International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 06 | June - 2023 SJIF Rating: 8.176 ISSN: 2582-3930

Fake Image Generation Using Deep Convolutional Generative Adversarial Networks (DCGANs)

Mohit Joshi*

* Department of School of Computing, Graphic Era Hill University

Abstract- As computing power has developed over the years, various generative models have also been created in the machine learning field. One particularly well-liked generative model, known as Generative Adversarial Networks, has only recently been introduced and studied among all these generative models. It is predicated on the idea of two competitors who are always attempting to exceed one another. The goal of this review is to thoroughly analyse the GAN-related literature and present a concise version of the literature that has been studied on GAN, including the idea behind it, its primary function, proposed base model modifications, and current trends in this field of study. This paper is intended to help in providing a comprehensive understanding of GAN.

I. INTRODUCTION

Generative adversarial networks are a strong class of unsupervised learning neural networks. GANs was developed by Ian J. Goodfellow in 2014(Goodfellow et al., 2014). GANs are primarily a framework which has two competing neural network models that compete and can analyze, collect and capture changes in dataset.

Generative Adversarial Networks are used to develop and train generative models with two sub-models as a supervised learning task. The sub-models of GANs are generator and discriminator these 2 are main network of GANs. where the generator model is trained to generate a new sample from the provided input and the discriminator model that tries to identify which is real and which is fake from the provided input(Brownlee, 2019).

The GAN working based on three principles, firstly to make the generative model learn, and the data can be generated employing some probabilistic representation. Secondly, the training of a model is done can be done in any conflicting situation. Lastly by using the deep learning neural networks and using the artificial intelligence algorithms for training the complete system (Liu & Tuzel, 2016). The main idea behind GANs can be interpreted as a game between two players- the generator and the discriminator. The generator tries to generate samples that follow the same underlying distribution as the train data. The discriminator tries to distinguish between the samples generated by the generator (fake data), and the actual data from the train set. The goal of the generator is to fool the discriminator by closely approximating the underlying distribution in order to generate samples that are indistinguishable from the actual data. On the other hand, the goal of the discriminator is to identify the fake data from the real data. The discriminator simply has the task of a binary classification problem, where it determines if the data is real or fake.

The step-by-step functionality of GAN has been explained as follows:

- The users have produced using a generator by the discriminative network from the true data distribution.
- The system has trained so that the liability rate of the network can be increased, and the discriminator network can be fooled by producing such candidates that are not synthesized i.e. still part of data distribution.
- A dataset acts as initial training data for the discriminator.
- For training samples datasets are presented till accuracy is achieved.
- The generator is trained to produce candidates when the discriminator is fooled when it is fed random input it processes them.
- Lastly, backpropagation has been applied to generators as well as discriminator where the former produces better images, and the latter is skilled at fading artificial images.
- A deconvolutional neural network is a generative network and CNN acts as a discriminator.
- Sometimes GANs deal with mode collapse when the network fails to generalize in case missing entire modes from input data.
- Many solutions for one problem are proposed by the researchers.

II. LITERATURE REVIEW

Overview of GAN Architecture: A Generative Adversarial Network (GANs) has two key components—the generator and the discriminator, both of which are deep neural networks.

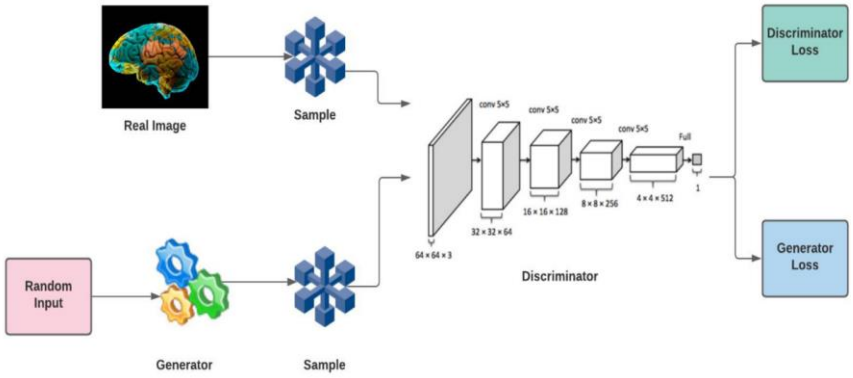

Fig 1: Block diagram of the Generative Adversarial Network (GAN).

