Biometric Fingerprint Generation using Generative Adversarial Networks

Ogban-Asuquo Ugot and [1]Chika Yinka-Banjo

Abstract The availability of biometric data for cybersecurity studies is very important and the Generative Adversarial Network (GAN) provides a means of generating realistic fake biometric data that may mitigate some of the privacy concerns with using real biometric data. This paper trains a Deep Convolutional Generative Adversarial Network (DCGAN) to generate fake biometric fingerprint images. The quality of the generated images is measured using the NFIQ2 algorithm and although the qualitative analysis of the fake fingerprints indicate that the images are of good quality, the NFIQ2 scores obtained for the fake fingerprints are of low utility value. We also explore fingerprint matching by first extracting the minutiae feature points from both the fake generated fingerprint samples and the real fingerprint images using the MINDTCT technique. The minutiae feature points from the fake fingerprint images are then matched (one-to-many) against feature points extracted from real fingerprint images using the BOZORTH3 algorithm. This is done in order to appraise the GANs ability to generate data points with features that generalize beyond the precise domain of the training datasets features. The results from the fingerprint matching show that the fake generated fingerprint samples contain unique fingerprint images.

Keywords: Deep Learning, Generative Modelling, GANs, Biometric fingerprint, Cybersecurity.

1 Chika Yinka-Banjo

University of Lagos, Akoka, Nigeria

cyinkabanjo@unilag.edu

1 Introduction

A generative model takes a collection of training samples and estimates the probability distribution that describes those training points. Generative models offer two basic functionalities [18];

1. The ability to take a collection of points and estimate a density function that describes the probability distribution that generated them, this is known as density estimation;

2. Observe many samples from a distribution and then learn to create more similar samples from that same distribution.

The model used for fingerprint generation in this paper is a class of generative modelling that falls under the second function. Invented by Ian Goodfellow [20], Generative Adversarial Networks (GANs) offer an alternative means of generating samples (learning the density function) as opposed to directly estimating a well-defined density function that describes the probability distribution defining the original data [19]. The intuitive idea behind GANs is the ability to train two separate deep neural networks using a form of minimax game [20]. The first neural network is called a generator and its goal is to generate fake data samples that are realistic enough to spoof the second neural network, called a discriminator. The discriminator's goal is to classify samples presented as input, as either real or fake. Ideally, the training algorithm for the GAN should enable the generator network to improve on its ability to generate fake samples while the discriminator's ability to distinguish between real and fake samples decreases.

GANs have been used in state-of-the-art realistic image generation tasks such as face generation [83], improving image resolution [44], generative image manipulation [81], [87] and image to text translation [34]. GANs are the state of the art for generating sharp and realistic images in all these applications. GANs were not always this successful though and the success of much more recent image generation tasks is attributed to the Deep Convolutional Generative Adversarial Network (DCGAN) architecture [56]. Initially, when GANs were invented, the training algorithm did not scale very well to large images [19], and although the early GAN architecture made use of deep convolutional networks, DCGANs make key adjustments that result in crisp generated images. Amongst other things, batch normalization across all the layers except the output layer give DCGANs the edge over older GANs [56]. Most modern implementations of GANs especially for image generation make use of the DCGAN architecture, the GAN implementation in this paper follows suit, while making some modifications to enable the GAN generate larger images needed for the biometric fingerprint matching algorithms.

The generative ability of GANs have been extended beyond artistic purposes. According to [77], the generative adversarial network can be adopted for cybersecurity research and application using two approaches. The first approach involves using the GAN to generate realistic data samples that can be used to spoof already existing cybersecurity systems. For example, a recent study shows that GANs can be successfully trained to generate passwords of users [29]. In the study, the generative adver-

sarial network is trained on 9 million samples of user passwords and the results show that the GAN was able to generate passwords that spoofed password authentication systems up to 70% of the time. The second approach involves training a GAN to specifically generate synthetic malware samples which can then be used as training data for a machine learning based cybersecurity system. In MalGAN [32], a machine learning based malware detector is trained on synthetic malware data generated by a GAN. The success of either approach depends on the availability of real cybersecurity datasets that can then be used to train the GAN. Thus, another application for the GAN which supplements the two approaches mentioned is simply using the GAN to generate more realistic looking samples for cybersecurity datasets such as biometric datasets including fingerprint, iris, and palmprint data. This paper follows this approach by training a GAN to generate synthetic fingerprint images.

The aim of this study is to validate the generative abilities of a (Deep Convolutional) Generative Adversarial Network in generating biometric fingerprint data that can be used for biometric authentication studies. We employ an empirical biometric fingerprint evaluation metric (NFIQ2) to measure the quality of the fake fingerprint samples generated by the GAN, then we run a similarity study using the BOZORTH3 algorithm by matching the generated fake samples against the real fingerprint images from the original dataset and against fake fingerprint samples. Sect. 2 presents a history of generative modelling building up to the GAN, we then present theoretical descriptions of the GAN and we end the section by highlighting in more detail, the application of GANs in cybersecurity. In Sect. 3 we present the GAN model used for the fingerprint generation. Sect. 4 presents the training and experimental setup while Sect. 5 presents the results.

2 Related work

Machine learning is the subfield of Artificial Intelligence generally concerned with methods through which an agent can learn useful information from its own experience. A broad field in its own right, machine learning is classified into supervised, unsupervised and reinforcement learning. In supervised learning, a human has to efficiently annotate the learning dataset with clear examples that map input features to an output [19]. The learning agent learns an objective function that minimizes the error between the estimated output and the real output. An agent's performance is quantified by its ability to generalize to novel input and thus correctly estimate the output [72],[3]. Reinforcement learning may rely less on large datasets for training an agent [63]. In reinforcement learning, the agent learns to choose a rational action that maximizes its performance measure within a domain specific environment [71]. The intuition is that when the agent makes a right choice of action, it is rewarded and when it makes a wrong choice it is penalized. The goal is then to maximize a performance score by avoiding penalties as much as possible. The right sequence of actions is usually defined by a policy, and the goal of the reinforcement learning agent is to learn this policy [52]. Unsupervised learning can informally be described as an attempt to learn useful information from data without annotated examples and output target labels. Unsupervised learning is usually associated with, density estimation and learn-

ing to draw samples from a distribution, clustering of data into similar groups amongst many things. Within the context of density estimation, generative modelling in machine learning can be classified under unsupervised learning. Here, a generative model takes a collection of training examples and tries to infer the probability distribution that describes those training points [19].

2.1 Deep Learning

Loosely inspired by biological neurons in the brain, artificial neural networks [61] are a major research area in machine learning and artificial intelligence and has been applied extensively in diverse areas including healthcare [16], [53], network security [30], timeseries predictions [26], [47] and engineering [26], [48], [54], [77], [80]. Deep learning is part of a broader family of machine learning methods and artificial neural networks (see Fig. 1) based on the ability to learning data representations using multilayer "deep" neural networks. Learning can be supervised, semi-supervised or unsupervised. This section, presents a review of the subject. The guideline and significant topics presented here follow the organizational structure of the subject in [19].

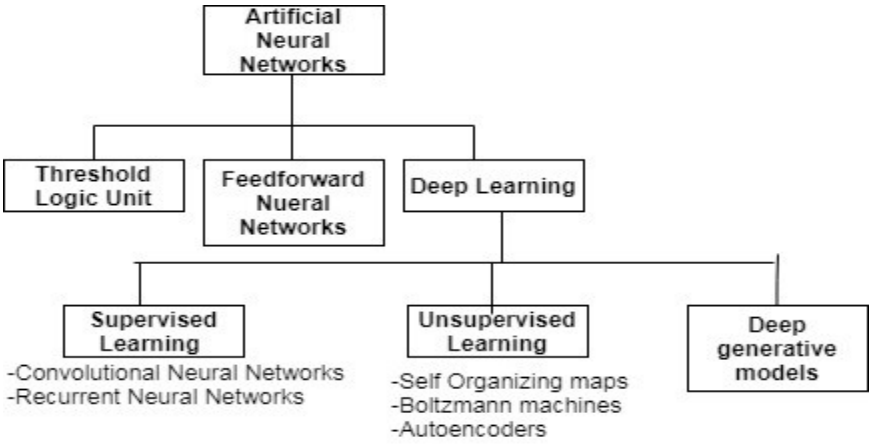

Fig. 1 Artificial Neural Networks Taxonomy

2.1.1 Feedforward Neural Networks

Feedforward neural networks otherwise known as Multilayer Perceptron (MLP) are at the foundation of what is now known as deep learning. The ultimate goal of a feedforward network is to approximate some function f^{ι}. Feed forward networks define a mapping $y=f(z;\theta)$ and learn the value of the parameters θ that result in the

best function approximation f^{i}. A feedforward network is typically represented by composing together many different functions for example;

$$f(z)=f^{(3)}(f^{(2)}(f^{(1)}(z)))\, f(z)=f^{(3)}(f^{(2)}(f^{(1)}(z)))\, f(z)=f^{(3)}(f^{(2)}(f^{(1)}(z))) \quad (1)$$

In (1), $f^{(1)}$ is called the first layer of the network, $f^{(2)}$ is called the second layer and so on, z is the input vector. The length of the chain is called the depth of the model. The final layer of a feedforward network is called the output layer and the first layer is called the input layer. The layers in between the input and output layer are called the hidden layers because the desired output of the hidden layer is not defined in the training data.

