Generative Adversarial Networks: A Review and Application

Drew Lawless & Chris Embs
Emory University, CS 334
December 11, 2024

Abstract

Generative Adversarial Networks (GAN) are a relatively recent addition to the numerous types of neural networks that are being modified for specific outputs. They consist of two separate net models, a generator (G) and a discriminator (D), which are trained and tested simultaneously with competing motivations. Based upon input data of any variety, G is designed to produce synthetic versions that mimic their qualities, while D attempts to correctly classify G's output as original or not. As D becomes more adept at deciphering between these two datasets, G is more inclined to fabricate accurate representations of its input in a zero-sum game against one another. This will ultimately result in a highly robust data generator and class discriminator. Their primary use continues to focus on image and video recognition or segmentation, but there are a multitude of newer versions with more complex layouts to both mitigate prior issues and expand on their capabilities. A seemingly simple yet advanced example of this is the deep convolutional GAN (DCGAN), which introduces convolutional layers and spatial up-sampling to prevent mode collapse in G's output. In this paper, we provided a basic overview of the GAN and DCGAN layout in conjunction with relevant equations. Alongside this, we made our DCGAN for analyzing and upscaling an online, public dataset of color images. Our final model exhibited relatively stable degrees of competition between the generator and discriminator. The somewhat fluctuating MSE score suggests higher variability in pixel-wise quality over epochs, and consistently low SSIM scores imply little increase in structural similarity to the sample images. Overall, the general behavior of the model showed competitive improvement, but it could be expanded in the future for perceptual accuracy.

1 Introduction

Generative Adversarial Networks (GANs), first developed and published by Ian Goodfellow in a 2014 seminal paper, were designed to greatly improve upon the lacking development of deep generative models at the time [2]. While discriminatory models are relatively effective at maintaining weight-gradients through backpropagation and dropout, generative models have a much harder time reconstructing input from latent space to data space. This includes gradient explosion from ReLU functions and the introduction of noise or artifacts into the output. The focus of GANs in comparison to prior generative models is the adversarial component of its design. A generator model (G) is trained from a random input noise vector (z) derived from the input data distribution (x). G learns to produce fake data based on characteristic features of z and has its own data distribution (p_g). A discriminator model (D) then receives information on p_g and is trained in a competitive manner to precisely define these inputs as either real or fake (0 or 1) in a binary classification system. This is done in coordination with x to provide D a basis for real images, as well. The overall function of this design is the formation of a two-player minimax game that focuses on G's goal of fooling D and D's goal of predicting G's output. As a result, both a proficient generator and a robust discriminator can be developed simultaneously [14]. In the vanilla GAN from the original paper, its advantages included no need for Markov chains for handling maximum likelihood estimations (MLE) and intractability, backpropagation controlling gradients effectively, numerous activation functions available for given perceptrons, and absence of inference for latent variable computation. The main disadvantage is the necessity for synchronization of G's training with D's, too, in order to avoid mode collapse and minimized learning in creating fake data. It also lacks a true probability distribution of its outputs. As a potential improvement to the Vanilla-GAN model, the Deep Convolutional GAN (DCGAN) was developed shortly after in a 2016 paper [9]. Its critical differences from the origi-

nal model are the replacement of fully-connected layers with more convolutional layers, strided convolutions instead of pooling functions, and strictly applying batch normalization. The overall focus of these convolutional-heavy GANs is to increase resolution from the original data samples to produce higher-quality fakes via G. Thus, they were a great inspiration for our project, which is to design a DCGAN trained on the CIFAR-10 dataset. We've detailed both the Vanilla-GAN and DCGAN outlines along with relevant computation for understanding their comparative function. This was followed by our own development of a super resolution DCGAN designed to upscale and analyze the public CIFAR-10 dataset, which was used in the original Goodfellow et al. paper as well. The report on our findings include generator and discriminator individual loss via cross-entropy, mean squared error (MSE) for pixel-wise loss from the sample images, and how they changed throughout the training of the program.

