Universidade Federal de Pernambuco
Centro de Informática

Graduação em Engenharia da Computação

Data Augmentation for Offline Handwritten Signature Verification

Adonias Vicente da Silva Barros

Trabalho de Graduação

Recife
Dezembro de 2018

Universidade Federal de Pernambuco
Centro de Informática

Adonias Vicente da Silva Barros

Data Augmentation for Offline Handwritten Signature Verification

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Cleber Zanchettin*

Recife
Dezembro de 2018

To my mother and sister for always support my dreams and believe in me.

Acknowledgements

Agradeço ao meu orientador, o professor Dr. Cleber Zanchettin pelo apoio durante a minha pesquisa e por acreditar no meu potencial. Agradeço aos professores Adriano e Edna pela oportunidade que me deram de conhecer uma outra realidade e mudar a minha visão de mundo. Agradeço à minha mãe Audenice e à minha irmã Estefany por estarem sempre comigo durante essa jornada. Agradeço família e amigos pelo apoio e por acreditarem no meu sucesso. Agradeço à Li-Ting por sempre acreditar em mim. Agradeço ao Centro de Informática da UFPE, professores e funcionários que me moldaram como pessoa e profissional e a todos da Document Solutions.

Consagre ao Senhor tudo o que você faz, e os seus planos serão bem-sucedidos.

—BÍBLIA SAGRADA (Provérbios 16:3)

Resumo

A tecnologia biométrica é usada em uma ampla variedade de aplicações de segurança e principalmente para verificar a identidade de uma pessoa. No entanto, para esses sistemas a quantidade de informações sobre cada indivíduo é limitada a um pequeno conjunto de amostras, o que torna a verificação biométrica uma tarefa desafiadora. O objetivo deste estudo é analisar a aplicação de uma técnica de aumento de dados usando uma rede generativa adversarial profunda. O modelo proposto foi empiricamente testado no banco de dados de assinaturas GPDS300. Baseado nos resultados, o modelo é capaz de gerar assinaturas que podem ser utilizadas para aumentar os dados de treinamento disponíveis em um sistema de verificação de assinaturas.

Palavras-chave: aumento de dados, sistema offline de verificação de assinaturas, deep convolutional generative adversarial network, aprendizagem profunda, classificador dependente de escritor

Abstract

Biometrics technology is used in a wide variety of security systems and especially to verify the identity of a person. However, for these systems the amount of information about each person is limited to a small set of samples, which makes biometric verification a challenging task. This study analyzes the application of a data augmentation technique using a deep convolutional generative adversarial network. The model was empirically tested in the GPDS300 signet dataset. Based on the results, this model is capable of generating signatures that can be used to increase the data available in a signature verification system.

Keywords: data augmentation, offline signature verification system; deep convolutional generative adversarial network; deep learning, writer-dependent classifier

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Handwritten signature verification is a widely employed technique to identify people's identity in financial and administrative areas due to the non-invasive process of signature collection and the familiarity of users with this method [1]. During the past few decades, forensic document examiners have handled verification tasks and have been responsible for deciding if a signature is genuine or a forgery. However, with the advance of many machine learning models, mostly neural networks, this manual method has been replaced by automatic verification systems. In such systems, usually, a model is trained over a learning set of user signatures and then used for verification.

Actually, with the advances of deep learning methods, many neural network models have been employed as signature verification systems[2][3]. In such systems, the neural network receives a set of data and trains over them to classify or verify new unseen samples. Nowadays most research studies focus on learning feature representation demonstrating better results in multiple benchmarks such as CEDAR[4], MCYT-75[5], and GPDS Synthetic Signature[6].

One of the biggest challenges in the signature verification field is the limited number of samples per user. Usually, there are not sufficient data samples available for training the model, consequently restricting the performance of real applications. For instance, financial and administrative contracts often demand few signatures for the same user. As a result, the error rate of the verification task in automatic models is higher.

To address this issue, several researchers have proposed techniques to generate new images by applying transformations to images. This set of techniques are so-called Data augmentation techniques [7] [8]. Traditional data augmentation methods perform simple transformations of the original image, such as scaling, translation, rotation, flipping, lightning condition. In the handwritten signature field, Huang and Yan [9] researched some ways to "disturb" a genuine signature and generate new samples using "slight distortions" and "heavy distortions" like rotation, scaling, slant, etc. Other authors [10][11] have proposed a signature synthesis approach inspired by a neuromotor model to duplicate signatures. However such approaches fail to create a considerable number of high-skilled signatures and to bring any new visual features to improve the network learning ability.

