2 Ablation Studies

The primary ablations were performed on the space of loss functions. The final form of Anime Loss mentioned in the Loss Functions section above was the result of progressive experimentation using different losses.

The performance of the trained network was evaluated on the basis of FID score. First, the teacher Anime GAN-V2 was run on two randomly generated samples from the face dataset. The ground-truth baseline is the FID score between these two samples which represent the same distribution. This lower-bounds the FID score. On the other end of the spectrum, we also measure the FID score against a data sample comprising of a completely different anime style. This represents an out-of-domain dataset which is an upper-bound on the FID score.

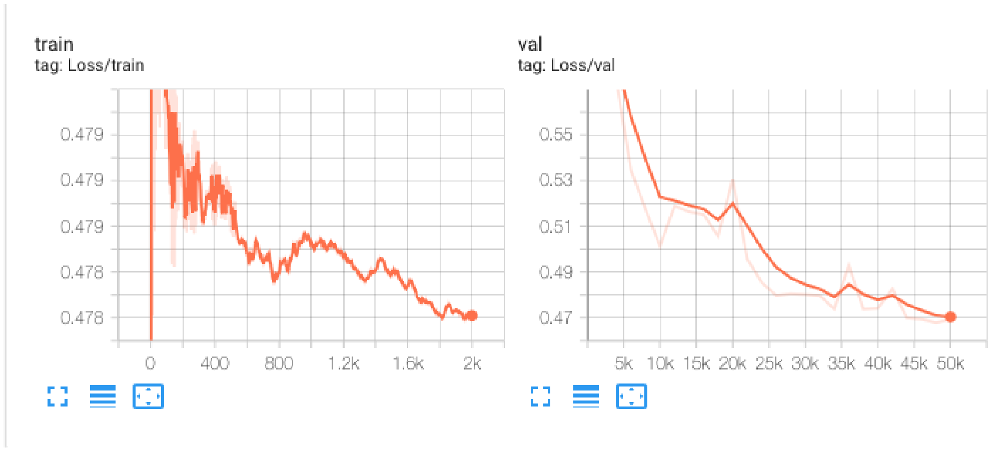

Figure 2: Tensorboard Training and Validation Combined Loss Curves for Paprika Style

Table 2: FID Scores against Ground Truth for Paprika, Webtoon and Face Portrait V2 anime styles

Model	Ground Truth	Out-of-Domain	L1	L1,Sobel	L1,Sobel,Perceptual
Paprika	1.6	82	47	37	17
Webtoon	1.24	124	44	32	29
Face V2	1.61	57	40	35	24

As can be seen from table 2, the network trained with all components of the loss performs best when evaluated against the ground-truth data distribution using FID. Given the result of this study, we continue to train two more networks which approximate the Webtoon and Face Portrait styles respectively. It is worth noting though that the different styles of anime perform slightly differently for the same hyperparameters. **In particular, since majority of the experimentation was done with the Paprika style, all hyperparameters are tuned to that style and it performs the best out of the three styles. The other two styles were trained as a stretch goal.**

5.3 Results: Comparison of MACs and FPS

Using the open-source PyTorch Op-Counter(29) tool to profile the number of parameters and MACs (multiply-accumulate operations) in the teacher Anime-GAN-V2 and the trained output model. **The teacher model has 26 Billion MACs and 2 Million parameters**. In contrast, **the student model has 130 Million MACs and 373 Thousand paramaters**. This is a **significant reduction in compute and size** while maintaining the overall quality of generated output.

Table 3: FPS Comparison on Mobile Devices

Model	Huawei P8 Lite	Samsung Galaxy S7	iPhone XR	iPhone X	Pixel 6 Pro
Teacher	1	5	10	15	14
Student	18	25	30	30	30

For comparing frames-per-second on real devices, we make use of Snapchat's SnapML(4) toolkit which is part of the overall Lens Studio(2) set of tools which allows the community to deploy ML models to mobile-devices in a point-and-click manner. For this test, it was sufficient to modify simple existing templates. Table 3 shows the comparison of the FPS across a range of devices. Given the comparison of MACs and parameters, it is no surprise that the student model is much faster than the teacher model. However, in addition, it is clear that the student model maintains a stable 20 FPS even on phones on the lower end of the Android spectrum like Huawei P8 Lite. It is also worth mentioning that it was extremly hard to convert the teacher model to an ONNX format which was runnable on-device as it uses some advanced PyTorch patterns and operators like GroupNorm which were not

Figure 3: Snapcodes for paprika, webtoon and face v2 styles respectively. Please the snacode of choice in Snapchat to try the student model.

supported on a lot of mobile frameworks. The GroupNorm layer was replaced by InstanceNorm in order to work on device. Thus another benefit of the distillation is to make the application readily available on a range of mobile devices and mobile-frameworks.

