

SKDCGN: Source-free Knowledge Distillation of Counterfactual Generative Networks using cGANs

- Supplementary Material

Anonymous ECCV submission

Appendix

A Details of the different models

A.1 Original CGN architecture

This section contains a diagram of the original CGN architecture, as presented in [1].

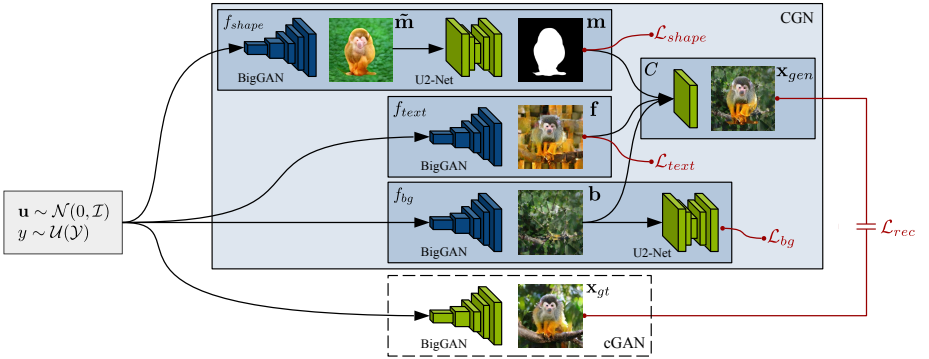


Fig. 1: CGN architecture diagram. Retrieved from [1].

Figure 1 illustrates the CGN architecture. The network is split into four mechanisms, the shape mechanism f_{shape} , the texture mechanism f_{text} , the background mechanism f_{bg} , and the composer C . Components with trainable parameters are blue, components with fixed parameters are green. The primary supervision is provided by an unconstrained conditional GAN (cGAN) via the reconstruction loss \mathcal{L}_{rec} . The cGAN is only used for training, as indicated by the dotted lines. Each mechanism takes as input the noise vector \mathbf{u} (sampled from a spherical Gaussian) and the label y (drawn uniformly from the set of possible labels \mathcal{Y}) and minimizes its respective loss (\mathcal{L}_{shape} , \mathcal{L}_{text} , and \mathcal{L}_{bg}). To generate a set of counterfactual images, we sample \mathbf{u} and then independently sample y for each mechanism.

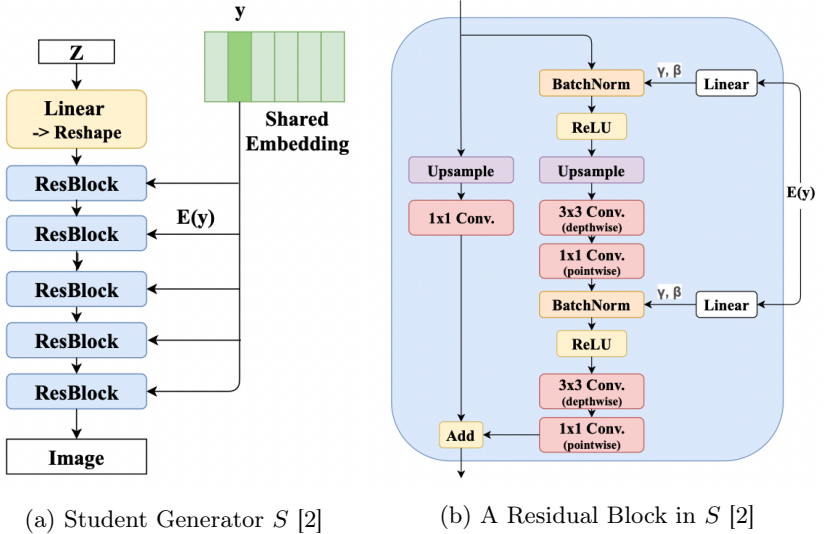


Fig. 2: Architecture of the TinyGAN (student) generator

A.2 TinyGAN architecture

This section provides an brief overview of the TinyGAN architecture. For more details, refer to [2].

Generator. As shown in Figure 2, TinyGAN comprises a ResNet [3]-based generator with class-conditional BatchNorm [4] [5]. To keep a tight computation budget, it does not adopt attention-based [6] or progressive-growing mechanisms [7]. To substantially reduce the model size compared to BigGAN, it:

- Relies on using fewer channels;
- Replaces standard convolution by depthwise separable convolution;
- Adopts a simpler way to introduce class conditions.