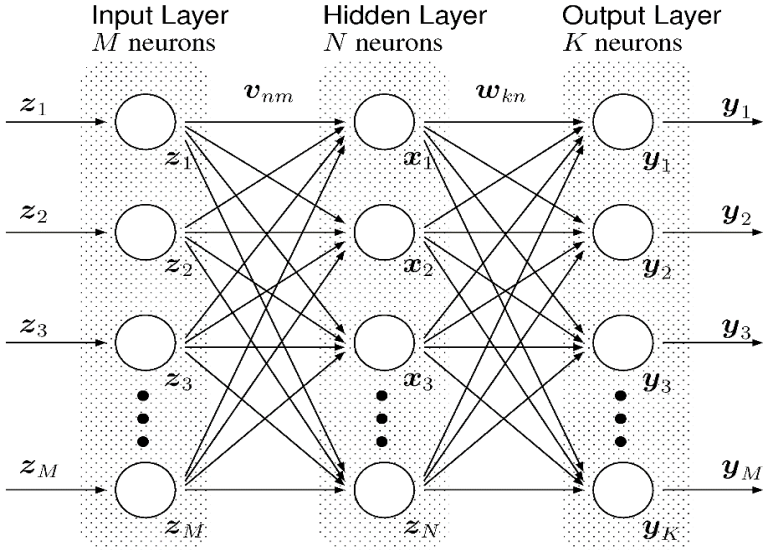

Fig. 2 Feedforward neural network showing the input, hidden and output layer

Activation functions are therefore required to compute the hidden layer values. The feedforward network topology is known as its architecture and architectural designs must consider the number of layers and how these layers are connected. Learning in feedforward networks is achieved through back propagation algorithm which is used to compute the gradients needed to adjust the weights of the model [41].

A simple 3-layer feedforward network is shown in Fig. 2, in the diagram v_{nm} is a vector of weights from the input layer to the hidden layer and w_{kn} is the vector of weights from the hidden layer to the output layer. These are the weights which the backpropagation algorithm optimizes [59]. We can specify the complete network as an extension of (2) as follows;

$$f(z, v_{nm}, c, w_{kn}, b) = w_{kn}{}^{T} max\left(0, v_{nm}{}^{T} z + c\right) + b \quad (2)$$

In (2) and relative to Fig. 2, c and b are the bias for the input and hidden layers respectively. Also, (2) makes use of the rectified linear function to transform ($max\{0, x\}$) the signal from the input layer, where $x = v_{nm}{}^{T} z + c$, this represents the vector product and summation of the input to the hidden layer.

2.1.2 Gradient Based Learning

Designing and training a machine learning model usually involves specifying an optimization procedure, a cost function and a model family. The procedure in training deep learning models is no different, and one must be careful in understanding the nature of deep learning family of models. Unlike linear models such as linear regression, deep learning models and neural networks in general exhibit a nonlinearity which causes the cost function to become non-convex [8]. This means that neural networks have to be trained using iterative, gradient-based optimizers that help to drive the cost function to a very low value [59].

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(w_i) \quad (3)$$

The most widely used and accepted of these optimization algorithms is the gradient decent. Shown in (3), the gradient decent algorithm makes use of the gradient of the cost function ($\frac{\partial}{\partial w_i} Loss(w_i)$), together with α (the learning rate) to obtain the optimal weights. This product $\alpha \frac{\partial}{\partial w_i} Loss(w_i)$, when iteratively subtracted from the weights of the model is guaranteed to drive the weights to a local minimum [59]. Modern implementations of gradient decent algorithm take the form of Adaptive Moment estimation (ADAM) which according to Kingma and Ba [40] improves stochastic gradient decent by computing individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

2.1.3 Cost Functions

Gradient decent enables us to optimize the weights that minimize the cost function for the deep network. A cost function is simply the objective function a deep network is trying to minimize [73]. The choice of a cost function is extremely important and we can define cost functions within various context. Some important cost functions are reviewed in the following subsections.

Learning Conditional Distributions

When we describe the task of training a feed forward network simply as performing maximum likelihood $p_{model}(y/x)$, the cost function can be defined as the negative loglikelihood also known as the cross-entropy between the training data and model distribution [19]. This is defined by (4);

$$J(\theta) = -E_{x\ p_{data}} \log P_{model}(y/x). \tag{4}$$

An advantage of this cost function in (4), is that one does not need to design a cost function every time for a specific deep model. The cost function is explicitly defined by $\log P_{model}(y/x)$.

Learning Conditional Statistics

Instead of learning the probability distribution $p(y/x;\theta)$, we can learn just one property or statistic of y given x. For example, we may have a prediction $f(x;\theta)$ that we wish to use to predict the mean of y. From this point, we can view the cost function as being functional rather than being just a function.

$$mse = \frac{1}{n}\sum_{n=1} (\hat{y} - y)^2 \tag{5}$$

An example of such a cost function is the mean squared error in (5), others include the mean absolute error and r-squared error [73].

Output Units

The choice of the cost function is also dependent on the choice of output unit. An output unit provides an additional transformation from the hidden layer to complete the task and provides the final output signal [8]. The following output units can be used;

1. **Linear Units:**

   We can define this unit as;

   $$f(x) = ax \tag{6}$$

   The linear function in (6) is used in an output layer and will simply transform the input x into ax.

2. **Sigmoid Units:**

   The sigmoid function is widely used for output units for Bernoulli output distributions and is defined as follows;

   $$f(x) = 1/(1+e^{-x}) \tag{7}$$

   The sigmoid function (7) ranges from 0 to 1 and is desirable for classification tasks. The derivative is smooth and dependent on x making it very suitable for backpropagation [51].

3. **Softmax Units:**

Softmax units are also a type of sigmoid function that are suitable for multinouli distributions where the task is multi-class classification. The function is defined in (8);

$$\sigma(z)j = \frac{e^{zj}}{\sum_{k=1}^{k} e^{zk}} for j = 1, \ldots, k \tag{8}$$

2.1.4 Hidden Units

The choice of the hidden units is also an important factor of overall network performance. In general, the hidden units should be differentiable functions capable of performing affine transformations on their input signals, these functions are usually regarded as activation functions. The rectified linear unit (ReLU) is one of the most widely used activation functions for the hidden units and is defined as follows [51];

$$f(x) = max\{0, x\} \tag{9}$$

The ReLU function (9) is nonlinear, which means we can backpropagate the errors and have multiple layers of neurons being activated by the ReLU function [51]. Other activation functions for the hidden units include the hyperbolic tangent (Tanh) and leaky ReLU.

2.1.5 Backpropagation

The history of backpropagation dates as far back as the year 1960 within the context of control theory [37]. In 1986, Rumelhart, Hinton and Williams [58] showed that backpropagation can be used to successfully train multilayer neural networks allowing them form internal representation of data in their hidden layers.

To understand how the error backpropagation algorithm works, we must first establish how the error is formed in the first place. When we use a feedforward neural network to accept an input x and produce an output $\hat{y}$, information flows forward through all the hidden layers and finally produces $\hat{y}$ at the output layer. This is process is called feedforward propagation. The output from a feedforward network $\hat{y}$ may be called the estimated or predicted output and we can then use a cost function to calculate the difference between the estimated output and the real or desired output defined in our dataset. This difference produced by the cost function is known as the error [40].

Calculating this error at the output layer is easy because we know what the desired output should be as defined in the dataset. This is different for the hidden layers since we have no defined output for the hidden layers. The backpropagation algorithm allows us to use the error calculated by the cost function to then flow backwards through the network in order to calculate the gradient [59]. We can think of the gradi-

ent as the partial derivative of a function with respect to its input, this way, we are simply calculating how each parameter on every layer in the network contributes to the overall output calculated by the cost function.

This explanation is better represented symbolically, let f, h and g be functions representing our input, hidden and output layers respectively and let $x, z,$ and w be vectors from the input, hidden and output layers respectively. We can then represent (1) as follows;

$$\hat{y} = g(h(f(x))) \tag{10}$$

In (10), $\hat{y}$ is the predicted output. We can also extend (10) by writing expressions for f and h separately, here $z = f(x)$ and $w = h(z)$ therefore $\hat{y} = g(w)$.

To calculate the gradients, backpropagation uses the chain rule of calculus, which states that if we have two functions $y = g(x)$ and $z = f$ ¿ then;

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{dy}{dx} \tag{11}$$

We can generalize (10) for (11) presented as follows;

$$\frac{\partial \hat{y}}{\partial x} = \frac{\partial w}{\partial z}\frac{dz}{dx} \tag{12}$$

From (12), the backpropagation algorithm calculates the gradient of the error on the output layer with respect to input x as the product of the gradient of the previous functions (layers) $h(z)$ and $f(x)$ with respect to their own inputs. The algorithm calculates this gradient iteratively on all layers of the feedforward network, the calculated gradients are then used in the gradient decent algorithm to optimize the weights needed to reduce the cost.

Computing a gradient in a computation graph such as the feedforward network with n nodes will never execute more than $O(n^2)$ operations or store the output of more than $O(n^2)$ operations.

2.2 Deep Generative Modelling

The goal for a generative model is realistic data generation and generative modelling is an active research area focusing on the mechanisms that set out to achieve that goal. In its simplest form, generative modelling may be described as the maximum likelihood estimate of a synthetic data distribution given samples from an original dataset [18].

$$\theta^{i} = arg\,max\, E_{x\ p_{data}} \log P_{model}(x/\theta). \tag{13}$$

In (13), x is a vector describing the input, $P_{model}(x/\theta)$ is a density function controlled by the parameter θ. The maximum likelihood consists of measuring the log probability that the density function assigns to all data points and adjusting the parameter θ to increase that probability.