In this work, we analyze the application of a data augmentation technique using a Conditional Deep Convolutional Generative Adversarial Network (CDCGAN)[12][13] and an Info Deep Convolutional Generative Adversarial Network (InfoDCGAN)[14][13] over a signature dataset. This model can generate new images from the genuine ones copying their distribution. Our goal is to increase the dataset size by creating high-quality skilled signatures, which can further be used to pre-train a given signature verification system to improve training process performance. Such networks have already been proved to be excellent methods for data

augmentation [13].

This undergraduate work is organized into five chapters. Initially, we introduce essential concepts to understand signature verification systems, generative adversarial networks , and data augmentation in Chapter 2. After that, we present the steps of how we built the generative adversarial network, such as preprocessing, model architecture, neural network training, and image analysis in Chapter 3. Following, we show the experiments over a signature dataset and then analyze the generated samples in Chapter 4. Finally, we present our conclusions and describe some future works in Chapter 5.

CHAPTER 2

Background and Related Works

This chapter presents some of the main concepts of handwritten signature verification systems, generative adversarial networks and convolutional neural networks and the data augmentation process, finally including related research from other authors.

2.1 Handwritten signature verification systems

Biometric systems are responsible for recognizing someone based on measurements of biological traits, for instance, fingerprint, face, and iris. When such systems are employed in handwritten signatures, they are widely known as Handwritten signature verification systems.

These systems are mainly used in two cases: verification and identification. A verification system aims to automatically discriminate if a given sample signature is indeed from one person, while identification systems are responsible for identifying who is the owner of the given sample signature [1].

Another important concept about signature verification systems refers to the acquisition method: online or offline. In online (dynamic) verification systems, signatures are captured in real time by some device, such as a digitizing tablet, which provides some dynamic information of the user's signing process, for instance, hand pressure, azimuth/altitude angle, stroke order, pen inclination, etc. By contrast, in the offline (static) verification systems, signatures are captured after the writing process in a digital form, and only uses 2D visual (pixel) images usually acquired by the scanning process.

Acquired signatures are classified into four classes: genuine, random forgeries, simple forgeries, and skilled forgeries. Genuine are real examples provided by some writer. Random forgeries are falsifications where the writer does not have any information about the genuine signature. Simple forgeries are falsifications where the writer knows only the person's name, but not his signature. Finally, skilled forgeries are falsifications where the writer knows the user's name and signature and usually copy it. Moreover, signatures from the same user typically display a high intra-class variability due to the user's signature variance over time and a low inter-class variability when we consider skilled forgeries (Figure 2.1).

Signature verification is essential in preventing the falsification of documents. According to Hafemann *et al.*[1] this problem is modeled as a verification task. Generally, the model is trained over a learning sample set containing genuine signatures from some writers. Afterward, this model is employed for verification: a user claims some identity and provides a query signature. The model then classifies the signature as genuine or forgery. Finally, the performance of the model is evaluated according to a test set.

Figure 2.1 Samples from the GPDS-960 dataset. Each row contains three genuine signatures from the same user and a skilled forgery.[13]

Such classification models are divided into two categories: Writer-Independent (WI) and Writer-Dependent (WD). WI systems are used to identify who is the owner of the signature. On the other hand, Writer-Dependent (WI) systems are employed to determine if a signature belongs to a specific user.

Actually, in the offline signature verification field, researchers have employed many techniques to identify a signature. They have put most of the effort into extracting handcrafted features to represent signatures, such as geometric features [15], directional features [16], and texture features [17]. Nevertheless, in recent years, with the development of deep learning models, handcrafted features have been replaced by hand-engineered feature extractors using raw data (pixels).

Haffemann *et al.*[18] proposed a Writer-Independent feature learning method, where a Convolutional Neural Network (CNN) is used to learn feature representations. After that, a writer-dependent classifier uses this representation in the training process. Zhang *et al.*[2] proposed using Generative Adversarial Networks (GAN) [19] for learning the features from a subset of users. In this case, they trained two networks: one to generate signatures and another one to discriminate if an image is from a real or an automatically generated signature [1], using the discriminator layers to extract features for future transfer learning.

2.2 Generative adversarial networks

Generative Adversarial Network or GAN is a framework proposed by Goodfellow *et al.* [19] for estimating generative models via an adversarial process, training two models simultaneously. This framework also can be interpreted as a minimax two-player game.

Two neural networks compose the GAN, a generator network G and a discriminator network D. The generator aims to generate new samples, and the discriminator seeks to discriminate between generated samples and real samples.