Overall, TinyGAN’s generator has $16\times$ less parameters than BigGAN’s generator.

Discriminator. Following [8] [9], [2] opt for spectral normalized discriminator and introduce the class condition via projection. But instead of utilizing complicated residual blocks, they simply stack multiple convolutional layers with stride as used in DCGAN [10], which greatly reduces the number of parameters.

Overall, TinyGAN’s discriminator has $10\times$ less parameters than BigGAN’s discriminator.

A.3 Baseline model

The baseline is a standard CGN architecture whose BigGANs have been replaced with TinyGANs. Due to the need of a pre-trained model that (i) supervises the

CGN training using a reconstruction loss and (ii) serves as the initialization of the IM GANs, a TinyGAN was trained from scratch using the KD strategy described in [2]. In this section we present qualitative results of both the newly-trained TinyGAN and of baseline model.

B SKDCGN

B.1 TinyGAN training data for SKDCGN

Implementing the architecture of the SKDCGN in Figure 1 in the main paper requires training a TinyGAN for each Independent Mechanism of the CGN. To this end, we extract each IM backbone (BigGAN + U2-Net for *shape*, BigGAN for *texture*, BigGAN for *background*) from the CGN architecture, then use each of them as a black-box teacher for our student. The KD training procedure, however, requires training data. Hence prior to training, 1000 images per class (totalling 1 million samples) were generated using each IM backbone extracted from the pre-trained CGN provided by [1].

B.2 Generated Counterfactuals

Here we compare the counterfactuals generated from the SKDCGN model with the ones produced from CGN. Refer to Figure 3 for visualization.

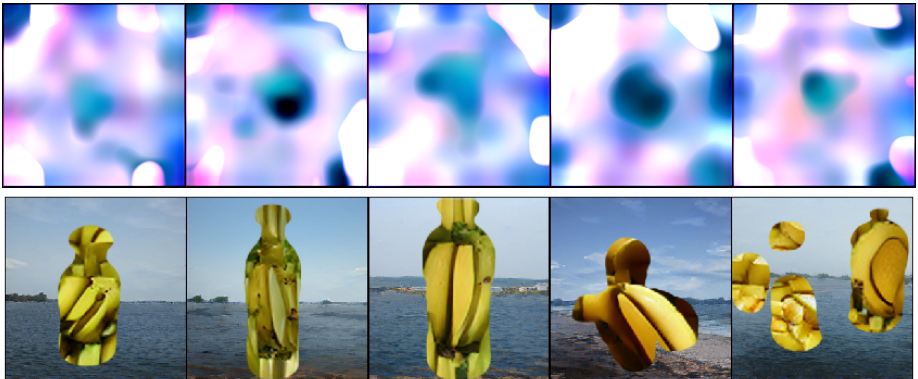
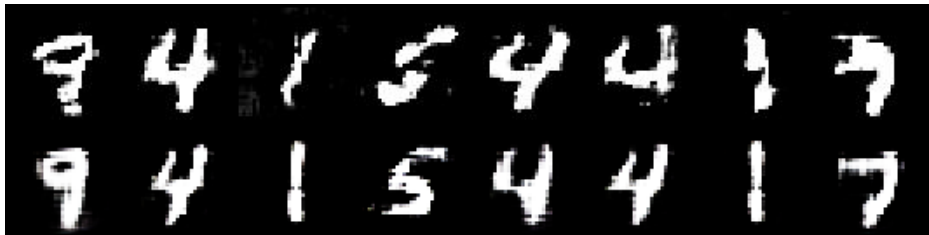


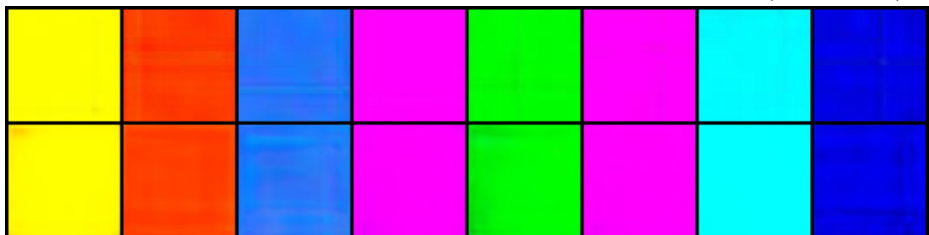
Fig. 3: Images generated by SKDCGN with three independently trained TinyGANs (top), and the original CGN architecture (bottom)