2 Vanilla-GAN

2.1 Outline

The generator G is best defined as the differentiable function $G(z;\theta_g)$, which consists of the pure random noise vector z and the parameters of the multilayer network θ_g for mapping the latent space data distribution of the input x to data space. Its output x_g is expected to mimic the real data x_r from the real data distribution $p_r(x)$ using its fake data distribution $p_g(x)$ following successful training. The random noise vector is sampled from the prior Gaussian distribution $p(z)$ The discriminator D is best defined as the binary function $D(x;\theta_d)$, which consists of the real data distribution x_r and the parameters of the multilayer network θ_d for mapping. Its output is defined as y_1, which is a binary classification on whether the input is from x_g or from x_r. The two equations below represent the outputs of G and D respectively:

$$x_g = G(z;\theta_g)$$

$$y_1 = D(x;\theta_d)$$

Ultimately, G can be perceived as a generator supervised by D when given reward or punishment in the form of D's cross-entropy loss. D can also be viewed as a discriminator supervised by G, as it takes G's cross-entropy loss from inaccurate fakes,

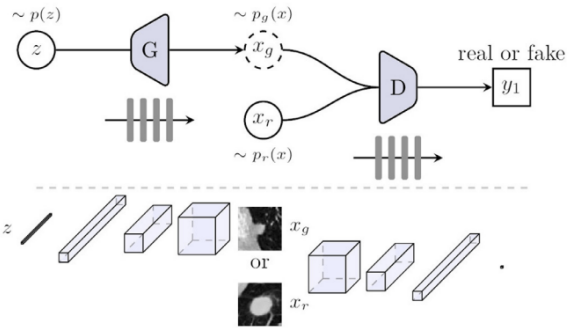

Figure 1: "Schematic view of the vanilla GAN for synthesis of lung nodules on CT images. The top of the figure shows the network configuration. The part below shows the input, output, and the internal feature representations of the generator G and discriminator D. G transforms a sample z from $p(z)$ into a generated nodule x_g. D is a binary classifier that differentiates the generated and real images of lung nodules formed by x_g and x_r, respectively." - (As stated by Yi et al., 2019)

too. Information on the gradients is backpropagated from D to G as they both correct their hidden layer feature weights for more optimal fooling or discerning[14]. A visual layout of these models, their inputs and outputs, and the convolutional and fully-connected layers is shown in figure 2.

The cross-entropy loss is a critical metric for evaluating the performance of the discriminator by comparing the predicted label and actual label for the input images from the original data sample or the generator's data output. The generator wants to minimize its own cross-entropy loss in the exact opposite way by increase the loss of the discriminator[14]. The two relevant equations (adversarial loss) for training objectives related these models are listed below as well as the overall value function:

$$\mathrm{L}^{D}_{GAN} = \max_D \Big(E_{x_r \sim p_r(x)}[\log D(x_r)] + E_{x_g \sim p_g(x)}[\log(1 - D(x_g))] \Big)$$

$$L^{G}_{GAN} = \min_G E_{x_g \sim p_g(x)}[\log(1 - D(x_g))]$$

2.2 Evaluation Metrics

Evaluation metrics, while critical to assessing the performance of a GAN, remain complex to implement due to the wide range of structures and applications of the model [9]. As GANs are traditionally utilized for the generation of visual data, such as images and videos, a unique challenge arises where it becomes difficult to apply quantitative markers

