Icon Generation with Conditional Generative Adversarial Networks

Diogo Medeiros
School of Science and Technology
University of Trás-os-Montes and Alto Douro
Vila Real, Portugal
al70633@alunos.utad.pt

***Abstract*—Conditional Generative Adversarial Networks (cGANs) were used to generate icons with specific characteristics or properties given the Icons-50 dataset. The performance of the cGANs was impacted by the imbalanced nature of the original dataset. Filtering the original dataset to include only the 10 most frequent classes (the Icons-10 dataset) improved the performance of the cGANs, allowing them to generate more detailed and striking icons. This paper demonstrates the potential of cGANs for use in applications where a large number of icons with specific characteristics are needed.**

***Index Terms*—GAN, cGAN, Icons-50, icons, style, applications**

I. Introduction

A. Background

Conditional Generative Adversarial Networks (cGANs) [1] are a variant of Generative Adversarial Networks (GANs) that allow the generation of new data samples with specific properties or characteristics. They consist of two neural networks: a generator and a discriminator. The generator is trained to generate new data samples that are similar to a training set, while the discriminator is trained to distinguish between real and fake data samples.

cGANs introduce an additional input to the generator and discriminator networks, known as the conditioning variable, which allows the model to generate data with specific characteristics or properties. This can be useful in a variety of applications, such as image generation, text generation, and speech generation, where the output data needs to have specific attributes or features [1].

For example, in image generation, the conditioning variable may be a label indicating the class of the image to be generated (e.g., a specific type of animal or object), or it may be an attribute of the image (e.g., the presence or absence of a certain color).

B. Problem

Given a public dataset, Icons-50 [2], [3], comprised of icons and their respective labels, we want to be able to generate a new set of sample icons corresponding to a specific label, such as "clock" or "airplane".

This could be useful in applications where a large number of icons with specific characteristics are needed, such as in user interface design or graphics design.

C. Objective

Using cGANs, it is possible to generate icons with specific characteristics or properties, given a public dataset such as Icons-50. To do so, the cGAN would be trained on said dataset, with the conditioning variable being the desired label of the generated icons, in this case represented by an integer value corresponding to a class of icons, such "cat" or "bird". [1]

D. Related Work

In Mirza and Osindero [1] , the authors introduce the concept of conditional generative adversarial networks (cGANs), which are a variant of generative adversarial networks (GANs) that allow the generation of new data samples with specific properties or characteristics. The cGANs are constructed by feeding the data and the desired characteristics or properties to both the generator and discriminator networks. The authors demonstrate the use of cGANs for generating MNIST digits conditioned on class labels and for learning a multi-modal model. They also provide examples of using cGANs for image tagging, where the model is able to generate descriptive tags that are not part of the training labels.

In Liu et al. [4] , the authors propose a cGAN-based method for repairing damaged fonts based on style. They use the content accuracy and style similarity of the repaired font as evaluation indices to assess the accuracy of the restored style. The results show that the cGAN-based method is able to repair damaged fonts in a way that is similar to the correct content.

In Loey et al. [5] , the authors propose the use of classical data augmentation techniques along with a cGAN based on a deep transfer learning model for COVID-19 detection in chest CT scan images. The motivation for this research was the limited availability of benchmark datasets for COVID-19, particularly in chest CT images. To address this, the authors collected all available images for COVID-19 and used classical data augmentation techniques along with a cGAN to generate additional images to help in the detection of the virus. The results showed that the classical data augmentation techniques along with the cGAN improved the performance of classification in all selected models.

[1] Source code and trained models are available at https://github.com/Necas209/Icon-Generation-cGANs

In Ma et al. [6] , the authors propose an end-to-end framework based on a cGAN for simultaneously reducing speckle noise and enhancing contrast in retinal OCT images. The cGAN is trained using a novel method for obtaining clean images from outputs of commercial OCT scanners, and an edge loss function is added to the final objective to ensure that the model is sensitive to edge-related details. The results show that the proposed method outperforms traditional and other deep learning methods in terms of denoising performance, and has good generalization ability for different types of retinal OCT images.

II. Materials and Methods

A. Data

To achieve our goal, we opted for the public dataset "Icons-50" [2] , first introduced in [3]. According to the authors, this dataset was initially designed to test the robustness of classifiers to surface variations in objects. The dataset includes both new styles of known objects and unexpected instances of known classes, and the authors propose two methods to improve the surface variation robustness of classifiers.