B.3 Results obtained on MNIST dataset on SKDCGN model

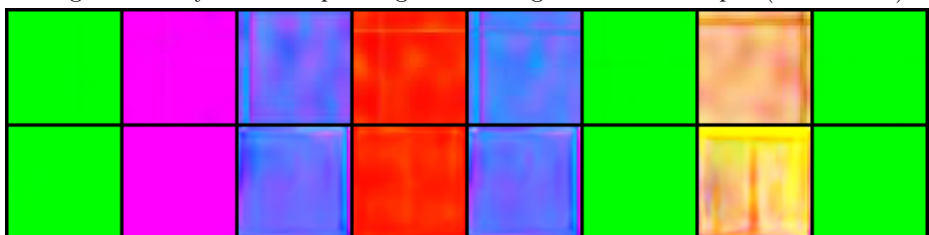
In this section, we present the results obtained on MNIST dataset on the proposed architecture (SKDCGN) using default parameters. For visualization refer to 4.



(a) A comparison of images generated by the CGN **shape** backbone (*top row*) and those generated by the corresponding SKDCGN given the same input (*bottom row*).



(b) A comparison of images generated by the CGN **texture** backbone (*top row*) and those generated by the corresponding SKDCGN given the same input (*bottom row*).



(c) A comparison of images generated by the CGN **background** backbone (*top row*) and those generated by the corresponding TinyGAN given the same input (*bottom row*).

Fig. 4: A comparison of images generated by the CGN backbones and those generated by the corresponding SKDCGN (given the same input) for each independent mechanism.

C Baseline Model

C.1 Training Details

The training procedure of a CGN requires a pre-trained GAN to provide primary supervision via the reconstruction loss. However, the original TinyGAN was only trained on only animal classes, hence the publicly-available model could not be used for our baseline. In order to consistently use the same dataset for all the experiments, we re-trained a TinyGAN from scratch (as described in [2]) on all classes of ImageNet-1k. The images generated by TinyGAN are visualized in Appendix C.1. The images generated for each Independent Mechanism using our

baseline model can be seen in C.1. Apart from this, we additionally generated the counterfactuals using the baseline model which are shown in Appendix C.1.

Generated outputs of TinyGAN trained on ImageNet-1k A TinyGAN was trained using all 1000 classes of the ImageNet-1k dataset. Training details are provided by [2]. Although the original paper trains the model for 1.2 million epochs, we are forced to restrict the amount of iterations due to computational constraints. After distilling the knowledge of a BigGAN for 18 epochs, our TinyGAN generates reasonable images, as seen in Figure 5b. To compare the image generation we have also presented images generated after the first epoch as well 5a. It can be observed that if we further train the model, it could produce images better in quality. Note that animal classes are better captured by the model: this is inline with the findings of [2].



(a) A comparison of images generated by BigGAN and the TinyGAN. Images in top row are produced by BigGAN, while those in bottom row are by SKDCGN given the same input after 1st epoch.



(b) A comparison of images generated by BigGAN and the TinyGAN. Images in top row are produced by BigGAN, while those in bottom row are by SKDCGN given the same input after 18th epoch.

Fig. 5: A comparison of images generated by BigGAN and the TinyGAN. Images in top row are produced by BigGAN, while those in bottom row are by SKDCGN given the same input

Generated outputs of the baseline trained on ImageNet-1k Figure 6 illustrates the individual outputs of each IMs at the starting, after epoch $300k^{th}$, epoch $600k^{th}$, epoch $900k^{th}$, and epoch $1.2million^{th}$ (from left to right). In each figure, we show from top to bottom : pre-masks \tilde{m} , masks m , texture f , background b , and composite images x_{gen} .