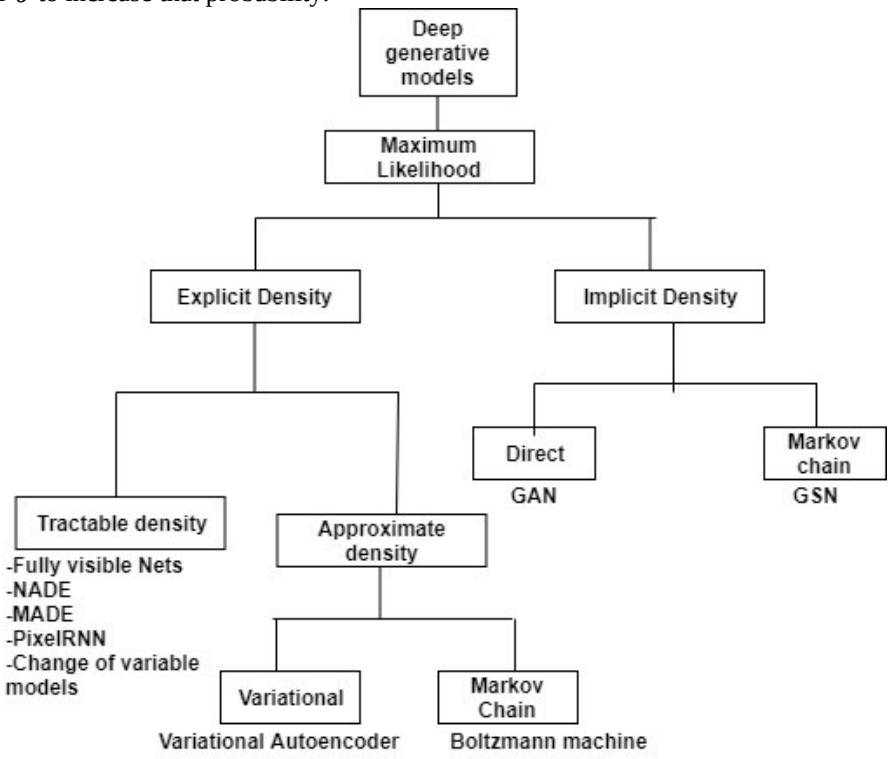

Fig. 3 Taxonomy of Generative models.

Deep generative modelling is the subfield of generative modelling that employs connectionist theories primarily centered around artificial neural networks. In Fig. 3, we show the taxonomy of deep generative models whose methods can be classified as a maximum likelihood estimation. The difference between these models lies in the way they perform the maximum likelihood estimates described in (13). In the following subsections, we briefly discuss notable generative models from the taxonomy in Fig. 3 in a bid to setup the introduction of generative adversarial networks. We highlight the key features of these models, their strength and weaknesses.

2.2.1 Explicit Density

The taxonomy in Fig. 3 is classified into two broad classes, explicit and implicit density. When the density function P_{model} in (13) is clearly defined with parameters that can be evaluated on the training data, we say that the generative model makes use of an explicit density function. However, defining a precise density function is a non-trivial task especially for intractable distributions over natural images and speech data. In such scenarios where the problems are intractable, the explicit density function is no more a clearly defined model but rather an approximate model. Thus, we classify the explicit density into tractable and approximate.

2.2.1.1 Tractable Density

Fully Visible Belief Nets (FVBN):

Fully visible belief nets were introduced in [17] and they make use of the chain rule of probability to define the probability distribution of a vector as the product over each member of the vector.

$$P_{model}(x) = P_{model}(x¿¿1)\prod_{i=2}^{n} P_{model}(x_1 \vee x_i, \ldots, x_{i-1})¿$$

(14)

In (14), the probability distribution of x_1 is multiplied by the distribution of x_2 given x_1. This is repeated for all members of the vector. There are other notable implementations of the fully visible belief net, that make key improvements over the original models such as the PixelCNN [55] and WaveNet [70] used for realistic audio generation.

FVBNs generate good quality data, however a major disadvantage of these class of models is that the cost for generation is $O(n)$which is slow. Another important disadvantage is that the family of FVBNs do not make use of a latent code. The latent code provides a means of controlling the sampling quality of the model which leads to the generation of higher quality images as we shall see the in case of other models such as GANs [18].

Change of Variables

The Nonlinear ICA [33] is a notable example of another class of explicit tractable density models based on the change of variables. The change of variables is used to transform a simple distribution like a gaussian into another distribution using nonlinear functions.

$$y = g(x) \Rightarrow px(x) = py(g(x)) \vee det\ (\frac{\partial g(x)}{\partial x}) \vee ¿$$

(15)

In (15), using the nonlinear function $g(x)$ we can transform a latent space containing x into a space of natural images.

One drawback of these family of models is that the dimension of the input data must be the same as the latent variables. This places a constraint on the design of solutions for a wide range of problems using these models. Another drawback is that the models have to be invertible and the determinant of the Jacobian must be tractable.

2.2.1.2 Approximate Density

These are generative models that make use of a tractable approximation of an intractable density function [18].

Variational Autoencoder (VAE)

The most notable of the approximate density models is the variational autoencoders [41], [57]. The basic idea is to write down a density function $\log p(x)$ as shown in (16).

$$logp(x) \geq logp(x) - D_{KL} ¿ \qquad (16)$$

In (16), the latent variable z needs to be estimated and marginalized, this makes the density function intractable. The variable z is a vector that describes a lower dimensional encoding of the input image. The variational autoencoder defines a lower bound on the probability distribution q that estimates the posterior distribution describing z as illustrated in (17) [18];

$$L(x;\theta) \leq \log pmodel(x;\theta) \qquad (17)$$

The challenge with the variational autoencoder is that the estimates of q must be highly efficient or else the model will be asymptotically inconsistent [41]. Another important setback is the quality of generated data, were samples tend to have lower quality. However, a recent study has shown that careful architectural selections such as deeper models improve quality of generated data [69].

Markov Chain Approximations

The Boltzmann machine is the primary model that makes use of Markov Monte Carlo approximation methods to approximate an intractable explicit function [4], [27], [28]. The Boltzmann machine is defined by an energy function whose probability is proportional to the power of the energy function. The intractability sets in when we have to renormalize the power of the energy function into an actual probability by dividing the sum over all different possible states. Using Monte Carlo approximation methods [19], we are able to approximate this density function. Monte Carlo approximations like this fail to mix between different modes and also perform poorly in high dimensional spaces because they break down. Due to these setbacks, Boltzmann machines are rarely used for the for large-scale image generation tasks.

2.2.2 Implicit Density

The implicit density models interact directly with the density function P_{model}. Instead of explicitly defining the density function, these models learn the function by sampling from it. These models constitute the second branch of the taxonomy in Fig. 3. One notable model under these family of models is the generative stochastic network [9] that makes use of a Markov chain transition operator run several times to obtain a sample from the model. Apart from the cost of training the stochastic network, the models struggle with high dimensional spaces typical of large image generation tasks. The other implicit density model directly samples from P_{model} and is able to generate samples in a single step. At the time of their introduction, GANs were the only notable member of this family, but since then they have been joined by additional models based on kernelized moment matching [45], [14].

2.3 Generative Adversarial Networks (GANs)

We conclude our discussion of generative modelling by discussing the generative adversarial network in more detail. First, we outline the key properties that set GANs aside from other models in Fig. 3. We can outline these properties as follows [18];

1. Latent code: The latent code provides a means of controlling the sampling quality of the generative model. GANs and other models like the Boltzmann machine and VAEs that make use of a latent code have an edge over other models such as the FVBNs.
2. Asymptotically consistent: However, unlike VAEs, GANs are asymptotically consistent. The consistency is dependent on if we are able to reach the Nash equilibrium of the game played by the discriminator and generator models of the GAN. According to Goodfellow [18], the Nash equilibrium if found, guarantees that an accurate probability distribution describing the training data has been found.
3. No Markov chains: GANs do not make use of Markov chain Monte Carlo techniques. This avoids setbacks such as struggling with high dimensional spaces. GANs have been proven to do well in high dimensional spaces [56].
4. High quality generated data: Although there are no empirical means to judge the quality of generated data, qualitative results show that GANs consistently produce high quality generated data.

2.4 How GANs Work

The GAN consists of two neural network models set against each other in the sense of adversarial game theory [20]. The goal for the two networks is to maximize their performance in a game that has a well-defined payoff function. Within the framework, the two different neural networks are known as the *Generator* and *Discriminator*. We discuss the working of the two models making up the GAN in the following subsections.

2.6.1 Generator

The generator (G) is the primary network in the GAN because it performs the task of sampling and generating new data points. The generator is represented as a deep convolutional neural network, however in theory it can be represented by any model with a differential property [20]. Symbolically, we can describe the generator network as follows;

$$x = G(z; \theta^{(G)}) \tag{18}$$

In (18), z contains the latent code while x is the observed variable generated by the generator. According to Goodfellow [18], it is important that x has a higher dimension than z so that the generator does not generate samples from a lower dimensional manifold within x space.

2.6.2 Discriminator

The discriminator's (D) role is to classify input data as real or fake. Similar to the generator, the discriminator can be represented with any differential model but is most commonly represented as a deep convolutional neural network [79]. The discriminator role is simply a binary classification task distinguishing between real and fake samples.

2.5 Training a GAN

The training process for the GAN (See Fig. 5) begins by sampling the latent vector z from the prior distribution over the latent variables, $z$is essentially a vector of unstructured Gaussian noise. It controls the quality of the generated data as well as ensuring that generated samples are from different areas of the distribution space [18].

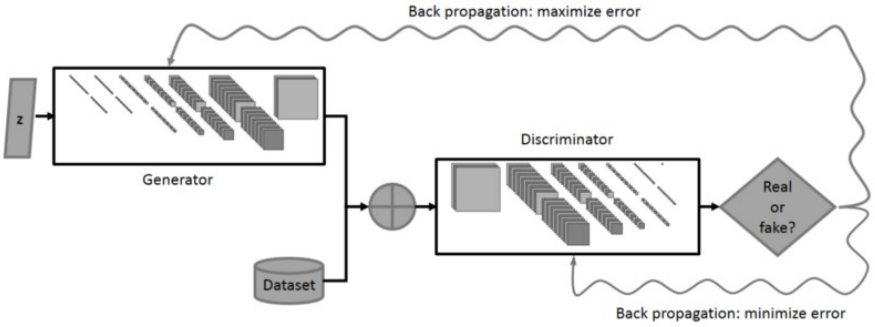

Fig. 4 The basic Generative Adversarial Network architecture.