Discriminator:

A GAN discriminator is a classifier. It distinguishes between real data and data generated by a GAN generator. It can use any network architecture suitable for the type of data that needs to be generated by the GAN.

The discriminator trains on two types of data. The first is real-world data, such as real pictures of animals. These are treated by the discriminator as ground truth labels of what a "real" artifact looks like. The second is artifacts created by the generator. These are used as ground truth labels of what a "fake" artifact looks like.

The discriminator training process works as follows:

1. The discriminator classifies real and fake data received from the generator.
2. The discriminator computes a loss function that penalizes it for misclassifying a real artifact as fake or a fake artifact as real.
3. The discriminator updates its weights by backpropagating to minimize the discriminator loss—without being aware of the generator loss function.

Generator:

The generator part of the GAN learns to incorporate the discriminator's feedback to generate fake data. Its goal is to cause the discriminator to classify its fake output as real.

An important question is the initial data used by the generator. All neural networks need to start from some input. In its basic form, a GAN takes random noise as input (a common choice is a normal distribution). The generator then transforms this noise into artifacts in the format required by the user.
By introducing noise, the GAN can generate many variations on the data, by sampling at different locations in the target distribution. More advanced GAN architectures use specific inputs to help the network train faster and more effectively.

Generator training works as follows:

1. The generator accepts its initial input, typically random noise, and transforms it into meaningful output.
2. The discriminator provides a classification for the output, classifying it either as real or fake.
3. The discriminator calculates its loss function.
4. Backpropagation is performed through both discriminator and generator networks to obtain gradients.
5. These gradients are used to adjust only the weights of the generator, so that the generated content will be more likely to fool the discriminator next time.

DCGAN :

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. The discriminator is made up of strided convolution layers, batchnorm layers, and LeakyReLU activations. The input is a 3x64x64 input image and the output is a scalar probability that the input is from the real data distribution. The generator is comprised of convolutional-transpose layers, batchnorm layers, and ReLU

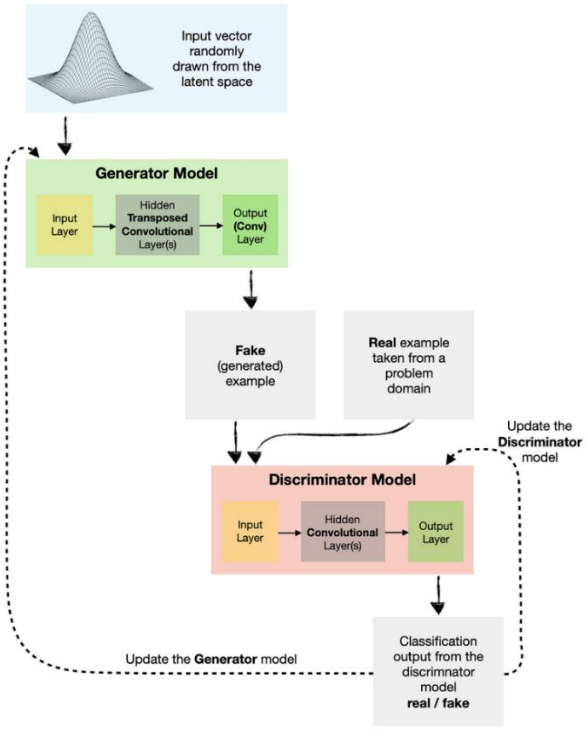

Fig 2 : Block Diagram of the Deep Convolutional GAN activations.

The input is a latent vector, z, that is drawn from a standard normal distribution and the output is a 3x64x64 RGB image. The strided conv-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image.