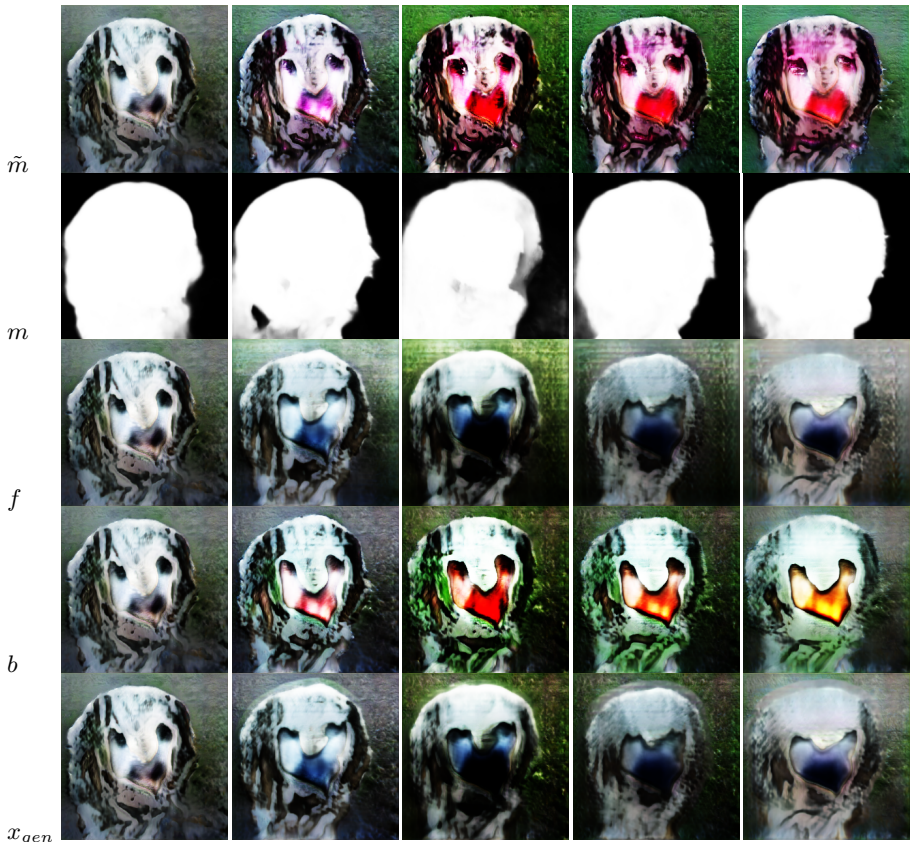


Fig. 6: Individual IM Outputs after training for baseline. From top to bottom: m , \tilde{m} , f , b , x_{gen} . From left to right: at the start of training, after epoch $300k^{th}$, epoch $600k^{th}$, epoch $900k^{th}$, and epoch $1.2million^{th}$

Generated Counterfactual Images of Baseline trained on ImageNet-1k
Finally, we show counterfactual images generated by the baseline model in Figure 7.



Fig. 7: Counterfactuals generated by baseline on test data for ImageNet-1k

D Improving the SKDCGN process

As mentioned in section improvement, we observed that the outputs from CGN are noisy in nature , and a teacher can only be as good as student. Fig 8 evidently describes how noisy the MNIST digits are. This is the reason we observe several artefacts in our architecture as well. However in this section we try to improve our architecture by several methods.