to evaluate the model. As a result, GANs are often evaluated qualitatively, by visual inspection. Visual inspection, however, is not just time consuming, but is also prone to subjectivity. While to the inspector, an image might look satisfactory and real, it may not actually be suitable for that specific GAN's output goals. Visual inspection also often overlooks mode collapse, where the images created by the generator look real but are all far too similar, as the model fails to cover the full distribution of data. Despite these challenges, common machine learning evaluation metrics, such as precision, recall, and F1 score, can still provide useful information about the model's performance. With these metrics, insights can be made into the trade-off between quality and diversity, allowing for the quantification of over-fitting or mode collapse[10]. The most widely adopted quantitative evaluation metric for the GAN is the Inception Score (IS). The IS evaluates both the quality and the diversity of the produced set of images based on a pre-trained classifier, the Inception Model. Samples that are ideal visually and are therefore easily classifiable, produce a highly confident conditional distribution $P(Y|X)$, where Y represents class labels and X the generated image as seen in EQ.1 below. Samples that have ideal diversity create a marginal distribution $P(Y)$ that has high entropy, which is then compared to $P(Y|X)$ to see how they deviate. In turn, a high IS score indicates that a sample can be confidently classified, but is also spread across many classes. The Fréchet Inception Distance (FID), compares the distribution of the generated images to that of the real images using the Fréchet distance. The goal of this approach is to assess how similar the two distributions are by comparing their means (μ_r, Σ_r) and covariances (μ_g, Σ_g) as seen in EQ.2 below. By evaluating the distributions, both image quality and diversity can be measured with better performance being signified by a lower score. The Multi Scale Structural Similarity Index (MS-SSIM) measures diversity by assessing perceptual similarity between two images. The index in which this metric assesses the images can be split into Luminance $l(x, y)$, Contrast $c(x, y)$, and Structure $s(x, y)$ as seen in EQ.3 below. Similarly to the Discriminator, the MS-SSIM outputs a score between 0 and 1, however this score corresponds to how similar the image is to another. A high score between generated images therefore indicates a lack of diversity between those produced by the Generator, but a high score between a generated image and a real one signifies the quality of the image[1].

$$IS = \exp\left(E_x\left[KL(p(y|x)\|p(y))\right]\right)$$

$$\text{FID} = \|\mu_r - \mu_g\|^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

$$MS - SSIM(x, y) = \prod_{j=1}^{M} (SSIM_j(x, y))^{\alpha_j}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

2.3 Challenges

A challenge that commonly occurs when first creating a GAN is Mode Collapse. This occurs when the generator outputs a narrow selection of data despite receiving a random input of noise z. For image generation this may mean that despite being trained on a set of varying image inputs, the Generator learns to fool the Discriminator only by finding a small set of outputs that are able to consistently pass. This in turn will lead the model to output images lacking diversity regardless of the set from which they are trained. While this often occurs when the Generator over-optimizes for the discriminator, it can also be caused if the Discriminator becomes too strong too early. Here, the Discriminator's extreme confidence will consistently assign the generated data low probabilities, leaving the Generator to focus only on a narrow range of outputs that can produce meaningful progress. This idea is called a Vanishing Gradient where D(G(z)) = 0, meaning the discriminator can perfectly identify the given samples, leads the generator's loss function to approach 0, resulting in near-zero gradients. Somewhat similar to Mode Collapse is the issue Non-Convergence in GANs which occurs as a result of the adversarial dynamic seen between the two numeral networks. GAN training is a minimax optimization problem, as can be seen in EQ.1 below, where the Generator and Discriminator have opposing objectives. This, however, means that the improvement of one model can debilitate the other creating an imbalance between the two networks. If the discriminator becomes too strong, it can all too

confidently identify the Generator's synthetic data leading to Mode Collapse as outlined above. If the Generator becomes too strong, however, its outputs will consistently fool the Discriminator, weakening its gradients and making it unable to correctly distinguish real from fake data. This phenomenon in which one adversarial network dominates another is called Instability. It is signified by Diverging Losses, where one model's loss grows uncontrollably as a result of the other's. It is also possible to see Oscillating Losses in which both the Generator and Discriminator switch dominance periodically, however, no progress towards an equilibrium is made[2]. The minimax value equation for the objective of the entire GAN is shown below:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

3 DCGAN

3.1 Key Changes

The DCGAN is an example of a CNN-based GAN that utilizes CNN architecture in both the generator and the discriminator. This involves a series of transposed convolutional layers for upsampling the image resolution until it's optimal level is reached. No max-pooling or average-pooling is used in these layers to prevent any gradient destabilization. Thus, strided convolutions which skip over some pixels in the input are used to both decrease computational burden as well as capture more of the entire image. Convergence speed was found to decrease greatly using these pooling functions for features inputs. The random noise vector is also passed through a fully-connected layer to extract important features determined previously by the generator, and this goes into the deep convolutional layers designed to convert it into a 4D tensor for image generation and reshaping. The discriminator performs the opposite in that it flattens the multi-dimensional input image for usage in the sigmoid activation function, thus outputting a binary classification as real or fake. Lastly, batch normalization for zero mean and unit variance improves the consistency of each layers' distribution of total inputs to the next as well as gradient flow. Mode collapse to a singular generator output was similarly mitigated by ensuring a regular degree of weight updating and conditioned layer activation. It was found that the best practice of this was not applying batch normalization to each layer but only to a select grouping of them to prevent mode instability and oscillation of generator performance. Refer to figure 2 for a basic outline of the generator used in the original DCGAN paper[3].