Note that anyone with access to the Snapchat app can try out the models within Snapchat as lenses by scanning the snapcodes in figure 3

6 Analysis

The Experiments and Results above show that it is possible to distill a Student GAN given a much larger and carefully trained teacher network. The approach to teacher-student training can thus be simplified to make the hyperparameters more stable. From the qualitative and quantitative results above, we can make some observations:

1. Due to its small size, the student generator is never completely able to approximate the teacher network. This can be seen in the small artifacts in the student outputs in the comparison image in figure 1. It may be possible to remove such artifacts, but it requires a lot more experimentation with training data.
2. In general, FID scores for data from the same domain tend to be quite low (although there is some variance depending on the dataset). If the baselines are to be used, there is a lot of room for improvement in results based on analysis in table 2. In particular, it may be possible to perform better with much larger batch sizes on a higher GPU memory.
3. While FID score is not comparable across datasets, from table 2, it is clear that the network comes closest to ground truth baseline for the paprika style. This is because paprika style was trained first and most hyperparameters were tuned for it. It is possible to get better results for all styles with a slightly wider network (3MB, 1.5MB when compressed).
4. Even with very large datasets like CelebA, the student model can both overfit as well as fail to generalize to images in the wild. This became more clear upon testing the model in the wild on real-data. The student model overfits slightly to work best on images which look like professional portraits. Despite mixing the data with small parts of Flickr faces dataset, it is clear that a lot more diversity in training data is required for best results.
5. The choice of loss functions is extremely important in such a distillation task. Just the addition of perceptual and sobel losses to the training dramatically improve the results of FID comparisons.
6. It is worth noting that the student model is 1.5 MB in size (700KB compressed) which is a 5x reduction in size from the teacher model (8MB). It may be possible to add more parameters to the student model to improve fidelity while not sacrificing performance.

7 Conclusion

This project attempted to compress large teacher Generator networks into significantly smaller student networks which approximated the outputs of the larger teacher networks. Doing so would allow us to use the smaller networks in a compute and energy efficient manner on commodity, mobile and embedded hardware. After the experimentation and the analysis done in this project, we can conclude the following:

1. While it is possible to compress larger teacher networks into smaller student networks which are much more compute efficient, **doing so requires a very large and diverse set of training data**. As was discovered during testing the student model in the wild, despite the use of the entire CelebA dataset, the student models still overfit slightly to images in the training data.
2. It is extremely important to use the right choice of loss functions for distillation tasks. Combining different loss functions, each of which attend to some shortcoming or characteristic of the student network output may produce qualitatively superior results.
3. Choice of coefficients when weighing different loss functions is important or else the larger-magnitude loss functions can dominate the smaller ones.
4. Training small and simple student generator networks may make the application available across a lot more classes of hardware where newer, more advanced neural network operators are not supported.
5. A small student network may still produce artifacts after being trained. It may be possible to remove such artifacts with a slightly larger network but this requires further experimentation.
6. When distilling student networks, there is a slight trade-off between computational complexity of student vs the training infrastructure. With a sophisticated training infrastructure with powerful and fast GPUs with a lot of memory, it is possible to use more advanced loss functions (perceptual losses and GKA) which are compute intensive but help guide the student in a very targeted way. Alternatively, we can use a slightly larger student network which has more capacity and thus make use of simpler per-pixel/gradient loss functions.

7.1 Future Work

Due to the time and budgetary constraints of this class, a lot of ideas were left unexplored which may improve the fidelity of the final trained output. Going forward, these are some of the ideas which can be explored.