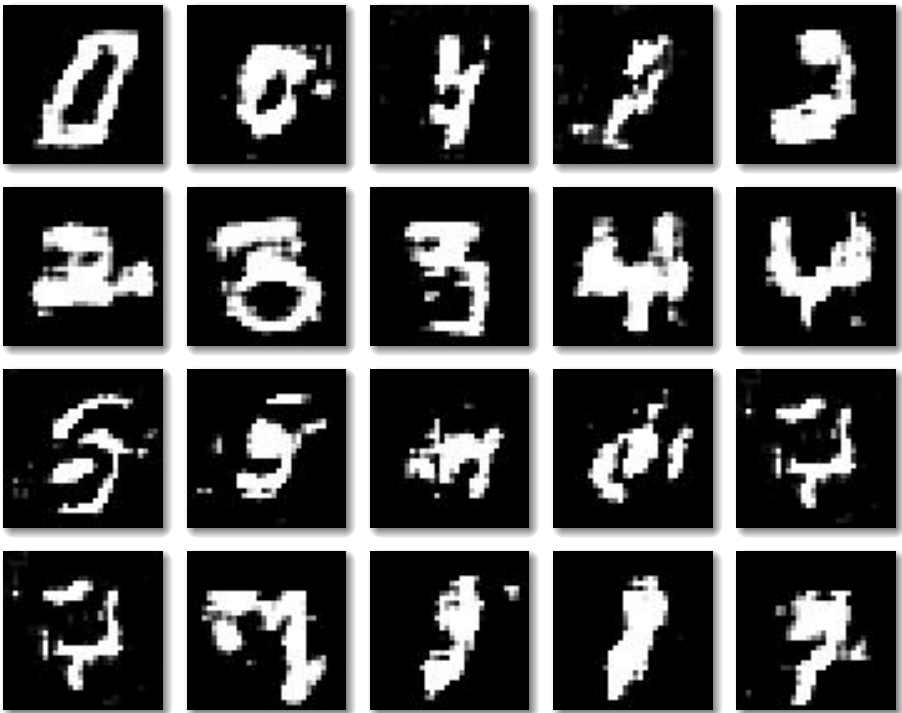


Fig. 8: Noisy outputs generated by the CGN when we made use of pretrained weights given by the authors.

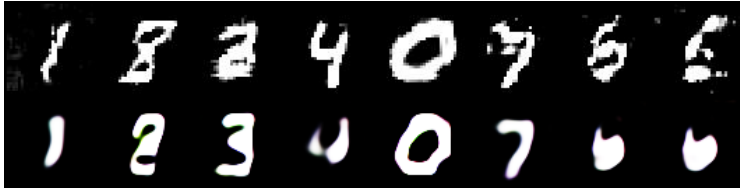
In the direction towards improving the images that are being generated by our architecture, we strongly believe the room of improvement lies in these components:

- Improving the quality of images that are being generated by the GAN network in our architecture. Usually loss functions like VGG based perception loss, L1 reconstruction loss are added.
- Improving the existing knowledge distillation framework such that the student learns better from the teacher’s guidance by adding new loss functions to the Knowledge Distillation task.

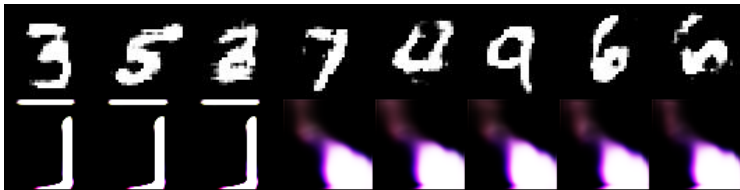
To improve the quality of images, we observe that our architecture already has most of the loss functions integrated implicitly/explicitly. Hence, we add the Cross entropy loss for the generator and discriminator for the mask IM of the architecture and get the results as shown in 9a for second epoch. We observe that digits like '0' are being reconstructed however for other digits the inputs look noisy in nature. By the end of 10th epoch for test set in Fig. 9b we observe that the digits are being reconstructed. We continue with the training since we expected better results than what we have already seen, however, contrary to our beliefs we observe artefacts by the end of 30th epoch as shown in Fig. 9c.



(a) A comparison of images generated by the CGN **shape** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) after 2 epochs on test data.



(b) A comparison of images generated by the CGN **texture** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) after 10 epochs on test data.

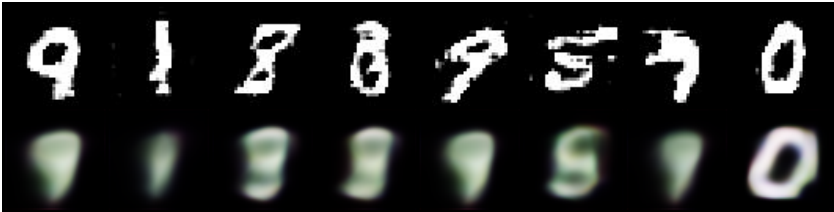


(c) A comparison of images generated by the **background** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) after 30 epochs on test data.

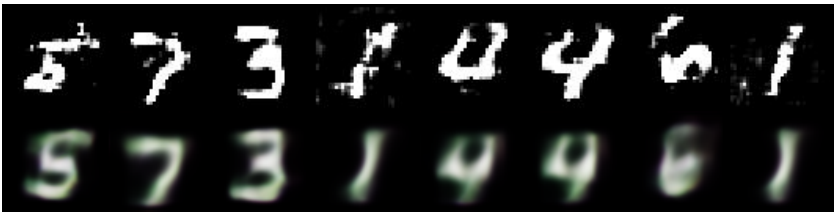
Fig.9: A comparison of images generated by the CGN backbones and those generated by the corresponding SKDCGN (given the same input) mask IM with cross entropy loss