The vector z is fed into the generator model to get a generated sample as output. The sample from the generator is then fed into the discriminator. The discriminator performs a binary classification and the cross-entropy loss is calculated. The com-

puted error is then backpropagated to both the generator and discriminator networks. Training is typically stopped when the discriminator is unable to accurately classify data as real or fake as indicated from the loss. This point is known as the saddle point and in theory should be the global minimum in the network parameter space. In practice, the discriminator is run on two separate minibatch samples from the training dataset (the real samples) and the generated samples respectively. We can use a modern deep learning optimization algorithm like Adaptive Moment estimation (ADAM) on the two minibatches and the gradient decent is carried out for the generator and discriminator simultaneously.

2.6 Cost Function for the GAN

GANs do not directly infer the probability distribution $p(z/x)$. Instead, from the prior, we sample values of z and then sample values of x from $p(x/z)$. This property of the GAN makes the training algorithm simple and successful because it avoids the problem of directly inferring the probability distribution defining the data [18]. Another key property contributing to the GANs success is the cost function for discriminator (D) and generator (G¿ network. The cost function for the two networks is defined in (19) and (20).

$$J^{(D)} = \frac{-1}{2} E_{x\ p_{data}} \log D(x) - \frac{1}{2} E_{x\ p_{data}} \log\left(1 - D\left(G(x)\right)\right) \tag{19}$$

$$J^{(G)} = -J^{(D)} \tag{20}$$

In (19), $J^{(D)}$ is the cross entropy between the discriminator's prediction and the correct labels. The generator cost $J^{(G)}$ in (20) minimizes the log probability of the discriminator cost function represented as the negated value of $J^{(D)}$. One can think of this as having a single value the generator is trying to minimize and the discriminator is trying to maximize [20].

2.6.1 Non-Saturating Game

Negating the cost function for the discriminator in (20) has some unwanted consequences. If the generator consistently fails to generate fake samples that fool the discriminator, then the generator's gradient over those training iterations will be extremely small and close to zero. This is because the output of the discriminator has saturated and essentially there is no training happening in both networks. Instead of negating $J^{(D)}$, Goodfellow [18] proposes a solution to this problem by changing the order of the arguments to the cross-entropy function. This means that rather than trying to minimize the log probability of the right answer given by the discriminator, we have the generator trying to maximize the probability of the wrong answer.

At this point it is no longer possible to describe the equilibrium with just one function as shown in (19) and (20). The generator cost function is now defined as follows;

$$J^{(G)} = \frac{-1}{2} E_z \log D(G(z)) \tag{21}$$

Both functions in (20) and (21) monotonically decrease in the same direction but are steep in different places. GAN inventor Goodfellow [18] advocates for the use of the cost function in (21) for the generator network, because it solves the problem of non-saturating game.

2.7 Deep Convolutional Generative Adversarial Network (DCGAN)

The DCGAN was introduced in [56], to solve the problem of generating images with high dimensions. The techniques used in DCGAN bypasses other techniques such as using hand designed Laplacian pyramids to scale to large images [13].

The DCGAN lays emphasis on sequential convolutional layers while making use of batch normalization techniques [35] on every layer except for the last layer of the generator network.

The architecture for the DCGAN does not contain pooling layers and the spatial dimension of the feature maps is increased using transposed convolutions with a stride greater than 1, [64]. This done in order to achieve better image resolution. The transposed convolution is an inverse of the convolution operation and is a form of image reconstruction given the filters and activations [78].

2.8 Cybersecurity and GANs

This section briefly presents some notable studies that successfully apply generative adversarial networks in cybersecurity systems. The following subsections present studies with research goals that can either be classified as building adversarial systems or strengthening security systems. A more detailed review of the studies highlighted here can be found in the review paper [77].

2.10.1 Improving Adversarial Detection and Data Privacy

The primary challenge for a cybersecurity systems is knowing what type of attack to expect. This can help to strengthen the process of setting up precautionary systems for mitigating attacks [50]. For machine learning based security systems, the knowledge of the possible attacks helps researchers to create robust training data for the model. The training dataset must contain features that that have been sampled from a wide array of attack scenarios [2], [10], [46]. However, there is a no guarantee of finding large training datasets over a specific problem domain and the process of creating datasets is a manual task involving a lot of time and effort. The GAN provides a means of creating realistic samples that could serve as large datasets for cybersecurity studies. Using the GAN, one can train and sample an unlimited number of samples from various problem domains across different security attack types. Studies have shown that machine learning based security system are able to accurately detect novel adversarial attacks after training on GAN generated samples [23], [38], [75].

For more data centric applications of GANs in cybersecurity, one can review studies where GANs have been used for stenography and cryptography [25], [62]. The research goal here is to use the GAN to learn the encoding and embedding space. The GAN is then used to generate encoded samples [1], [62], [82]. Data privacy is another area where GANs have been applied. Researchers in [31], present a technique termed Generative Adversarial Privacy (GAP), an innovative context-aware model of privacy that allows the GAN learn privatization mechanisms from the dataset without requiring access to the dataset statistics. Other data privacy techniques using GANs involve using the GAN to induce noise into the sample data in a way that preserves data privacy [36], [74], [7], [68], [39].

2.10.2 Creating Adversarial Systems

Adversarial systems like PassGAN highlighted in Sect. 1 demonstrate the potential of using GANs for creating adversarial systems. Recall that in the PassGAN study, the GAN was trained on real password data and was able to generate fake passwords that matched 51-73% passwords in a test set [29].

Another study uses a GAN trained on real biometric fingerprint data to generate fingerprint minutiae features that were then used to create "masterprints". The results show that on a commercial fingerprint system, the masterprint matches to 22% of all users in strict security settings, and 75% of all users in looser security settings [11]. Adversarial systems like PassGAN can be used to spoof authentication systems.

Generating adversarial data is another key area where GANs have been applied. The integrity of a security system that makes use of a machine learning classify is reliant on the ability of the model to have more true-positives than false-positives or true-negatives. Adversarial data are samples that have the ability to fool a classifier to classify the input as a false-positive or a true-negative [5], [13], [21], [42], [15]. GANs have been trained to learn the embedding space for adversarial data and then used to generate more adversarial data samples with introduced adversarial perturbations [43], [24]. Adversarial image data can be used to spoof authentication systems that make use of image recognition software.

On the subject of creating malware, malware authors often design malware without having much detailed knowledge of the architecture of Malware detectors. MalGAN was introduced as a GAN based malware generator. MalGAN is designed to generate malware for blackbox attacks where only the knowledge of input features for the detector are known. The authors of the study [32], use a gradient based approach to generate adversarial android malware examples.

3 Model Architecture for the Biometric Fingerprint GAN

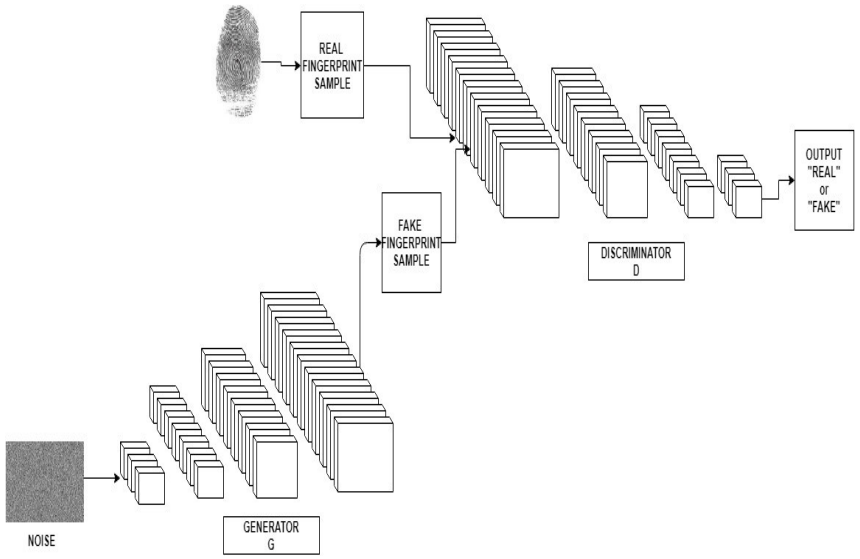

Fig. 5 Block diagram of the Generative Adversarial Network architecture for fingerprint generation

In order to generate synthetic fingerprint data, our GAN model makes use of two deep convolutional neural networks. In Fig. 5, the first neural network is the Discriminator D and it is a conventional convolutional neural network with the sigmoid function as the output node. The second neural network, is the generator, responsible for the actual generation of the synthetic fingerprints. The generator G is a deep convolutional neural network that makes use of the convolution operation. The two networks play the minimax game against each other in a bid to maximize a payoff. The loss function used for the discriminator and generator are shown below in (22) and (23) respectively.

$$J^{(D)} = \frac{-1}{2} E_{x p_{data}} logD(x) - \frac{1}{2} E_{x p_{data}} \log\left(1 - D\left(G(x)\right)\right) \tag{22}$$

In Eq 22, $J^{(D)}$ is the cross-entropy between the discriminator's prediction and the ground truth, $J^{(D)}$is the cross-entropy loss for binary classifications on both the real samples from the training set and the fake sample data generated by the generator neural network.

$$J^{(G)} = \frac{-1}{2} E_z \log D\left(G\left(z\right)\right) \tag{23}$$

In (23), we have the generator trying to maximize the probability of a wrong classification by the discriminator. Both networks are trained simultaneously using the ADAM version of stochastic gradient descent. The ADAM optimizer [40] is used to optimize the parameters of the two neural networks as a function of their gradients. The gradients are calculated using the backpropagation algorithm. We present the generator and discriminator network architectures in more detail in the next subsections.