Overview of DCGAN Architecture:

The generator of the DCGAN architecture takes 100 uniform generated values using normal distribution as an input. First, it changes the dimension to 4x4x1024 and performed a fractionally stridden convolution 4 times with a stride of 1/2 (this means every time when applied, it doubles the image dimension while reducing the number of output channels). The generated output has dimensions of (64, 64, 3). There are some architectural changes proposed in the generator such as the removal of all fully connected layers, and the use of Batch Normalization which helps in stabilizing training. In this paper, the authors use ReLU activation function in all layers of the generator, except for the output layers. We will be implementing generator with similar guidelines but not completely the same architecture.

The role of the discriminator here is to determine that the image comes from either a real dataset or a generator. The discriminator can be simply designed like a convolution neural network that performs an image classification task. However, the authors of this paper suggested some changes in the discriminator architecture. Instead of fully connected layers, they used only strided-convolutions with LeakyReLU as an activation function, the input of the generator is a single image from the dataset or generated image and the output is a score that determines whether the image is real or generated.

III. METHOD PROPOSED

Training of DCGAN:

Since GANs consist of two networks trained separately, the training algorithm must solve a complex problem. A GAN needs to coordinate two training components: a generator and a discriminator. GAN convergence can be difficult to identify.

Generators and discriminators have different training procedures, so it is not obvious how GAN training works as a unit.

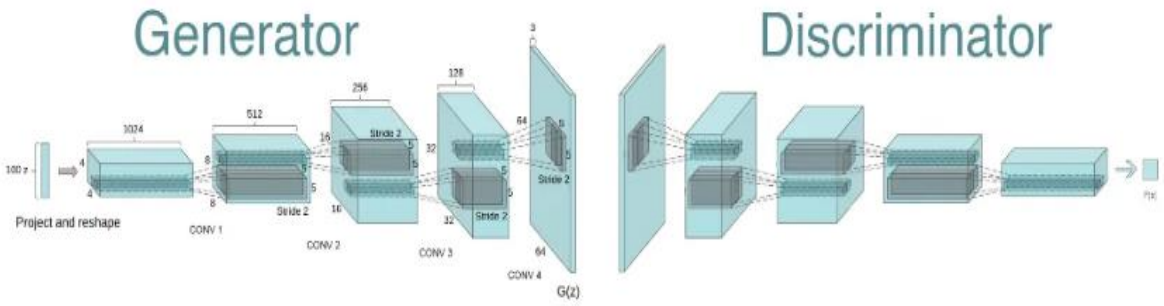

Fig 3 : DCGAN Architecture.

1. A discriminator trains for a set period.
2. A generator trains for a set period.

The model repeats the first two steps to continue training the generator and discriminator networks against each other.

The generator must remain constant throughout the discriminator's training phase. The discriminator training process involves distinguishing between real and fake data, so it must also learn to identify the generator's defects. Fully trained generators present a different challenge from untrained generators, which produce random outputs.

Likewise, the discriminator must remain constant throughout the generator's training phase to allow the generator to tackle a consistent target. With this iterative feedback loop, the GAN can solve complex problems. This process should start with simple classification problems and gradually introduce harder generation problems.

Suppose the initial training phase, using a randomly generated output, fails to train the classifier to distinguish between generated and real data. In that case, it is not possible to progress with the GAN training. The generator improves during training and impacts the discriminator's performance because it becomes harder to distinguish between genuine and fake inputs. If the generator is completely successful, the discriminator is 50% accurate. The discriminator makes a coin-toss prediction.

This process can lead to convergence problems for the overall GAN. In other words, the discriminator's feedback loses meaning over time. If the GAN continues to train beyond the point where the discriminator can only provide random feedback, the generator starts training on meaningless feedback, degrading the generator's quality in turn.

Convergence is usually a transient, short-lived state for GANs rather than a stable state.

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

a) Get the Data:

We'll be using the CelebFaces Attributes Dataset *(CelebA)* to train our adversarial networks.

CelebFaces Attributes Dataset(CelebA) is a large-scale face attributes dataset with more than **200K** celebrity images, each with **40** attribute annotations. The images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities, and rich annotations, including

- **10,177** number of **identities**,
- **202,599** number of **face images**, and
- **5 landmark locations**, **40 binary attributes** annotations per image.