3.2 Super Resolution

One of the major additions we made to our DCGAN model that wasn't a component from the original paper was the concept of super resolution. This involves conversion of low resolution (LR) inputs to high resolution (HR) fake images using the generator's recovery of high-frequency features, textures, and details in its convolutional hidden layers[6]. The DCGAN is a great medium through which this can be used due to it focus on whole-image analysis and reducing computational complexity in the deconvolution process. Greater depth to it's hidden layers allows for more opportunity at isolating the most appealing and definable characteristics of the random noise vector. Even though the total number of pixels may not be increasing, the exact visual information portrayed by each pixel is aligned more so with the original images and of greater quality. Adversarial loss in the form of previously mentioned cross-entropy is used by both the generator and the discriminator. This involves the generator determining what aspects of resolution increase allow for more convincing images, while the discriminator determines what quality is more in line with the original sample images. Perceptual loss is used specifically for single-image super resolution, just as in our DCGAN model. Compared to mean-squared error (MSE) and other pixel-wise evaluation metrics, this calculates the difference between features maps obtained in hidden convolutional layers. The main purpose of this is to ensure the resolution increase isn't obtained over the maintenance of basic identifiable image details. The MSE and perceptual loss equations are shown below[7]:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$P_{loss} = \frac{1}{N}\sum_{i=1}^{N}|f(x_i) - y_i|$$

4 Our Model

4.1 Fundamentals

Our DCGAN utilizes the python-based TensorFlow software platform for machine learning, which is

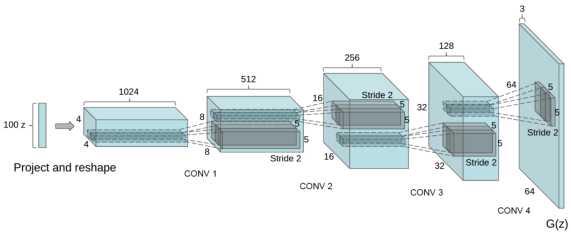

Figure 2: "DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64 × 64 pixel image. Notably, no fully connected or pooling layers are used." - (As stated by Radford et al., 2016)

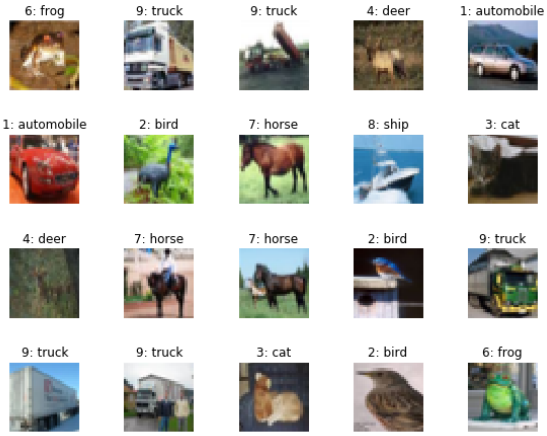

Figure 3: An example set of images from the CIFAR-10 dataset. Each image has an integer classifier to represent the object or animal present in the image.