The Icons-50 dataset consists of 10,000 images belonging to 50 classes of icons (e.g., people, food, activities, places, objects, symbols, etc.) collected from different technology companies and platforms (e.g., Apple, Samsung, Google, Facebook, etc.). Each class has icons with different styles (e.g., Microsoft's flat vector graphics icon style) and different class subtypes (e.g., 'duck' or 'eagle' subtypes in the 'birds' class) [2].

To summarize, each entry in the dataset consists of the following parameters:

- image - a 3x32x32 NumPy array corresponding to the RGB icon
- class - an integer between 0 and 49 representing the icon's class
- subtype - a string corresponding to the class subtype, e.g., "broken_heart" for the class "heart" (label 26)
- style - a string corresponding to the icon's source, such as Microsoft or Apple
- rendition - an integer between 0 and 9 representing the icon's version

When analyzing the class distribution, we came to the conclusion that the Icons-50 dataset was considerably unbalanced, as seen in Fig. 1.

This led to the creation of Icons-10, a subset of Icons-50 containing the top 10 most frequent classes in the dataset, with the old labels mapped to new values between 0 and 9. The Icons-10 dataset comprises 4170 images and has a fairly balanced class distribution, seen both in Fig. 2 , and in Table I.

B. Methods

We opted to adapt an already existing implementation of a cGAN model using the `keras` library, with the original source code available in [7], [8]. In the original post, the author chooses to train the model with the MNIST Fashion dataset, which comprises gray-level images with 28x28 pixels, far different from the Icons-50 dataset specifications.

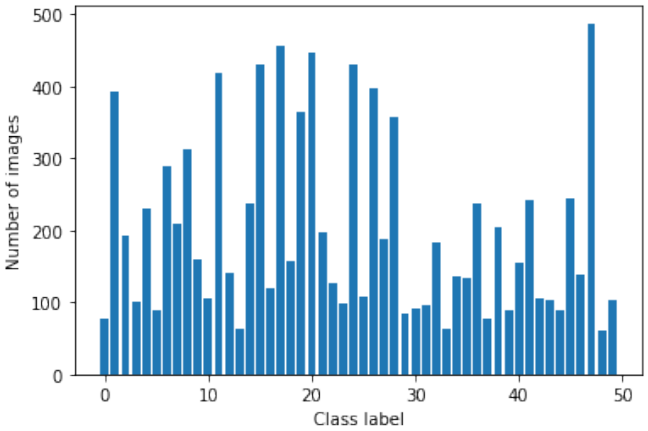


Fig. 1: Class distribution of the Icons-50 dataset

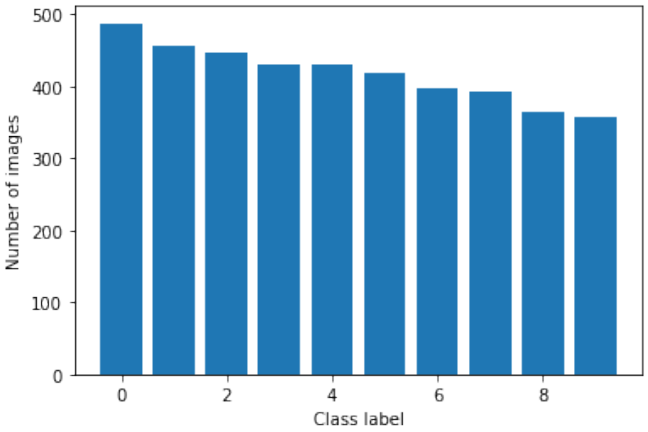


Fig. 2: Class distribution of the Icons-10 dataset

In the case of a regular GAN, in Fig. 3, the generator takes in the input vector randomly drawn from the latent space and outputs the generated image, which is fed to the discriminator, alongside real examples from our dataset, and outputs a classification: whether it is real.