The dataset can be employed as the training and test sets for the following computer vision tasks: face attribute recognition, face recognition, face detection, landmark (or facial part) localization, and face editing & synthesis.

b) Prepare the Data:

Since we're focusing on the Model Generating Fake images and already have good quality of dataset, so instead of doing the pre-processing ourselves, we will be using a pre-processed dataset. Although we can download the smaller subset of the **CelebA** dataset from here. And do some processing.

c) Defining Model:

The discriminator as we know is responsible for classifying the images as real or fake. Thus, this is a typical classifier network.

The discriminator architecture consists of 4 convolutional layers, each followed by a leaky ReLU activation function with a negative slope of 0.2. The first convolutional layer takes in an image tensor with 3 channels (RGB) and outputs a tensor with conv_dim channels, where conv_dim is a hyperparameter that controls the number of channels in the feature maps at each layer. The remaining three convolutional layers have 2x the number of channels as the previous layer. Each convolutional layer uses a kernel size of 4 and does not use batch normalization. After the convolutional layers, the output tensor is flattened and passed through a fully connected layer with a single output unit, which produces the final output scalar.

The generator network is responsible for generating fake images that could fool the discriminator network into being classified as real. Over time the generator becomes pretty good in fooling the discriminator.

The generator takes in a noise vector of size (batch_size, z_size) as input and outputs an image tensor of size (batch_size, 3, 64, 64).

The generator architecture consists of 4 transposed convolutional layers, each followed by a ReLU activation function. The input noise vector is first passed through a fully connected layer that maps it to a tensor of size (batch_size, conv_dim8, 2, 2), where conv_dim is a hyperparameter that controls the number of channels in the feature maps at each layer. This tensor is then reshaped to have shape (batch_size, conv_dim8, 2, 2) and is passed through the 4 transposed convolutional layers. The first transposed convolutional layer has conv_dim8 input channels and conv_dim4 output channels, and each subsequent layer has half the number of output channels as the previous layer. Each transposed convolutional layer uses a kernel size of 4 and does not use batch normalization, except for the final layer, which does not use batch normalization and has 3 output channels (corresponding to the RGB channels of the image).

After the final transposed convolutional layer, the output tensor is passed through a hyperbolic tangent (tanh) activation function, which maps the output to the range [-1, 1], corresponding to the range of pixel values in the input images.

d) Weight Initialization:

To help the models converge, we initialized the weights of the convolutional and linear layers in the model based on the original *DCGAN* paper, which says: All weights are initialized from a zero-centered Normal distribution with a standard deviation of 0.02.

e) Loss Function and Optimizer:

These are functions for computing the binary cross-entropy loss between the discriminator's output and the target labels, for the real and fake data cases. The loss is computed using the BCEWithLogitsLoss function, which combines a sigmoid activation and binary cross-entropy loss in a numerically stable way. The target labels are set to 0.9 for the real case and 1 for the fake case, as suggested in the original GAN paper.

The d_optimizer and g_optimizer are Adam optimizers used to update the parameters of the discriminator (D) and generator (G) models, respectively. The learning rate is set to 0.0002 and the beta1 and beta2 coefficients of the Adam optimizer are set to 0.5 and 0.999, respectively. These hyperparameters are commonly used in GAN training and have been shown to work well in practice.

$$-[ylog(\hat{y}) + (1-y)log(1-\hat{y})]$$

(Equation of Binar Cross-Entropy Loss)

f) Training:

The training loop consists of two main parts:

Train the discriminator: In this part, the discriminator is trained on a batch of real images (from the dataset) and a batch of fake images (generated by the generator). The goal is to make the discriminator classify the real images as real (label 1) and the fake images as fake (label 0). This is done by computing the loss of the discriminator on both real and fake images and performing backpropagation to update its parameters.

Train the generator: In this part, the generator is trained to generate images that fool the discriminator into thinking they are real. The generator takes random noise as input and generates fake images, which are then fed into the discriminator. The goal is to make the discriminator classify the fake images as real (label 1). This is done by computing the loss of the generator on the output of the discriminator and performing backpropagation to update its parameters.