Explaining the generative process, a generative model G receives a sample noise z (normal or uniform distribution) input representing the latent features of the generated image. In practice, the generative model is a convolutional neural network, basically performing transposed

convolutions to upsample the input z. As a result, the model G generates new images from this input. On the other hand, the discriminator model D receives real images and generated images as inputs and discriminates them estimating the probability that a sample comes from a real or generated sample. As a result, the discriminator learns features which contribute to recognizing real images (Figure 2.2.).

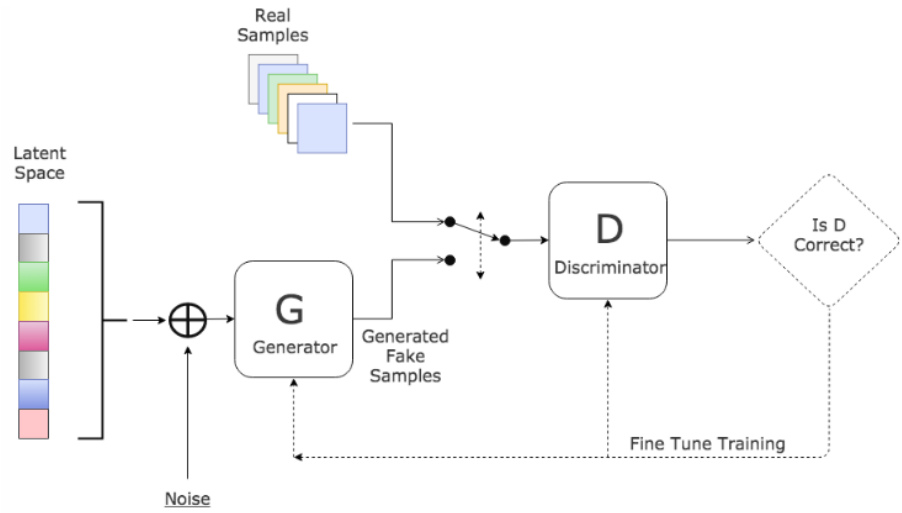


Figure 2.2 GAN architecture [20]

Mathematically, to learn the generator's distribution over data x, the author defined a prior on input noise variables pz(z), and then represent a mapping to data space as G(z; g), where G is a differentiable function represented by a multilayer perceptron with parameters g. The author also defines a second multilayer perceptron D(x; d) that outputs a single scalar. D(x) represents the probability that x came from the data rather than pg. He trains D to maximize the probability of assigning the correct label to both training examples and samples from G. After that, simultaneously he trains G to minimize log(1 D(G(z))). In other words, D and G play the following two-player minimax game with value function V (G, D).

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x))}\left[logD(x)\right] + \mathbb{E}_{z\sim p_z(z))}\left[log(1-D(G(z)))\right] \tag{2.1}$$

In other words, in the training process, the objective of G is to generate images with the highest value of D(x), the G generated samples are placed as inputs to D, which is trained as a deep network classifier and discriminates if the image is generated or real. D aims to maximize the probability of recognizing real images as real and generated images as fake. So, the target value of D is back propagated all the way back to G, training G to create images closer to the real image distribution according to the Algorithm 1.

Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used k = 1, the least expensive option, in our experiments. [19]

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $z^{(1)},...,z^{(m)}$ from noise prior $p_g(z)$

  • Sample minibatch of m examples $x^{(1)},...,x^{(m)}$ from data generating distribution $p_{data}(x)$

  • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta d} \frac{1}{m} \sum_{i=1}^{m} [log\, D(x^{(i)}) + log\, (1 - D(G(z^{(i)})))]$$

  end for

  • Sample minibatch of m noise samples $z^{(1)},...,z^{(m)}$ from noise prior $p_g(z)$

  • Update the generator by descending it stochastic gradient:

$$\nabla_{\theta g} \frac{1}{m} \sum_{i=1}^{m} [log\, (1 - D(G(z^{(i)})))]$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments

In GAN, the latent space z feds the generator network which does not have any additional information about the images to be generated. An extension of the GAN is the conditional generative adversarial network or CGAN [12]. In this model, both generator and discriminator receive some extra information y such as class labels or data from other modalities. In the generator, y acts as an extension of the latent space z, which improves the generator starting process. And in the discriminator, it helps to discriminate the images. With the introduction of this mechanism, the generated images are expected to follow the characteristics of the given additional information y. Finally, the objective function becomes

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x))} [logD(x|y)] + \mathbb{E}_{z \sim p_z(z))} [log(1 - D(G(z|y))))] \tag{2.2}$$

where D(x|y) and G(z|y) represent the inputs of the two models x and z given an input y.