The main difference between the regular GAN and the cGAN models can be seen in Fig. 4. The conditioning variable, in our case, is the icon's class label, which is fed both to

TABLE I: Icons-10 Dataset

Class	Old label	New label	No. samples
worker	47	0	487
family	17	1	455
flag	20	2	446
face	15	3	429
hands	24	4	429
clock	11	5	417
heart	26	6	396
arrow	1	7	392
cat	19	8	363
ideograph	28	9	356

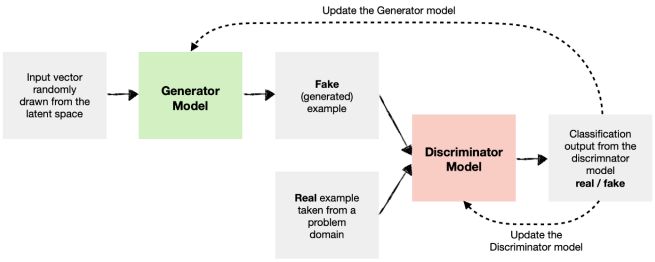


Fig. 3: Architecture of traditional Generative Adversarial Networks (GANs)

the generator when generating a new fake icon, and to the discriminator in order to learn whether it belongs to the specific class.

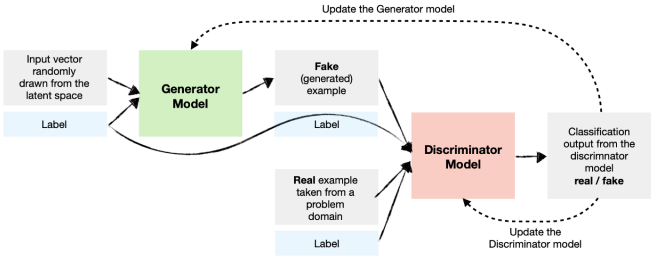


Fig. 4: Architecture of Conditional Generative Adversarial Networks (cGANs)

With all this in mind, we built our model, starting with the generator.

As it is clear from Fig. 5 , the generator takes in the class label of the icon we want to generate and an input vector drawn randomly from the latent space, with a dimension set at 100. It has a total of 1.38M parameters.

In terms of the activation layers, we opted for the leaky RELU with an alpha of 0.2, apart from the output layer with a *tanh* activation.

Now for the discriminator, as it is evident from (Fig. 6), we again chose the leaky RELU function with an alpha of 0.2 for most of the model, apart from the output, which in this case has a *sigmoid* activation, as we are dealing with a binary classification problem.

Since the discriminator is to be trained individually, it must be compiled with a given optimizer, in this case, the Adam optimizer, with a learning rate $LR = 2 \times 10^{-4}$ and a $\beta_1 = 0.5$. It has a total of 0.22M parameters.

Finally, we assemble our cGAN model by taking the generator's output and the input class label and feeding them to the discriminator, as seen in Fig. 7.

We chose to train trained two models with identical architectures, first with our original dataset, the Icons-50, and then with the new Icons-10 subset. Given the smaller dataset, we picked an appropriate batch size to match its length, seen in Table II.

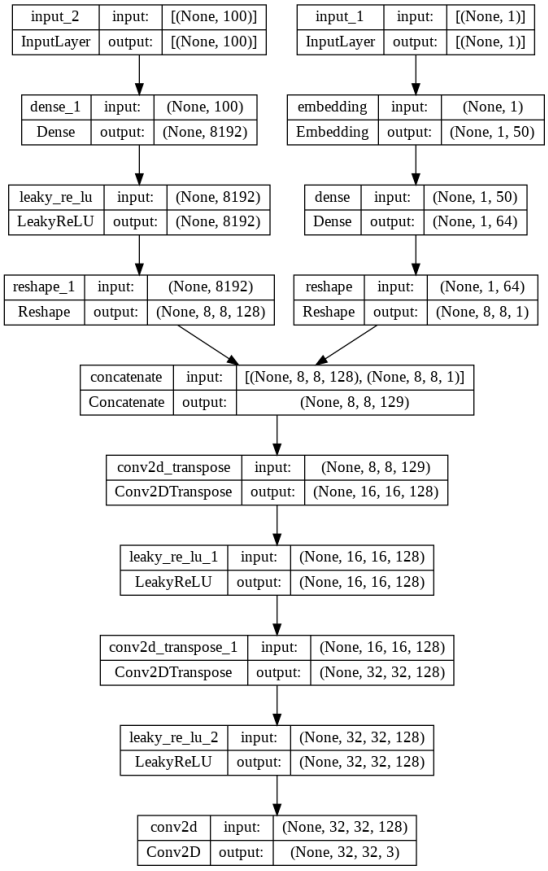


Fig. 5: Structure of the generator model

TABLE II: Model parameters

Dataset	Parameters	Epochs	Batch Size	Latent Dim
Icons-50	1.6M	100	50	100
Icons-10	1.6M	100	30	100

III. Results

After training the model with the Icons-50 dataset for 100 epochs, I was able to achieve a remarkable classification accuracy on the discriminator, seen in Fig. 8 , with a 94% accuracy for real images and a 98% accuracy for generated images.