in coordination with the NumPy library for multidimensional arrays and Matplotlib for statitstics and visuals related to the outputs [11]. Our initial code implemented a traditional DCGAN without much, if any, distinct changes from the original paper. Following a set of trial-and-error testing of the code as well as debugging, we decided to make a crucial change to the code. This involved adding super-resolution to upscale the original low-quality images by enhancing pixel sharpness and contrast instead of total image size. Due to the increased convolutional depth, more patterns and textures can be extracted from the input batches in the form of features for improving perceptual appeal and definition. This adds to both the human aspect and the discriminator's challenge of comparing from real to fake data[6]. The TensorFlow software and its library was also utilized for this implementation via pre-processing of images in lower resolution for conversion to higher resolution in the generator[12]. The training of our DCGAN model uses a series of 100 epochs with 128 images per batch and 391 steps per epoch. Each batch is a subset of images used to train both the generator and discriminator while updating the weights throughout the process. An epoch is counted as each time the models are allowed to see the entire dataset, which happens after a given number of batches are used. This is done with the purpose of introducing more variety to the training sets, optimizing gradients more consistently, and less usage of memory. In terms of the hidden convolutional layers of the generator, Leaky ReLU activation functions were used while tanh activation functions were used in the output layers. Only Leaky ReLU was used in the discriminator to ensure non-linearity. The Conv2DTranspose function was used in the generator to gradually increase spatial dimensions via upsampling and account for image quality simultaneously. The Conv2D function was used in the discriminator for extracting features and downsampling the spatial dimensions of the input images for fully-connected layer analysis of its binary classification. The general flow of the code consists of pre-processing the images, preparing the generator and discriminator models, training them on 100 epochs, and finally testing them for visual correctness and checking evaluation metrics of each net. Testing occurs following each epoch, but final metrics such as cross-entropy loss and MSE from the scikit-learn library are included for general model performance[5]. We created our code using Google Colab Pro and it's T4 GPU for faster image processing.

4.2 Dataset

Our DCGAN model was trained using *CIFAR-10*, a subset of the original *80 Million Tiny Image* dataset from a collaboration between *Massachusetts Institute for Technology* and *New York University* in 2008. It was modified by the *Canadian Institute for Advanced Research* (CIFAR) in conjunction with the *University of Toronto* in 2009[4]. It consists of 60000 labeled, 32x32 (pixel) color images. There are a total of 10 classes for any of the images, which include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. All classes are mutually exclusive with no overlap for potentially similar observational classification.

Five training batches and one test batch, each with 10000 images, were used in the creation of our model. Each training batch has 5000 randomly-selected images, while each test batch contains 1000 randomly-selected images, all from any class. Refer to figure 3 for example images from the original dataset.

4.3 Results

The results of our model were fairly positive since the evaluation metric that we thought was the most important was visual perception. The GAN almost always successfully created images that to human evaluators were recognizable and distinctly classifiable into one of the pre-defined CIFAR-10's image categories. After tweaking the Generator, Discriminator, and adding additional evaluation metrics to our code, the model did begin to run much more slowly. This is in part however, because of our decision to include a batch size of 391 and 100 epochs to allow time for our Generator and Discriminator losses to push towards their equilibrium, despite seeing reasonable results after just the first epoch. However, most of the additional runtime came as a result of our decision to calculate the model's MSE and SSIM as they are both computationally expensive requiring excess calculation to determine the similarity of each generated image from the real ones. When evaluating our graph of the Generator and Discriminator losses, as can be seen below, we noticed both losses to be flat-lined early on. For our model, this means the generator was not improving significantly over the course of the 100 epochs, and for the most part did not get any better at producing more realistic data in the eyes of the discriminator. Since the synthetic data the generator produced early on were of such great quality and there was so little improvement over time, we believe this issue to be due to Mode Collapse. This would also explain why we saw a flat-line in the discriminator's loss. Since the generator was able to find a small subset of data that could consistently pass without needing variation, the discriminator became no better at identifying that information over the course of the training. This result is further supported by the MSE and SSIM graphs, as the MSE improved only over the first few iterations, and the SSIM stayed the same throughout the duration of the training. This meant that the generated images quickly learned to pixel match with the real images early on, but did not find it necessary to improve in this area or in the structure and quality of the image in order to pass the discriminator consistently. As a result, with each subsequent epoch, there would be changes mostly to the lighting of the image, as the MSE fluctuated in spikes, but the structure and quality of the image remained mostly the same. Overall, despite not seeing little improvement among our losses and in our MSE or SSIM, the model was still able to produce recognizable images through its early training. Refer to figure 4 for detailed changes in the three previously mentioned evaluation metrics.