D.1 KL multiplied with layer instead of L1

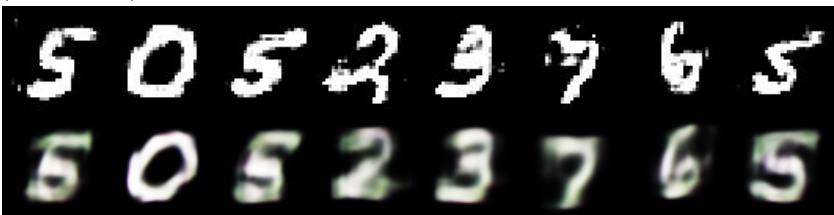
Since the image generation process already has most of the components to ensure that the reconstruction is in place, we tried to improve the Knowledge distillation between teacher and student network by integrating the KL divergence and multiply the loss with every layer of the network instead of L1 which is default. Possibly, because L1 reconstruction loss is explicitly needed that is to multiplied with the activation of every layer. We observe the results as shown in Fig. 10



(a) A comparison of images generated by the CGN **shape** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) after 2 epochs on test data.



(b) A comparison of images generated by the CGN **texture** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) after 10 epochs on test data.

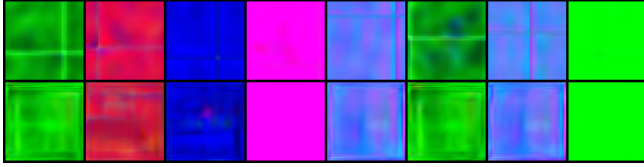


(c) A comparison of images generated by the **background** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) after 30 epochs on test data.

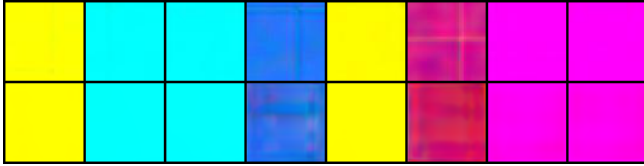
Fig. 10: A comparison of images generated by the CGN backbones and those generated by the corresponding SKDCGN (given the same input) mask IM with KL divergence multiplied with the activation of every layer instead of L1

D.2 KL divergence between teacher and student outputs

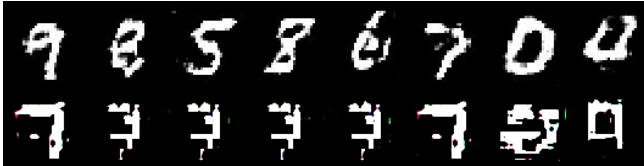
Since KL can't be a substitute for L1 loss to be multiplied with activations of every layer, we now modify the Knowledge distillation loss to be: L1 loss multiplied with the activation of every layer and computer KL divergence between the teacher's outputs and students outputs separately and do not multiply it with the activation of the layers anymore. We observe that we get better results by doing this for background and foreground however the results for mask IM lose shape after few epochs. We use KL divergence because leads to entropy minimization between the teacher and student and we obtain results as shown in 11.



(a) A comparison of images generated by the CGN **shape** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) for Foreground IM after 30 epochs on test data.



(b) A comparison of images generated by the CGN **texture** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) for background IM after 30 epochs on test data.



(c) A comparison of images generated by the **background** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) for mask IM after 30 epochs on test data.

Fig. 11: A comparison of images generated by the CGN backbones and those generated by the corresponding SKDCGN (given the same input) mask IM with KL divergence between teacher and student and L1 multiplied with the activation of every layer.

Since this approach gave us better results for several IMs we use the same approach to generate results for ImageNet-1k dataset, and obtain better results. From Fig. 12 we observe that with KL divergence we are able to converge faster than without it.