3.1 Generator Architecture

The generator consists of 1 transposed-convolutional layer and 7 convolutional layers. The first transposed layer is the input layer with Gaussian noise and it has dimension (1x1x4096), the output layer is a 2-dimensional convolutional layer with shape (256x256x8). As shown in Fig. 6, each convolutional layer is followed by a 2-dimensional upsampling layer. This enables the GAN generate larger images of dimension (256x256) by repeating the rows and columns of the layer by size 2. All of the model's convolutional layers use a stride of 2 except for the first layer. The stride controls the cross-correlation across the input image. The padding parameter controls the amount of implicit zero-paddings on both sides of the input image and is set constant at 1 on all layers.

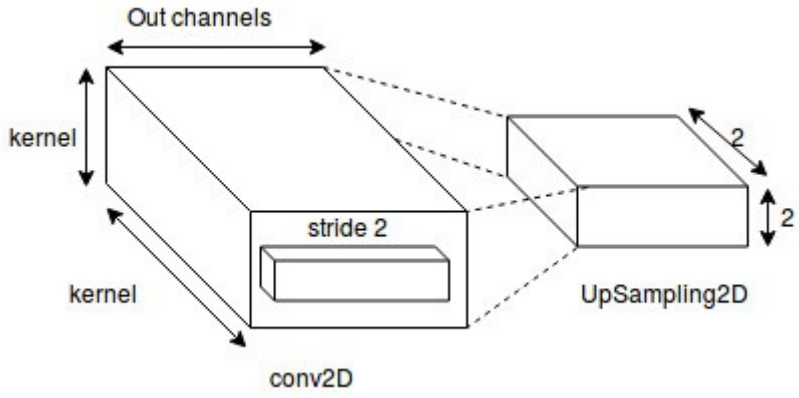

Fig. 6 Block diagram showing a typical covolutional layer in the Generator architecture

After each convolutional layer, we normalize all the features along the dimension of the batch and apply a ReLU rectification to break the linearity. Regularization [35] using the dropout technique is applied on each layer before upsampling.

3.2 Discriminator Architecture

The discriminator network is a convolutional network with 8 convolutional layers and one fully connected layer with an output. The first layer takes in an image and applies the convolution operation and we add Gaussian noise to avoid overfitting. A stride of 1 is used on all layers so as to retrieve enough information as much as possible.

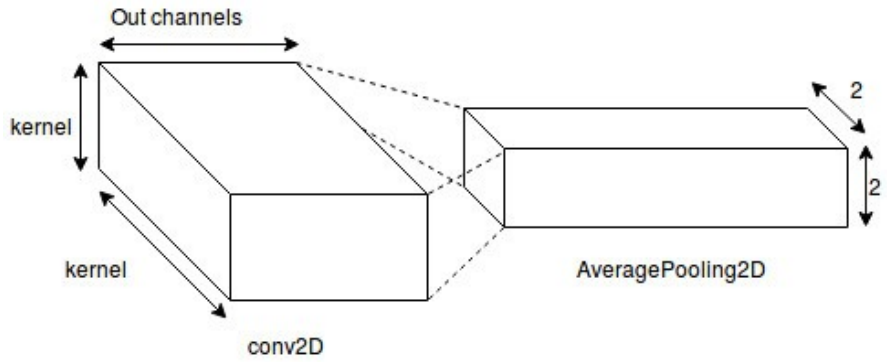

Fig. 7 Block diagram showing a typical convolutional layer in Discriminator architecture

Unlike the generator network (See Fig 7), each convolutional layer in Fig. 14 is followed by an average pooling layer. Also, instead of using the ReLU activation function, the LeakyReLU activation function is used after every convolutional layer in the discriminator. LeakyReLU will take the negative slope of the function until it gets to 0.2. These parameter choices have been chosen for best performance based on well researched work [87]. Next, we normalize all the features along the dimension of the batch and finally the output layer of the fully connected network makes use of the sigmoid activation function to break the linearity to stay between 0 and 1, representing fake or real classifications.

4 Model Training

The GANs are trained on a training set of 10,000 real fingerprint images for 400 epochs. Then, 5000 fake samples are generated using the trained generator network discussed in Sect. 3. The fingerprint generation is done by simply running a feed forward propagation through the generator network over 5000 iterations, Gaussian noise is fed into the network as input and a fake fingerprint image is produced as the output. The NFIQ2 fingerprint quality assessment is carried out on the fake generated samples. Then, the minutiae feature points of these fake fingerprint images are then used for one-to-many fingerprint matching on both the training set and the test set. Each fingerprint image is given a unique integer identifier that is used for tracking and logging during fingerprint matching.

4.1 Biometric Fingerprint Dataset

The dataset is the CASIA FingerprintV5 containing 20,000 fingerprint images [84]. The images have been captured using a URU4000 fingerprint sensor in one single session. The volunteers of the CASIA-FingerprintV5 comprise of workers, graduate students, waiters etc. Forty (40) fingerprint images were contributed from each volunteer, that is, five images per finger from their eight fingers (left and right thumb, second, third and fourth finger). To achieve various intra-class variations, each volunteer was asked to rotate their finger with various levels of pressure. All the fingerprint images in the dataset are 8-bit gray-level BMP files and have been scaled to a dimension of 256x256. This dimension is the same with that of the output images from the generator neural network. The dataset is split into a training and test set of 10,000 images each.

4.2 Fingerprint Feature Extraction and Matching

NFIQ2
NFIQ2 computes a set of quality features from a raw or WSQ compressed fingerprint image and uses these features to predict the utility of the image [86]. The utility of a fingerprint image is quantified as a quality score between the range of 0 to 100, where 0 means no utility value and 100 is the highest utility value. The NFIQ2 quality score is in compliance with the international biometric quality standard ISO/IEC 29794-1 [85].

MINDTCT
MINDTCT is a minutiae detector that automatically locates and records ridge endings and bifurcations in a fingerprint image. The MINDTCT algorithm involves generating image maps, binarizing the image, detecting minutiae, counting neighboring ridges and accessing minutiae quality [86].

BOZORTH3
BOZORTH3 is a minutiae-based fingerprint matching algorithm that will do both one-to-one and one-to-many matching operations. The algorithm computes a match score between the minutiae from any two fingerprint images to help determine if they are from the same finger. BOZORTH3 accepts minutiae generated by the MINDTCT algorithm.
All three algorithms were developed by the National Institute of Standards and Technology (NIST) and are accessible as part of an open-source software package known as the NIST Biometric Image Software (NBIS) [86].

4.3 Experimental Setup

The source code for the GAN is written in Python using the Pytorch deep learning library [89] and the training is executed on a Google cloud instance virtual machine (VM). The VM instance is an Ubuntu 16.04 operating system and it runs on Intel quadcore processor with one Nvidia Tesla P100 GPU. As stated earlier in Sect. 4.2,

implementations of the fingerprint extraction and matching are provided by the NIST Biometric Image Software (NBIS).

5 Results

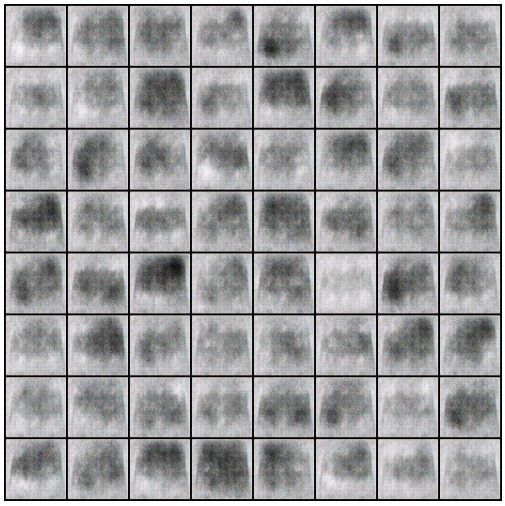

Fig. 8 Generated fingerprint samples from 100th epoch

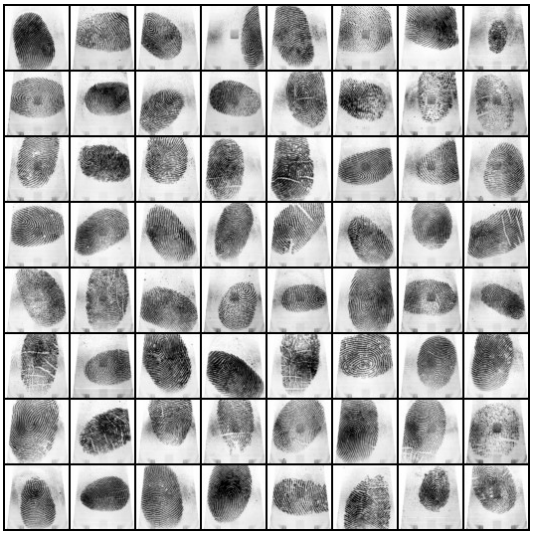

Fig. 9 Generated fingerprint samples from the 400th epoch

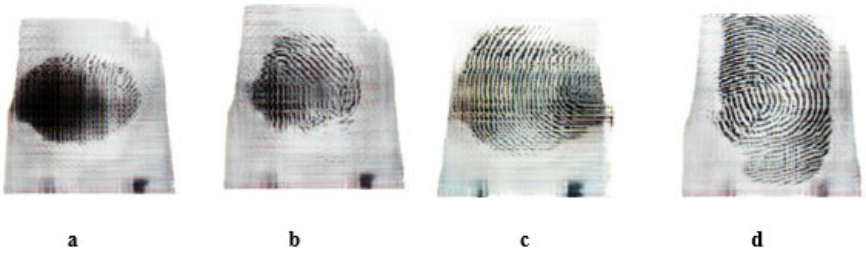

Fig. 10 Individual generated fingerprint samples from **a** 100th epoch **b** 200th epoch **c** 300th epoch **d** 400th epoch