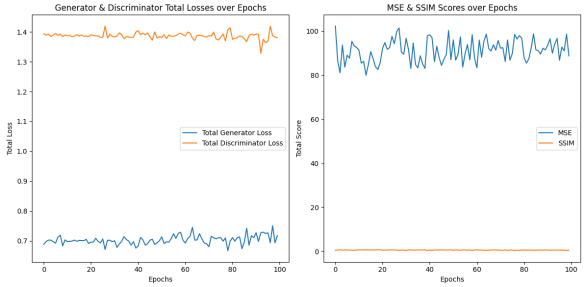

Figure 4: Three evaluation metrics used to analyze model performance via two graphs. Left graph details generator and discriminator adversarial loss throughout training. Right graph shows MSE and SSIM scores throughout training.

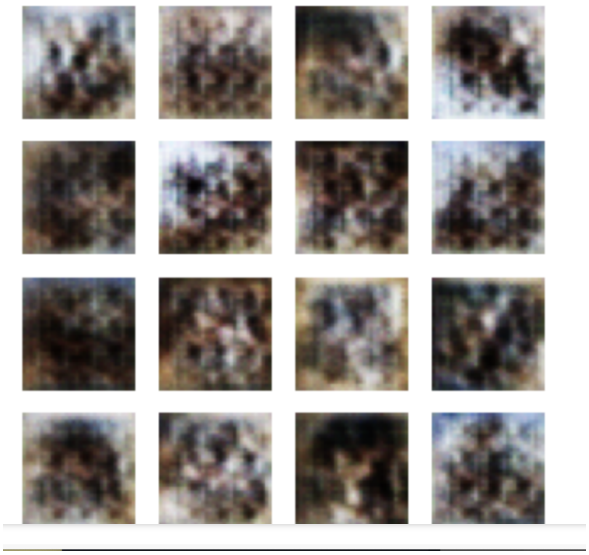

Figure 5: An example of the generator's output following 100 epochs of training using a batch size of 128. Despite increasing the number of epochs and the batch size from our previous iteration of the GAN, the images were still unrecognizable.

5 Future Directions

Overall, our implementation of a super-resolution DCGAN saw moderate success in terms of eval-

Figure 6: An example generator output following 100 epochs of training using a batch size of 128 images and 391 steps per epoch. This was done using super-resolution and increased total epochs from 50 in the previous code design. Similarity to deer, bird, automobile, and truck classes is evident from these four images

uation metrics used for measuring loss as well as visual quality over the training and testing epochs. Luckily, the model did not have any evidence of vanish gradients, mode collapse, convergence issues, or instability. This was most likely a result of the relative simplicity of our model, the low-resolution dataset used, and the small number of epochs compared to larger studies. Limits to our device's GPU and RAM capabilities via Google Colab required us to take some steps back in what we could include during this singular project of isolated scope. Our usage of just MSE and SSIM along with cross-entropy loss for the generator and discriminator's individual performances were useful; however, additional visual evaluation metrics such as FID, IS, and PNSR would've given greater detail on what was be gained from each epoch in terms of image quality and similarity to the sample dataset. Unfortunately, these require immensely complex algorithms and add computational burden not possible in this basic code implementation or course in general. Binary cross-entropy loss could have been replaced with metrics such as Wasserstein loss for continuous gradient feedback. Including perceptual loss with pre-trained models such as the Visual Geometry Group (VGG) CNN architecture could have provided a greater metric for comparing our model's resolution and

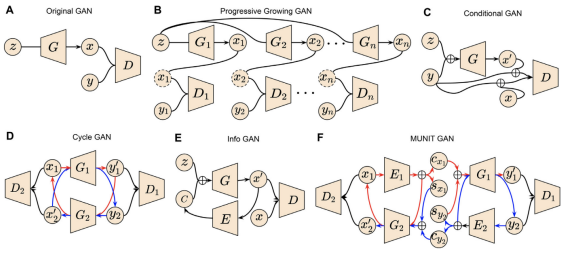

Figure 7: "Semantics of the original GAN and its extensions." - (As stated by Wang et al., 2023)