In terms of loss, while the discriminator finished with a satisfactory minimal loss of 0.329, the same could not be said for the generator, as its loss kept increasing as the training went along, finishing at a value of around 2.532, as can be seen in Fig. 9 .

Afterwards, I trained a new model, with the same structure, on the Icons-10 subset, again for 100 epochs. When analyzing the discriminator's performance, it seemed that both its loss and accuracy suffered.

While the loss only increased slightly, to an acceptable value of 0.643, the accuracy while classifying images as real or fake varied quite drastically during training, as can be seen on Fig. 10 , finishing at, respectively, 83.33% and 86.67%.

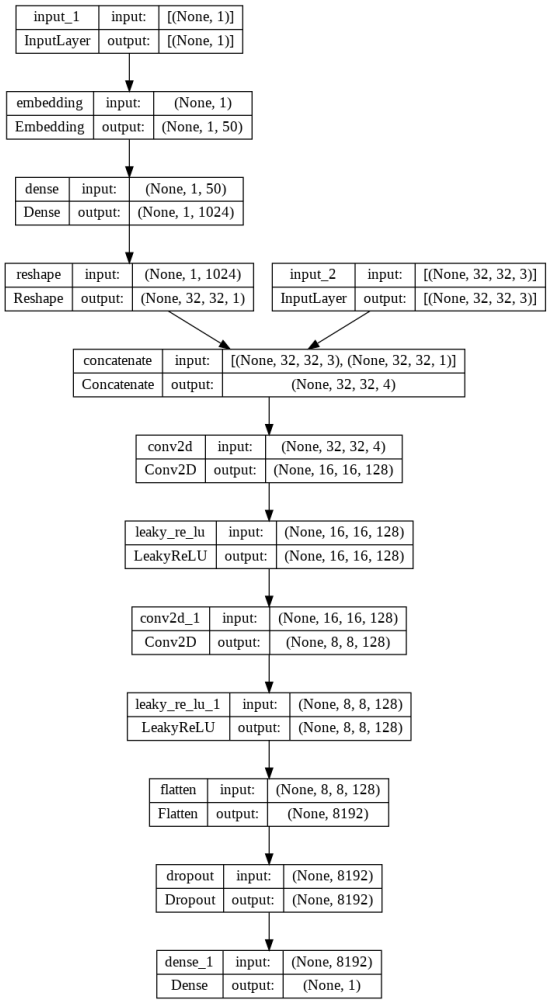


Fig. 6: Structure of the discriminator model

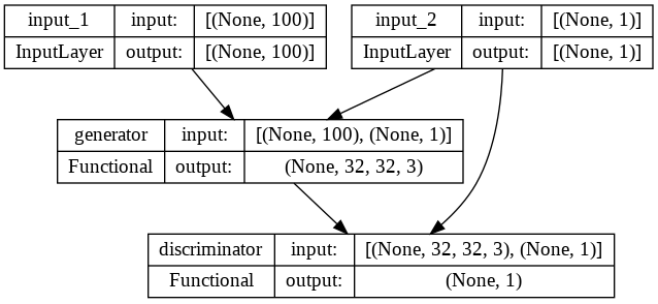


Fig. 7: Structure of the cGAN model

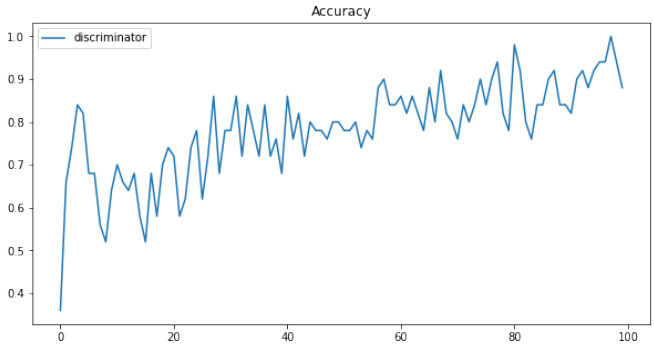


Fig. 8: Discriminator accuracy on the Icons-50 dataset

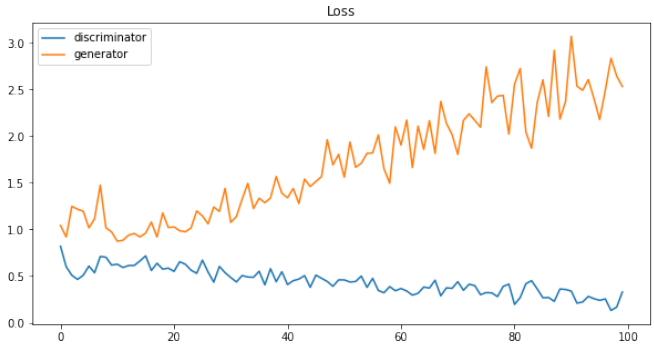


Fig. 9: Model loss on the Icons-50 dataset

In terms of the generator's performance, while the loss still varied quite significantly, the new dataset proved successful, with the model finishing with a much better loss of 1.433, as seen on Fig. 11 .

To better evaluate the performance of the trained generators, I generated a new set of icons belonging to the classes common to both datasets.