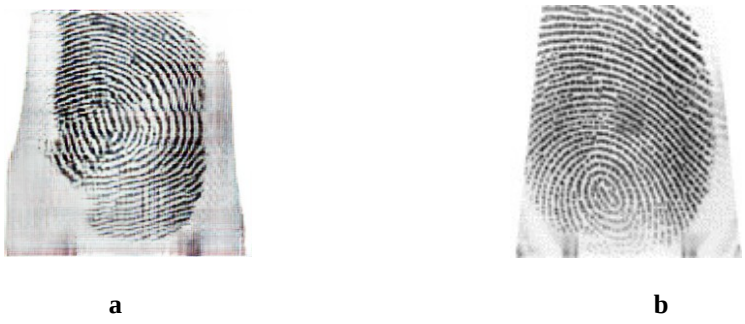

Fig. 11 a Fake generated fingerprint sample **b** Real fingerprint sample

The GAN training results shown in Fig. 8 and 9, show that the qualitative appraisal of the generated fingerprint samples improves as the training epoch increases. This is further highlighted in Fig. 10 where individual samples from the 100th to 400th epochs are shown. The improvement of the qualitative properties of the generated images as the epochs increases is consistent with related works such as [67], [44], [87]. In Fig. 11 we compare an individual generated fingerprint sample with a real fingerprint sample. The results show that the qualitative appraisal of generated fingerprint images is quite acceptable when compared to real fingerprint images. Training beyond 400 epochs did not improve the qualitative properties of the generated images.

5.1 NFIQ2 Fingerprint Quality Analysis

In Fig 12, we present the distribution for the NFIQ2 fingerprint quality score for the fake fingerprint samples generated by the GAN. The mean quality score is 13, which is low. Although qualitative visualization as seen in Fig.11 appears to be of good quality, the NFIQ2 scores indicate otherwise. This indicates that unlike other GAN related tasks, the quality of GAN generated biometric data must be quantitatively measured using standard biometric appraisal techniques.

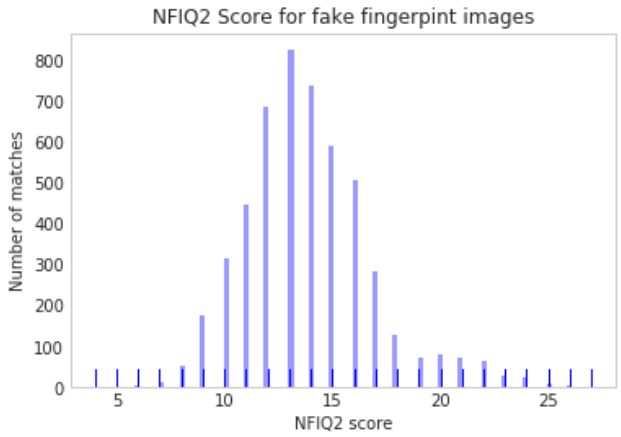

Fig. 12 NFIQ2 fingerprint quality score distribution for the fake fingerprint samples

We compare the quality score for the fake generated samples with that of the real fingerprint images from the training set and test set as shown in Fig. 13 and 14.

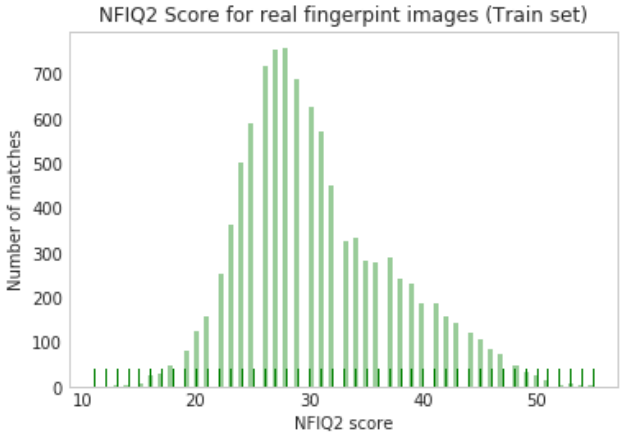

Fig. 13 NFIQ2 Quality score distribution for the real fingerprint samples from the training set

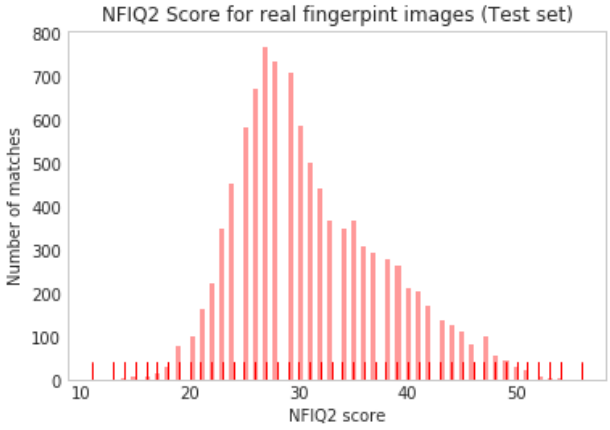

Fig. 14 NFIQ2 Quality score distribution for the real fingerprint samples from the test sets

The NFIQ2 scores for the real fingerprint images are relatively higher with a mean score of 30 and 31 on the training set (Fig. 13) and test set (Fig. 14) respectively. One may presume that there may be a correlation between the NFIQ2 scores of the real fingerprint training samples and that of the GAN generated fake fingerprints. That is, will a training set with high NFIQ2 scores lead to generated samples with high NFIQ2 scores. However, such presumptions are based on false premises because the minutiae feature points extracted for NFIQ2 quality assessments [88], do not make use of the type of convolutional filters found in deep convolutional GANs [56]. In other words, the GAN does not pay attention to the same kind of features that the NFIQ2 algorithm is trained to detect.

It is also important to mention that there are techniques and architectures that could be applied to improve the qualitative visualization of GAN generated images [83], [87]. These techniques such as in [60], [83], require large compute that were not accessible for consideration during the implementation of the GAN model used in this paper. Whether the qualitative improvements offered by other GAN architectures [6], [12], [22], [25], will translate to a higher NFIQ2 score is left for further investigation.

5.2 BOZORTH3 Fingerprint Matching Results

The fingerprint matching results visualized in Fig. 15 and Fig. 16 show a high number of matches with low scores of 20 to 22 which is just above the threshold of 20 set for the BOZORTH3 algorithm. In general, the matching scores are very low across both the test set and the training set. This implies that very few minutiae points are similar between the fake fingerprint images and the real images. A high BOZORTH3 fingerprint matching score (indicating high similarity) is usually within the range of 500 to 600.

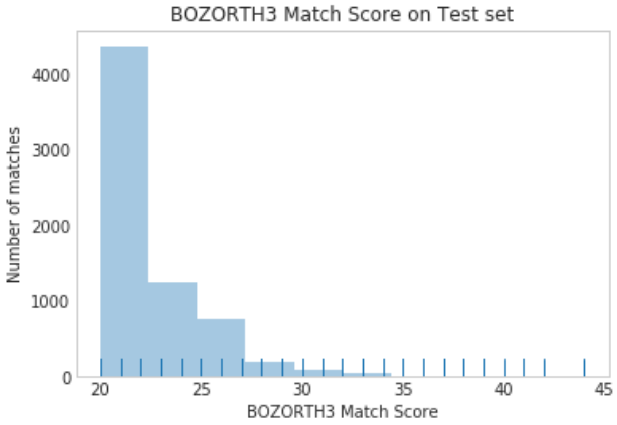

Fig. 15 BOZORTH3 fingerprint matching results between the fake fingerprint images and real fingerprint images in the test set

It is not unusual to have matching results with low BOZORTH3 scores given that a few minutiae points will be matched across two or more fingerprint pairs in a database. Such matches with few minutiae points, will have very low matching scores as is the case with our results.

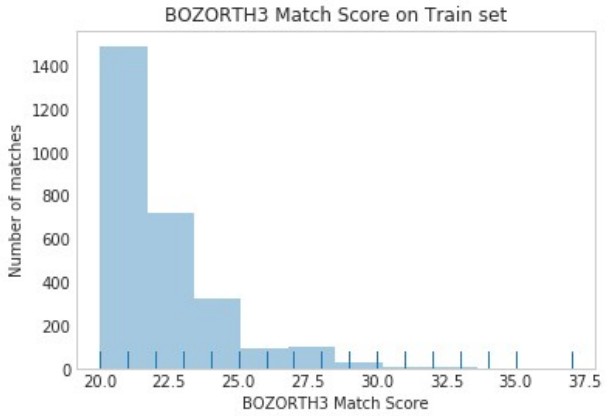

Fig. 16 BOZORTH3 fingerprint matching results between the fake fingerprint images and real fingerprint images in the tes**t set**

Ta**ble 1** Key biometric identification metrics obtained by analyzing the fingerprint matching data

Metric	Training set	Test set
False Rejection Rate (FRR)	0.034%	0.027%
False Acceptance Rate (FAR)	0.012%	0.015%
Average error rate	0.03%	0.02%
Accuracy	99%	99%

The fingerprint matching results show that the fake generated fingerprint images are relatively unique when matched across the test and training set fingerprint images. The FRR is the ratio of false rejections to the total number of matching attempts, while the FAR is the ratio of false acceptance to the total number of matching attempts. In table 1, both the FRR and FAR scores are low, indicating a high accuracy of the BOZORTH3 fingerprint matching algorithm [90].