1. **Much larger mixed datasets**: Despite usage of the entirety of CelebA dataset and parts of flickr-faces dataset, the student model still showed signs of overfitting. It should be possible to repeat this experiment with a much larger mixed dataset to address generalization issues with the student.
2. **Exploration of Kernel Alignment Losses**: Unfortunately, it was not possible to use both GKA loss(14) and Perceptual loss(15)(15) together. Doing so was too much for GPU memory and would require an extremely small and ineffectual batch size. However, given a bigger GPU like an NVIDIA A100, this may be possible and may produce better results.
3. **Region based losses**: The student network sometimes produced artifacts or was unable to get small details like eyes accurate. This may be rectifiable by using a stronger L2 loss around the hard-to-generate regions. While such a pipeline is easy to generate using a face tracking library like dlib(16), it would not support batching and would thus require pre-caching of facial keypoints of interest on disk. However, such an effort would require a lot more training time and compute resources. Alternatively, we may use a neural-face tracker which can efficiently process large batches of data.

8 Acknowledgements

There was a wealth of prior research, engineering techniques, open-source software and frameworks without which this project would not have been possible. In particular:

1. The original authors of AnimeGAN-V2(6)(7) and its PyTorch port(20) provided the teacher model architectures and their trained weights for various anime styles.
2. The Teachers Do More than Teach(14) paper presents a thorough analysis of what it takes to compress large generators into mobile optimized models.
3. SnapML Templates(4)(26)(3) provide true and tested mobile architectures which formed the initial backbone of this investigation.
4. The open-sourced PyTorch FID library(22) provides a well-tested and easy-to-use implementation of FID which formed the basis of qualitative and quantitative benchmark with which to judge the performance of student models trained with various methods.
5. The LensStudio(2) community and developers who built SnapML(4) which allows for seamless point-and-click experimentation of machine learning models on real devices. The various templates and documentation were an enormous resource for real-world testing of model performance.
6. Last, but not the least, the course staff of CS236 Fall 2021 for their guidance through multiple steps of this project.

9 Appendix

1. **All code** to run experiments carried out in this project can be found at this github repository.
2. **The custom datasets** used to train models for this project can be found at this public S3 link
3. **The lens studio** projects used to test on-device can be found here.

References

[1] Smooth l1 loss. https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html.
[2] Snapchat Lens Studio. https://lensstudio.snapchat.com/.
[3] Snapchat Lens Studio Segmentation. https://lensstudio.snapchat.com/templates/ml/custom-segmentation/.
[4] Snapchat Lens Studio SnapML. https://lensstudio.snapchat.com/guides/machine-learning/ml-overview/.
[5] S. Belousov. Mobilestylegan: A lightweight convolutional neural network for high-fidelity image synthesis.
[6] A. Chan. Anime gan v2. https://github.com/TachibanaYoshino/AnimeGANv2, 2020.
[7] J. Chen, G. Liu, and X. Chen. *AnimeGAN: A Novel Lightweight GAN for Photo Animation*, pages 242–256. 05 2020.
[8] D. P. E. R. E. Riba, D. Mishkin and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.
[9] Flickr. Flickr faces dataset. https://github.com/NVlabs/ffhq-dataset, 2019.
[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks.
[11] 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.
[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network.
[13] 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.
[14] Q. Jin, J. Ren, O. J. Woodford, J. Wang, G. Yuan, Y. Wang, and S. Tulyakov. Teachers do more than teach: Compressing image-to-image models.
[15] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution.
[16] D. E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.
[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.
[18] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited.
[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
[20] B. D. Lee. Anime gan v2 pytorch. https://github.com/bryandlee/animegan2-pytorch, 2021.
[21] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
[22] mseitzer. Pytorch fid. https://github.com/mseitzer/pytorch-fid, 2018.
[23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks.
[24] D. Silva. Pytorch lr finder. https://github.com/davidtvs/pytorch-lr-finder, 2020.
[25] L. N. Smith.
[26] Snapchat. Snapml templates. https://github.com/Snapchat/snapml-templates, 2021.

[27] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[28] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze. Fastdepth: Fast monocular depth estimation on embedded systems.

[29] L. Zhu. Thop: Pytorch-opcounter. https://github.com/Lyken17/pytorch-OpCounter, 2018.