feature extraction with a much more robust model trained on many types of images with varying content [8]. The DCGAN, while a great variation of the Vanilla-GAN for upscaling image quality given CIFAR-10's low pixel count, may be worth replacing or combining with other ones. The Conditional-GAN (CGAN) includes an extra variable matrix for focused generation features and parameters, thus giving it a more targeted approach to image recognition and extraction. The Progressively Growing-GAN (PGGAN) give both the generator and discriminator space to deepen their hidden convolutional layers for gradually increasing image resolution as needed and more computational options. This could've been especially helpful for our project when considering how generated image quality began to taper off and/or oscillate after roughly 50 epochs[13]. Refer to figure 7 for example outlines of other GAN variations.

BibTEX Files

References

[1] Ali Borji. 2022. Pros and cons of gan evaluation measures: New developments. *Computer Vision and Image Understanding*, 215:103329.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680. Curran Associates, Inc.

[3] John Jenkins and Kaushik Roy. 2024. Exploring deep convolutional generative adversarial networks (dcgan) in biometric systems: a survey study. *Discover Artificial Intelligence*, 4(1):1–12.

[4] Alex Krizhevsky and Geoffrey Hinton. 2009. Cifar-10 dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`. Accessed: 2024-12-09.

[5] Scikit learn developers. 2024. *Scikit-learn: Machine Learning in Python - Model Evaluation*. Accessed: 2024-12-10.

[6] Dawa Chyophel Lepcha, Bhawna Goyal, Ayush Dogra, and Vishal Goyal. 2023. Image super-resolution: A comprehensive review, recent trends, challenges and applications. *Information Fusion*, 91:230–260.

[7] Y. Li, J. Hu, H. Sari, S. Xue, R. Ma, S. Kandarpa, D. Visvikis, A. Rominger, H. Liu, and K. Shi. 2023. A deep neural network for parametric image reconstruction on a large axial field-of-view pet. *European Journal of Nuclear Medicine and Molecular Imaging*, 50(3):701–714.

[8] Kha-Luan Pham, Khanh-Mai Dang, Loi-Phat Tang, and Thanh-Nhan Nguyen. 2020. Gan generated portraits detection using modified vgg-16 and efficientnet. In *Proceedings of the 2020 7th NAFOSTED Conference on Information and Computer Science (NICS)*.

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

[10] Divya Saxena and Jiannong Cao. 2021. Generative adversarial networks (gans): Challenges, solutions, and future directions. *University Research Facility in Big Data Analytics (UBDA), The Hong Kong Polytechnic University*.

[11] TensorFlow. 2024. Dcgan tutorial. Accessed: 2024-12-09.

[12] TensorFlow Hub Team. 2024. Image enhancing with tensorflow hub. `https://www.tensorflow.org/hub/tutorials/image_enhancing`. Accessed: 2024-12-10.

[13] Rongguang Wang, Vishnu Bashyam, Zhijian Yang, Fanyang Yu, Vasiliki Tassopoulou, Sai Spandana Chintapalli, Ioanna Skampardoni, Lasya P. Sreepada, Dushyant Sahoo, Konstantina Nikita, Ahmed Abdulkadir, Junhao Wen, and Christos Davatzikos. 2023. Applications of generative adversarial networks in neuroimaging and clinical neuroscience. *NeuroImage*, 269:119898.

[14] Xin Yi, Ekta Walia, and Paul Babyn. 2019. Generative adversarial network in medical imaging: A review. *Medical Image Analysis*, 58:101552.

Contributions

Drew was responsible for writing the abstract, introduction, Vanilla-GAN outline, DCGAN key changes, super-resolution, our model fundamentals, dataset and future directions sections. Chris was responsible for writing the results, evaluation metrics, challenges, and acknowledgments. We both contributed equally to designing the code, analyzing its performance, literature review, bibliography, and referencing.

Acknowledgments

We would like to thank our professor, Dr. Fei Liu, for her excellent teaching of *CS 334: Machine Learning* at Emory University as well as her guidance in creating this project throughout the fall semester. We would also like to thank the many peers who participated in the course for their assistance during and after our project presentation. We greatly appreciate their willingness to listen and learn alongside us.