5.3 NFIQ2 Quality Analysis on Matched Samples

In order to investigate whether or not the NFIQ2 scores obtained influence the BOZORTH3 matching score, we sample the fingerprint images that were matched against the fake and real samples and have a BOZORTH3 score above 30 and then we visualize their NFIQ2 utility scores.

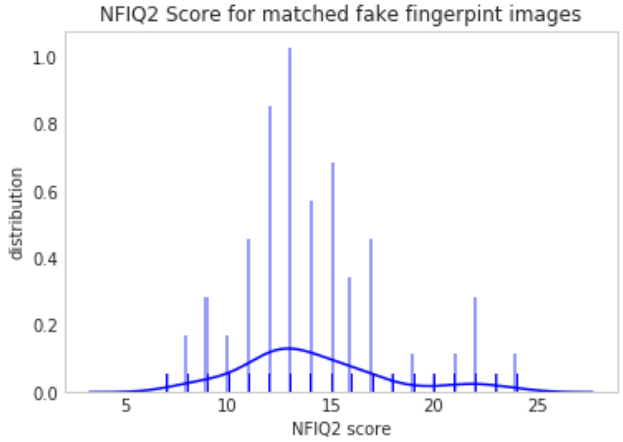

Fig. 17 Visualizing the NFIQ2 score distribution for the matched fingerprint images against fake fingerprint images

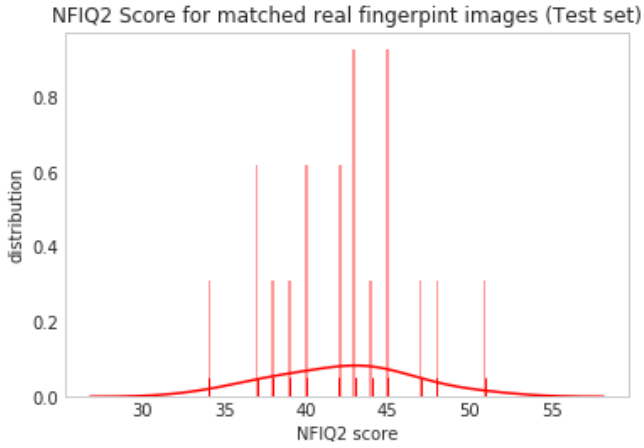

Fig. 18 Visualizing the NFIQ2 score distribution for the matched fingerprint images against fingerprint images in the test set

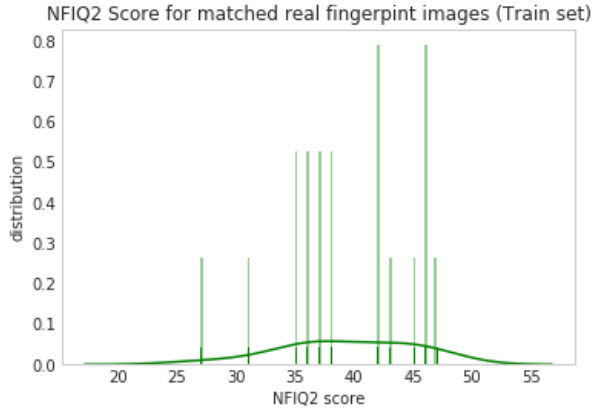

Fig. 19 Visualizing the NFIQ2 score distribution for the matched fingerprint images in the training set

The visualizations presented in Fig. 17, Fig. 18 and Fig. 19 show that the NFIQ2 scores for the three sample sets are Gaussian distributions similar to the distributions in Fig. 12, Fig. 13 and Fig. 14. However, the distributions for the test (Fig. 18) and training samples (Fig. 19) are slightly skewed toward a higher NFIQ2 score. This is reasonable when we consider the average NFIQ2 scores of 14, 39 and 42 from Fig. 17, Fig. 18 and Fig. 19 respectively. From the visualizations in this subsection, we observe that the BOZORTH3 algorithm is able to match fingerprints with relatively low and high NFIQ2 scores but there is evidence that a higher BOZORTH3 score is achieved if the two fingerprints have a high NFIQ2 score.

Table 2 Summary of Average NFIQ2 scores from samples across the datasets

Metric	Generated samples
Generated samples	14
Training set	39
Test set	42

Table 3 Summary of Average NFIQ2 scores from samples that were matched with scores greater than 30

Metric	Generated samples
Generated samples	14
Training set	38
Test set	40

Table 4 Summary of Average BOZORTH3 matching scores

	Generated samples	Training set	Test set
Generated samples	23	20	22

Tables 2 summarizes the average NFIQ2 scores for samples in all three sets while table 3 summarizes the average NFIQ2 scores only for samples that had a BOZORTH3 match score greater than a threshold of 30. Table 2 summarizes section 5.1 while table 3 summarizes section 5.3. Table 4 summarizes the average BOZORTH3 matching scores where the generated samples are matched against all generated samples, samples in the training set and samples in the test set. Table 4 summarizes section 5.2.

6 Conclusion and Future works

The generated fingerprint images generated by the DCGAN are qualitatively realistic but the utility scores from the NFIQ2 algorithm show that the generated images are of low quality within the standards of biometric fingerprint quality metrics defined by the National Institute of Standards and Technology. The results reveal serious implications for measuring the quality of GAN generated biometric data using qualitative visualization only. The NFIQ2 results also show that the DCGAN in this study is unable to generate biometric fingerprint images with high utility values. Finally, the BOZORTH3 fingerprint matching results show that the generated fingerprint samples are unique to the generated samples, the test set and the train set fingerprint images. This indicates that there are no privacy concerns where real fingerprint images have high similarity to the fake generated fingerprint images. Future work with variant GAN architectures should estimate the GANs overall ability to generate high utility biometric data.

References

1. Abadi M, Andersen D.G (2016) Learning to protect communications with adversarial neural cryptography. arXiv preprint arXiv:1610.06918

2. Apruzzese G, Colajanni M, Ferretti L, Guido, A, Marchetti M (2018) On the effectiveness of machine and deep learning for cyber security. In 2018 10th International Conference on Cyber Conflict (CyCon). IEEE

3. Akinyemi M, Yinka-Banjo C, Ugot O.A, Nwachuku A (2020) March. Estimating the Time-Lapse Between Medical Insurance Reimbursement with Non-parametric Regression Models. In Future of Information and Communication Conference (pp. 692-704). Springer, Cham

4. Ackley D.H, Hinton G.E, Sejnowski T.J (1985) A learning algorithm for Boltzmann machines. Cognitive science, 9(1), pp.147-169

5. Anderson H.S, Woodbridge J, Filar B (2016) DeepDGA: Adversarially-tuned domain generation and detection. In Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security (pp. 13-21). ACM

6. Arjovsky M, Chintala S, Bottou L (2017) Wasserstein gan. arXiv preprint arXiv:1701.07875
7. Beaulieu-Jones B.K, Wu Z.S, Williams C, Lee R, Bhavnani S.P, Byrd J.B, Greene C.S (2019) Privacy-preserving generative deep neural networks support clinical data sharing. Circulation: Cardiovascular Quality and Outcomes, 12(7), p.e005122

8. Bengio Y (2009) Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2(1), pp.1-127
9. Bengio Y, Thibodeau-Laufer E, Alain G, Yosinski J (2014) Deep generative stochastic networks trainable by backprop. In ICML'2014

10. Biggio B, Roli F (2018) Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition, 84, pp.317-331

11. Bontrager P, Togelius J, Memon N, (2017) Deepmasterprint: Generating fingerprints for presentation attacks. https://arxiv.org/abs/1705.07386

12. Chen X, Duan Y, Houthooft R, Schulman J, Sutskever I and Abbeel P (2016) Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In Advances in neural information processing systems (pp. 2172-2180)
13. Denton E.L, Chintala S, Fergus R (2015) Deep generative image models using a laplacian pyramid of adversarial networks. In Advances in neural information processing systems (pp. 1486-1494)

14. Dziugaite G K, Roy D. M, Ghahramani Z (2015) Training generative neural networks via maximum mean discrepancy optimization. arXiv preprint arXiv:1505.03906

15. Elsayed G.F, Shankar S, Cheung B, Papernot N, Kurakin A, Goodfellow I, Sohl-Dickstein J (2018) Adversarial examples that fool both human and computer vision. arXiv preprint arXiv:1802.08195

16. Esteva A, Robicquet A, Ramsundar B, Kuleshov V, DePristo M, Chou K, Cui C, Corrado G, Thrun S, Dean J (2019) A guide to deep learning in healthcare. Nature medicine, 25(1), pp.24-29
17. Frey B J, Hinton G E, Dayan P (1996) Does the wake-sleep algorithm produce good density estimators? In Advances in neural information processing systems (pp. 661-667)
18. Goodfellow I (2016) NIPS 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160

19. Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning (Vol. 1). Cambridge: MIT press

20. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A and Bengio Y, (2014) Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680)

21. Goodfellow I J, Shlens J, Szegedy C (2014) Explaining and harnessing adversarial examples arXiv preprint arXiv:1412.6572

22. Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V, Courville A C (2017) Improved training of wasserstein gans. In Advances in Neural Information Processing Systems (pp. 5767-5777)

23. Grosse K, Papernot N, Manoharan P, Backes M, McDaniel P (2017) September. Adversarial examples for malware detection. In European Symposium on Research in Computer Security (pp. 62-79). Springer, Cham

24. Hayes J, Melis L, Danezis G, De Cristofaro E (2019) LOGAN: Membership inference attacks against generative models. Proceedings on Privacy Enhancing Technologies, 2019(1), pp.133-152
25. Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S, Lerchner A (2016) beta-vae: Learning basic visual concepts with a constrained variational framework
26. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L. and Muller, P.A., 2019. Deep learning for time series classification: a review. Data Mining and Knowledge Discovery, 33(4), pp.917-963.
27. Hinton G.E (1984) Distributed representations
28. Hinton G.E, Sejnowski T.J (1986) Learning and relearning in Boltzmann machines. Parallel distributed processing: Explorations in the microstructure of cognition, 1, pp.282-317

29. Hitaj B, Gasti P, Ateniese G, Perez-Cruz F (2017) Passgan: A deep learning approach for password guessing. arXiv preprint arXiv:1709.00440

30. Homayoun, S., Ahmadzadeh, M., Hashemi, S., Dehghantanha, A. and Khayami, R., 2018. BoTShark: A deep learning approach for botnet traffic detection. In Cyber Threat Intelligence (pp. 137-153). Springer, Cham.