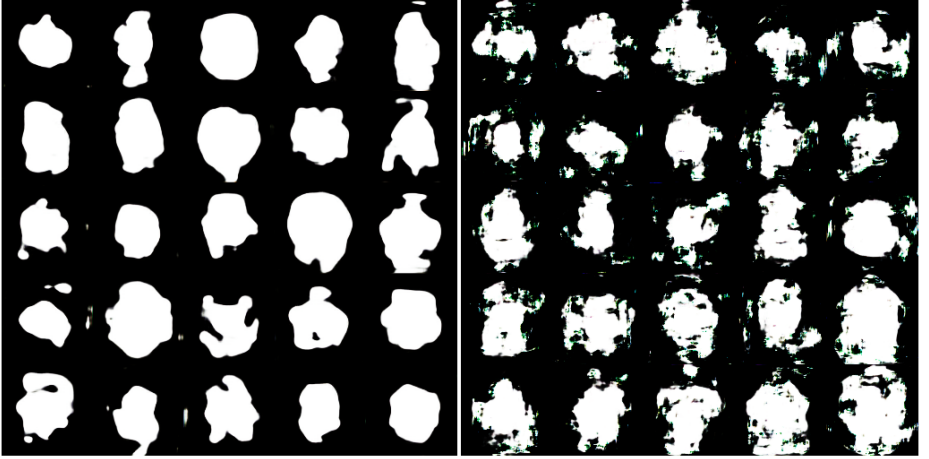
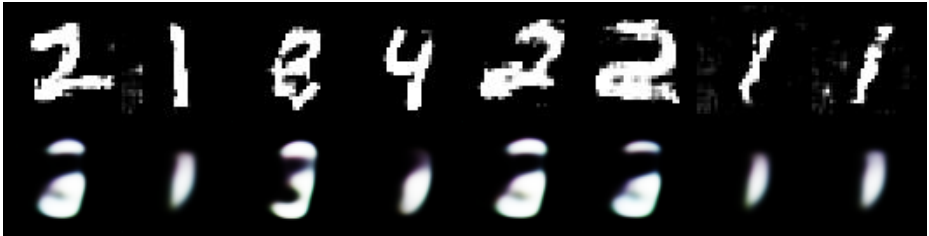


Fig. 12: Left: Mask outputs obtained after 1st epoch of Imagenet1k dataset by SKDCGN and Right: mask outputs obtained after 23rd epoch of Imagenet1k dataset by SKDCGN. Evidently, we observe that by adding KL Divergence we get better results right after 1st epoch, whereas we do not get similar outputs even after 23rd epoch without it.

D.3 MSE instead of L1

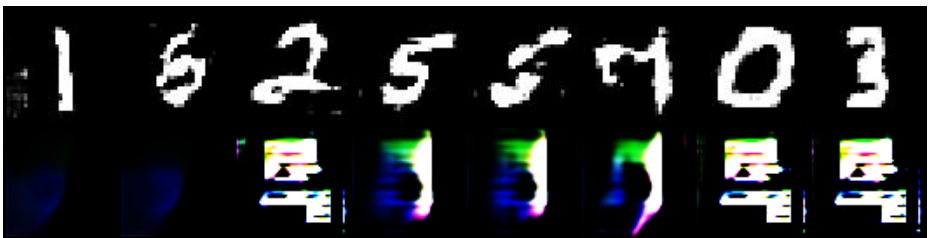
In addition, We also tried L2 loss instead of L1 loss but it lead to noisy outputs than previously generated and obtain results as shown in 13. Since, L2 assumes that the influence of noise is independent of the image's local characteristic the images are noisy in nature.



(a) A comparison of images generated by the CGN **shape** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) for mask IM after 2 epochs on test data.



(b) A comparison of images generated by the CGN **texture** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) for mask IM after 10 epochs on test data.



(c) A comparison of images generated by the CGN **background** backbone (*top* row) and those generated by the corresponding SKDCGN given the same input (*bottom* row) for mask IM after 30 epochs on test data.

Fig. 13: A comparison of images generated by the CGN backbones and those generated by the corresponding SKDCGN (given the same input) mask IM with L2 multiplied with the activation of every layer instead of L1.

References

1. Axel Sauer and Andreas Geiger. Counterfactual generative networks. *CoRR*, abs/2101.06046, 2021.
2. Ting-Yun Chang and Chi-Jen Lu. Tinygan: Distilling biggan for conditional image generation. *CoRR*, abs/2009.13829, 2020.
3. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 630–645, Cham, 2016. Springer International Publishing.
4. Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style, 2016.
5. Harm de Vries, Florian Strub, Jérémie Mary, H. Larochelle, Olivier Pietquin, and Aaron C. Courville. Modulating early visual processing by language. In *NIPS*, 2017.
6. Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2018.
7. Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2017.
8. Takeru Miyato and Masanori Koyama. cgans with projection discriminator, 2018.
9. Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.
10. Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.