When given a simple task, such as generating a "clock" icon, the model trained on the Icons-50 dataset only seemed to perceive the shape of the clock, with none of the defining details being present in the generated images, evident from Fig. 12a.

The second model performed much better on this task,

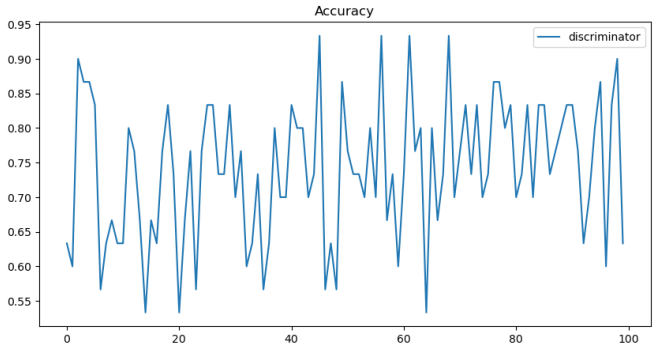


Fig. 10: Discriminator accuracy on the Icons-10 dataset

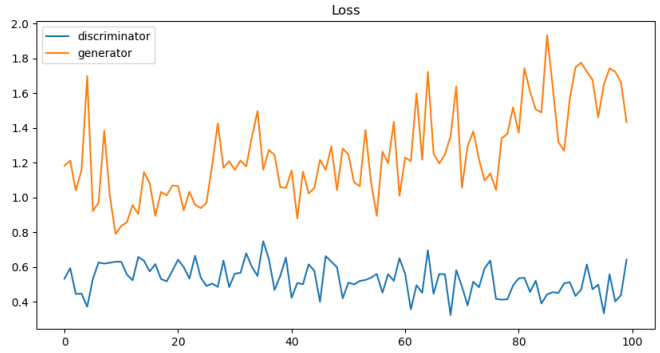


Fig. 11: Model loss on the Icons-10 dataset

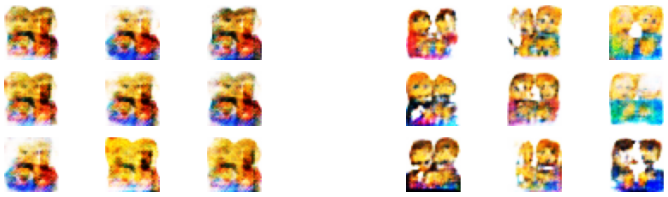
capturing finer details such as the correct color, and the hour and minutes hands, as it is clear from Fig. 12b.

(a) Icons-50 dataset (b) Icons-10 dataset

Fig. 12: Generated icons belonging to class "clock"

When dealing with a complex task, such as generating icons for the "family" class, the original generator was able to create recognizable images, even though they lacked definition and variability, as seen on Fig. 13a.

When trained on the Icons-10 dataset, the model was able to generate a unique and slightly more detailed set of icons, albeit with variable results, as seen on Fig. 13b.



(a) Icons-50 dataset (b) Icons-10 dataset

Fig. 13: Generated icons belonging to class "family"

IV. Discussion

When dealing with imbalanced data, such as the Icons-50 dataset, it is important to consider how this may impact the training of a machine learning (ML) model, especially when the number of classes is large.