31. Huang C, Kairouz P, Chen X, Sankar L, Rajagopal R (2018) Generative Adversarial Privacy. arXiv preprint arXiv:1807.05306

32. Hu W, Tan Y (2017) Generating adversarial malware examples for black-box attacks based on GAN. arXiv preprint arXiv:1702.05983
33. Hyvärinen A, Pajunen P (1999) Nonlinear independent component analysis: Existence and uniqueness results. Neural Networks, 12(3), pp.429-439

34. Isola P, Zhu J.Y, Zhou T, Efros A.A (2017) Image-to-image translation with conditional adversarial networks. arXiv preprint arXiv:1611.07004

35. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift

36. Jordon J, Yoon J, van der Schaar M (2018) PATE-GAN: Generating synthetic data with differential privacy guarantees. In International Conference on Learning Representations

37. Kelley H.J (1960) Gradient theory of optimal flight paths. Ars Journal, 30(10), pp.947-954

38. Kim J.Y, Bu S.J, Cho S.B (2018) Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. Information Sciences, 460, pp.83-102

39. Kim J.W, Moon S.M, Kang S.U, Jang B (2020) Effective Privacy-Preserving Collection of Health Data from a User's Wearable Device. Applied Sciences, 10(18), p.6396

40. Kingma D.P, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980

41. Kingma D. P, Salimans T, Welling M (2016) Improving variational inference with inverse autoregressive flow. NIPS

42. Kurakin A, Goodfellow I, Bengio S (2016) Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533

43. Kros J, Fischer I, Song D (2018) Adversarial examples for generative models. In 2018 IEEE Security and Privacy Workshops (SPW) (pp. 36-42). IEEE

44. Ledig C, Theis L, Huszar F, Caballero J, Aitken A.P, Tejani A, Totz J, Wang Z, Shi W (2016) Photo-realistic single image super-resolution using a generative adversarial network. CoRR, abs/1609.04802

45. Li H, Lin Z, Shen X, Brandt J, Hua G (2015) A convolutional neural network cascade for face detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5325-5334)

46. Lin Z, Shi Y, Xue Z (2018) IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection. arXiv preprint arXiv:1809.02077

47. Lotter W, Kreiman G, Cox D (2016) Deep predictive coding networks for video prediction and unsupervised learning. arXiv preprint arXiv:1605.08104

48. Lu, X., Zhong, Y., Zheng, Z., Liu, Y., Zhao, J., Ma, A. and Yang, J., 2019. Multi-scale and multi-task deep learning framework for automatic road extraction. IEEE Transactions on Geoscience and Remote Sensing, 57(11), pp.9362-9377.

49. Luger, G.F (2005) Artificial intelligence: structures and strategies for complex problem solving. Pearson education

50. Malhotra Y (2018) Machine Intelligence: AI, Machine Learning, Deep Learning & Generative Adversarial Networks: Model Risk Management in Operationalizing Machine Learning for Algorithm Deployment

51. Marsland S (2011) Machine learning: an algorithmic perspective. Chapman and Hall/CRC

52. Mnih V, Badia A.P, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning (pp. 1928-1937)

53. Mohsen, H, El-Dahshan, E.S.A, El-Horbaty, E.S.M, Salem, A.B.M (2018) Classification using deep learning neural networks for brain tumors. Future Computing and Informatics Journal, 3(1), pp.68-71

54. Bae, J.W, Rykhlevskii, A, Chee, G, Huff, K.D (2020) Deep learning approach to nuclear fuel transmutation in a fuel cycle simulator. Annals of Nuclear Energy, 139, p.107230.

55. Oord A.V.D, Kalchbrenner N, Kavukcuoglu K (2016) Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759

56. Radford A, Metz L, Chintala S (2015) Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434

57. Rezende D. J, Mohamed S, Wierstra D (2014) Stochastic backpropagation and approximate inference in deep generative models. In ICML'2014. Preprint: arXiv:1401.4082

58. Rumelhart D.E, Hinton G.E, Williams R.J (1986) Learning representations by back-propagating errors. nature, 323(6088), p.533

59. Russell S.J, Norvig P (2016) Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited

60. Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X (2016) Improved techniques for training gans. In Advances in Neural Information Processing Systems, pages 2226–2234

61. Schalkoff R.J (1997) Artificial neural networks (Vol. 1). New York: McGraw-Hill

62. Shi H, Dong J, Wang W, Qian Y, Zhang X (2017) Ssgan: Secure steganography based on generative adversarial networks. In Pacific Rim Conference on Multimedia (pp. 534-544). Springer, Cham

63. Silver D, Schrittwieser J, Simonyan K, Antonoglou I., Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y (2017) Mastering the game of go without human knowledge. Nature, 550(7676), p.354

64. Springenberg J. T, Dosovitskiy A, Brox T, Riedmiller M (2015) Striving for simplicity: The all convolutional net. In ICLR

65. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I., Fergus R (2013) Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199

66. Tang W, Tan S, Li B, Huang J (2017) Automatic steganographic distortion learning using a generative adversarial network. IEEE Signal Processing Letters, 24(10), pp.1547-1551

67. Theis L, Oord A.V.D, Bethge M (2015) A note on the evaluation of generative models. arXiv preprint arXiv:1511.01844

68. Triastcyn A, Faltings B (2020) Generating Higher-Fidelity Synthetic Datasets with Privacy Guarantees. arXiv preprint arXiv:2003.00997

69. Vahdat A, Kautz J (2020) Nvae: A deep hierarchical variational autoencoder. arXiv preprint arXiv:2007.03898

70. Van Den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K (2016) Wavenet: A generative model for raw audio. CoRR abs/1609.03499

71. Van Hasselt H, Guez A, Silver D (2016) Deep Reinforcement Learning with Double Q-Learning. In AAAI (Vol. 16, pp. 2094-2100)

72. Voyant C, Notton G, Kalogirou S, Nivet M.L, Paoli C, Motte F, Fouilloy A (2017) Machine learning methods for solar radiation forecasting: A review. Renewable Energy, 105, pp.569-582

73. Witten I.H, Frank E, Hall M.A, Pal C.J (2016) Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann

74. Xu C, Ren J, Zhang D, Zhang Y, Qin Z, Ren K (2019) GANobfuscator: Mitigating information leakage under GAN via differential privacy. IEEE Transactions on Information Forensics and Security, 14(9), pp.2358-2371

75. Yin C, Zhu Y, Liu S, Fei J, Zhang H (2018) An enhancing framework for botnet detection using generative adversarial networks. In 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD). IEEE

76. Yinka-Banjo C, Ugot O.A (2018) A Predictive Model for Automatic Generation Control in Smart Grids using Artificial Neural Networks in C Yinka-Banjo & O Ugot, AFRICATEK 2018**:** 2nd EAI International Conference on Emerging Technologies for Developing Countries**,** Cotonou, Benin

77. Yinka-Banjo C, Ugot O.A (2019) A review of generative adversarial networks and its application in cybersecurity. Artificial Intelligence Review, pp.1-16

78. Zeiler M.D, Krishnan D, Taylor G.W, Fergus R (2010) Deconvolutional networks

79. Zeiler M.D, Fergus R (2014) Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham

80. Zhang S, Yao L, Sun A, Tay Y (2019) Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR), 52(1), pp.1-38

81. Zhu J.-Y, Kr¨ahenb¨uhl P, Shechtman E, Efros A A (2016) Generative visual manipulation on the natural image manifold. In European Conference on Computer Vision, pages 597–613. Springer

82. Zügner D, Akbarnejad A, Günnemann S (2018) Adversarial attacks on neural networks for graph data. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 2847-2856). ACM

83. Karras, T, Laine, S Aila, T, (2019) A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4401-4410)

84. CASIA, (2018) Notes on CASIA-FingerprintV5 [online], http://biometrics.idealtest.org/dbDetailForUser.do?id=7 [Accessed on 12 June 2018)

85. ISO/IEC: '29794-1:2009. Information technology–Biometric sample quality–Part 1: Framework'. Tech. Rep., JTC 1/SC 37/WG 3, January 2016

86. Ko, K, (2007) User's guide to NIST biometric image software (NBIS) (No. NIST Interagency/Internal Report (NISTIR)-7392)

87. Brock, A, Donahue, J, Simonyan, K, (2018) Large scale gan training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096

88. Tabassi, E, Wilson, C, Watson, C.I., (2004) Fingerprint Image Qualitiy (No. NIST Interagency/Internal Report (NISTIR)-7151)

89. Paszke, A, Gross, S, Massa, F, Lerer, A, Bradbury, J, Chanan, G, Killeen, T, Lin, Z, Gimelshein, N, Antiga, L, Desmaison, A (2019) Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems (pp. 8026-8037)

90. Zhang, F, Xin, S, Feng, J (2019) Combining global and minutia deep features for partial high-resolution fingerprint matching. Pattern Recognition Letters, 119, pp.139-147