The loop is run for several epochs, with the generator and discriminator being updated in an alternating fashion. The losses of both networks are recorded after each batch and printed to the console at a specified interval. Additionally, the generator is evaluated on a fixed set of random noise after each
epoch, and the resulting fake images are saved to disk for later inspection.

IV. Evaluation And Results

The high fluctuation in the Generator training loss is because the input to the Generator Network is a batch of random noise vectors (each of z_size), each sampled from a uniform distribution of (-1,1) to generate new images for each epoch.

In the discriminator plot, we can observe a rise in the training loss (around 50 on the x-axis) followed by a gradual decrease till the end, this is because the Generator has started to generate some realistic image that fooled the Discriminator, leading to increase in error. But slowly as the training progresses, Discriminator becomes better at classifying fake and real images, leading to a gradual decrease in training error.

V. Conclusion and Future Work

We have seen how to generate realistic faces with DCGAN implementation on the celebA dataset. The images generated can be further improved by tuning the hyperparameters. One could also opt for deeper layers than the one here. Doing so would however result in an increased number of parameters which again would take a lot of time to train.

There is, however, a lot of room for development. The lack of processing resources is one of the main limitations of this solution. The act of creating an image requires extensive computational work. Compared to other, more basic picture datasets like MNIST, the images in this sample have a better relative resolution. The pictures are coloured as well. Therefore, using a GPU will significantly reduce run-time, opening the door to running more epochs.

However, simply increasing the number of epochs may not ensure the creation of better images. If the learning rate is too low, the discriminator will begin to outperform it and the images' quality will begin to decline. Increasing the number of layers in the generator is an apparent technique to enhance the model.

References

[1] Liu, M. Y., & Tuzel, O. (2016). Coupled generative adversarial networks.

[2] Grnarova, P., Zurich Kfir Levy, E. Y., Lucchi, A., Zurich Nathanaël Perraudin, E. T. H., Goodfellow Thomas Hofmann, I., Zurich Andreas Krause, E. T. H., et al. (2019). A Domain agnostic measure for monitoring and evaluating GANs.

[3] Brownlee, J. (2019). A Gentle Introduction to Generative Adversarial Networks (GANs). Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ (Accessed: 2 September 2019).

[4] Goodfellow, I. (2016). Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160.

[5] Taigman, Y., Polyak, A., & Wolf, L. (2016). Unsupervised cross-domain image generation. arXiv preprint arXiv:1611.02200.

[6] Bogunowicz, D., & Bogunowicz, D. (2018). The Best of GAN Papers in the Year 2018. DTRANSPOSED. Retrieved from https://dtransposed.github.io/blog/Best-of-GANs-2018-(Part-1-out-of-2).html (Accessed: 1 September 2019).

[7] Brock, A., Donahue, J., & Simonyan, K. (2018). Large Scale GAN Training for High Fidelity Natural Image Synthesis. Retrieved from https://openreview.net/forum?id=B1xsqj09Fm (Accessed: 2 September 2019).

[8] Brownlee, J. (2019). A Gentle Introduction to Generative Adversarial Networks (GANs). Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

[9] Cao, Y., et al. (2019). Recent Advances of Generative Adversarial Networks in Computer Vision. IEEE Access, 7, DOI: 10.1109/ACCESS.2018.2886814.

[10] Creswell, A., et al. (2018). Generative Adversarial Networks: An Overview. IEEE Signal Processing Magazine, 35(1), DOI: 10.1109/MSP.2017.2765202.

[11] Creswell, A., & Bharath, A. A. (2019). Inverting the Generator of a Generative Adversarial Network. IEEE Transactions on Neural Networks and Learning Systems, 30(7), DOI: 10.1109/TNNLS.2018.2875194.

[12] Deng, Z., et al. (2017). Structured Generative Adversarial Networks. ArXiv:1711.00889 [Cs]. Retrieved from http://arxiv.org/abs/1711.00889

[13] Dutt, R. V. S. K., & Premchand, P. (2017). Generative Adversarial Networks (GAN) Review. CVR Journal of Science and Technology.