When trained with the original unfiltered data, the discriminator was able to achieve a high accuracy of around 96% and a small loss of 0.329. While these results would be optimal in their own, when analyzed alongside the generator's performance, the model seemed to overfit the data, possibly due to the large number of classes, finishing off with a disappointing loss of 2.532.

Afterwards, I decided to filter out the dataset by choosing the 10 most frequent classes, in the hope that when training a new model, I would be able to achieve better results when it came to the icon generation. I called this new dataset, fittingly, the Icons-10 dataset.

I then trained a new model with an identical structure to the first one, but now on the filtered data. As to be expected, while the discriminator suffered a slight dip in performance, with an accuracy of 85% and a loss of 1.433, the generator improved quite significantly, with a lower loss of 1.433.

When analyzing the icons generated by each model, it became apparent that, while the first model was able to learn the overall structure and element composition of the icon, particularly in the case of the "family" class, the results were lackluster and lacked the proper detail to discern what they represented, without the added context, as seen in Fig. 12a,13a.

This suggests that the generator may not have been deep or large enough to learn the necessary features for generating icons for 50 different classes, while the model trained with the Icons-10 dataset was able to learn and generated detailed and striking images, such as Fig. 12b,13b.

V. Conclusions

In this paper, the ability of conditional generative adversarial networks (cGANs) to generate icons with specific characteristics or properties was explored using the Icons-50 dataset. To address the issue of imbalanced data in the original unfiltered dataset, a new dataset was created by filtering the original dataset to include only the 10 most frequent classes (the Icons-10 dataset).

When comparing the performance of the cGANs trained on these two datasets, it was found that the model trained on the Icons-10 dataset was able to generate more detailed and striking icons compared to the model trained on the unfiltered data. This suggests that the model trained on the Icons-10 dataset was able to learn more effectively and generate more accurate and representative icons for the desired classes.

In conclusion, cGANs can be an effective tool for generating icons with specific characteristics or properties, given a suitable dataset. In this case, filtering the original dataset to include only the most frequent classes improved the performance of the cGANs, demonstrating the importance of considering the impact of imbalanced data on model training.

Future work could include testing the performance of cGANs with different hyper-parameters or architectures, or using a different dataset with more balanced class distribution. Additionally, exploring the use of cGANs for generating icons with other specific characteristics or properties, such as color or shape, could be a valuable direction for future research.

References

[1] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *NIPS Deep Learning and Representation Learning Workshop*, 2014. [Online]. Available: https://arxiv.org/abs/1411.1784

[2] D. Hendrycks, “Icons-50 — Kaggle,” 2018. [Online]. Available: https://www.kaggle.com/datasets/danhendrycks/icons50
[3] D. Hendrycks and T. G. Dietterich, “Benchmarking Neural Network Robustness to Common Corruptions and Surface Variations,” *Proceedings of the International Conference on Learning Representations*, 7 2019.
[4] R. Liu, X. Wang, H. Lu, Z. Wu, Q. Fan, S. Li, and X. Jin, “SCCGAN: Style and Characters Inpainting Based on CGAN,” *Mobile Networks and Applications*, vol. 26, no. 1, pp. 3–12, 2 2021. [Online]. Available: https://link.springer.com/article/10.1007/s11036-020-01717-x
[5] M. Loey, G. Manogaran, and N. E. M. Khalifa, “A deep transfer learning model with classical data augmentation and CGAN to detect COVID-19 from chest CT radiography digital images,” *Neural Computing and Applications*, pp. 1–13, 10 2020. [Online]. Available: https://link.springer.com/article/10.1007/s00521-020-05437-x
[6] Y. Ma, X. Cheng, D. Xiang, F. Shi, W. Zhu, and X. Chen, “Speckle noise reduction in optical coherence tomography images based on edge-sensitive cGAN,” *Biomedical Optics Express, Vol. 9, Issue 11, pp. 5129-5146*, vol. 9, no. 11, pp. 5129–5146, 11 2018. [Online]. Available: https://opg.optica.org/boe/fulltext.cfm?uri=boe-9-11-5129
[7] J. Brownlee, “How to Develop a Conditional GAN (cGAN) From Scratch - MachineLearningMastery.com,” 7 2019. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/
[8] ——, “How to Develop a GAN to Generate CIFAR10 Small Color Photographs - MachineLearningMastery.com,” 7 2019. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/