However, GANs have an unstable training process, resulting in non-expected generated samples. For this reason, Redford *et al.* [13] proposed a set of constraints to improve the extraction of image representations, purely unsupervised, so-called Deep Convolutional Generative Adversarial Networks (DCGAN) Figure 2.3. This network architecture is an improvement of GANs [19], making them more stable to train. Thus, according to the author, the following constraints must be adopted:

a) Replace any pooling layers with strided convolutions[1] (discriminator) and fractional-strided convolutions[2] (generator).

[1] strided convolutions are operations that shrink the feature map size from one layer to another

[2] fractionally-strided convolution is a form of upsampling the feature map of a convolutional layer.

b) Use batchnorm[3] in both the generator and the discriminator.

c) Remove fully connected hidden layers for deeper architectures.

d) Use ReLU[4] activation in the generator for all layers except for the output, which uses Tanh.

e) Use LeakyReLU[5] activation in the discriminator for all layers.

An example model architecture is showing in the next figure.

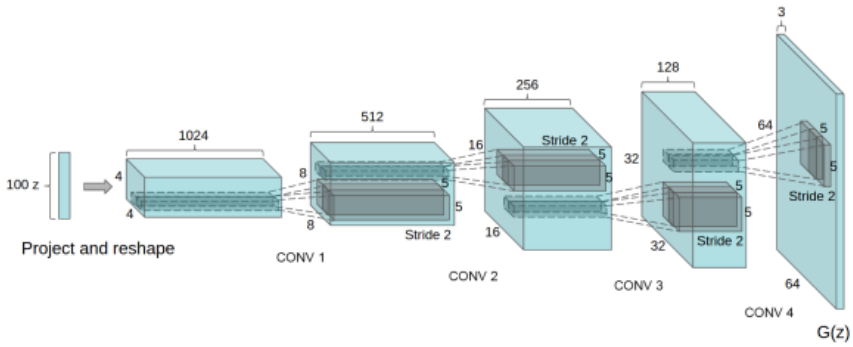


Figure 2.3 DCGAN generator used for LSUN scene modeling [13]

After the DCGAN, researchers developed other models of GANs for specific purposes, datasets, and samples. We briefly explain some of these different architectures in Section 2.4

2.3 Data augmentation

The objective of any machine learning model is to use the learned concepts and apply them to specific examples not seen by the model when it was learning. As a result, the model can generalize well from the training data to any data from the problem domain, allowing predictions for future data.

One of the main problems in such models and signature verification systems is the low number of samples for training the model.

To address this issue, one of the best ways to improve model performance, normally used by deep learning approaches, is to add more data to the training set, so-called data augmentation. Data augmentation has already proved to bring many benefits to convolutional neural networks (CNNs) [21], such as acting as a regularizer in preventing overfitting [6] in neural networks [22], and improving performance in imbalanced class problems [23].

[3]batch normalization normalizes the inputs to nonlinearities in every hidden layer

[4]rectified linear unit is an activation function defined as max(0,x) where x is the input

[5]leaky rectified linear unit is an activation function that allow a small, positive gradient when the unit is not active.

[6]Overfitting refers to a model that fits the training data too well, learning details and noise that negatively impacts the performance.

As an example, popular competition winning classifiers [7][24] adopted data augmentation techniques to increase the number of training samples and improve their performance.

Nowadays, some of the most popular approaches for data augmentation include:

- Flip - flipping images on horizontal or vertical axis;
- Rotate - rotating an image with a certain degree;
- Crop - cropping an image and resizing it;
- White noise - adding Gaussian noise;
- Color - random color manipulation;

Addressing this challenge, in the handwritten signature verification context the research community has proposed some data augmentation techniques. The different proposals are classified into two categories: generation of duplicated samples, and generation of new synthetic identities. In the first approach, samples are generated from existing ones, while the second one uses global characteristics from a signature database to create new samples with a unique identity.

Following we present some of the works in this field. Huang and Yan [25] proposed some techniques like rotation, scaling, slant, etc., to "disturb" a genuine signature and generate new samples using "slight distortions" to generate genuine signatures and "heavy distortions" to create forgeries. Ferrer *et al.* [11] proposed a signature synthesis approach cognitive-inspired on a neuromotor model divided into an action plan representing the trajectory on a spatial grid and the execution of the corresponding neuromuscular path applying a kinematic Kaiser filter. Ferrer *et al.* [10] also proposed a cognitive inspired algorithm to duplicate offline signatures using a set of nonlinear and linear transformations which simulated the human spatial cognitive map and motor system.

2.4 Other related works

In recent years, researchers have developed many models for generating new data in different applications. Ma *et al.* [26] developed a pose guided person image generator capable of creating images in any position given a target pose using a CGAN architecture. Another famous application is the cross-domain transfer. Researchers developed a CycleGAN[27], a model which transforms an image from one domain to another one given a style, for instance, creating a zebra from a horse. This model uses a generator network to generate new style images and another network in the reverse order to reconstruct the real images. Additionally, two discriminators are used, one to discriminate real samples, and another one to discriminate new style samples.

Increasing the image resolution is another critical area from GANs, Ledig *et al.* [28] developed a framework called super-resolution GAN or SRGAN capable of inferring photo-realistic natural images for 4x upscaling factors which is a GAN-based network optimized for a new perceptual loss. Another challenging problem refers to synthesizing images from text descriptions.

StackGAN[29] is a model capable of generating 256x256 pixels photo-realistic images conditioned on text descriptions decomposing the hard problem into more manageable sub-problems through a sketch-refinement process.

CHAPTER 3

Materials and Studied Methods

In this section, the experiment steps are presented, as well as details to reproduce the research. This section is divided into the signature corpus, tested models, preprocessing, model architectures, and image analysis.

3.1 Signature Corpus

The GPDS-300 is a publicly available offline signature dataset developed by the Grupo de Processado Digital de Señales [30]. It is composed by 16,200 offline signatures from 300 writers. Each writer contains 24 genuine signatures and 30 skilled forgeries obtained from 10 different forgers — lastly, signatures' size range from 153x258 pixels to 819x1137 pixels.

Figure 3.1 Dataset signatures. Each row contains signatures from one class in the dataset. The first two columns are genuine signatures, and the last two columns are forgeries (Author)

3.2 Tested models

In this work, we tested and modified some generative adversarial network architectures to generate synthetic signatures based on samples provided by the GPDS300 dataset. Firstly, we used an ordinary Generative Adversarial Network (GAN) [19] (Section 2.2). After that, we tested a more stable version of this network with some improvements in the network architecture for the training process, such as adding batch normalization and removing fully connected layers called Deep Convolutional Generative Adversarial Network (DCGAN) [13]. Next, we used the

Conditional Deep Convolutional Generative Adversarial Network (CDCGAN) [12], a modified version of the DCGAN which takes advantage of the label information from the dataset and uses it as input to improve the generator network. Finally, we employed an Info Deep Convolutional Generative Adversarial Network (InfoDCGAN)[14] which uses the same concept of the CDCGAN, but instead of using labels, it learns features from the discriminator network, besides adding a continuous code capable of varying the generated images.

Such models are designed by researchers to create synthetic images, for instance, faces, animals, and objects, however, actually, they are not used for signatures. After using the ordinary models, we proposed some modifications to improve the generated images. These modifications are further explained, and they are necessary due to the characteristics of signature images, such as size, variability, and color range.

To begin with, all models follow one common process with slight modifications. Firstly, the input images are preprocessed, resized and cropped to fit into the model which generates synthetic signatures. This model is divided into two Convolutional Neural Networks: the generator, which receives an input noise and outputs an image following the original image distribution. And, the discriminator, which gets the generated signature and outputs a probability between 0 and 1 of this signature to be genuine or a forgery. These two networks work in the same way described in the related works section. Finally, after training the system, the synthetic signatures with the highest scores are collected, and the others are discarded.

3.3 Preprocessing

The dataset already provides segmented signatures, so we will not address extraction in this research. However, the dataset contains images of different sizes, and convolutional neural networks require inputs with a fixed size. Thus, the neural network needs a preprocessing step. For this reason, we tested two preprocessing techniques.

In the first tested technique, we applied a modified version of the preprocessing method described in [18] removing some unnecessary steps and changing parameters. In this modified preprocessing method, we overlapped the signatures on a canvas of H x W size; which we choose according to the largest image in the dataset. Then, we binarized the images using Otsu's[1] algorithm to remove background noise according to a threshold and then find their center of mass. Besides, we resized and cropped the image to the desired final size in the network input (Figure 3.2). In the second technique, we employed simple resizing and cropping over the signatures to match the network input (Figure 3.3).

These two techniques mainly differ in the resulted number of channels of the preprocessed image and the image quality. While in the first method we obtain one channel (Binarized) without noise, in the second one, we obtain three channels (Red, Green, Blue) or RGB with possible noise.

[1]Otsu's algorithm perform clustering-based image thresholding automatically

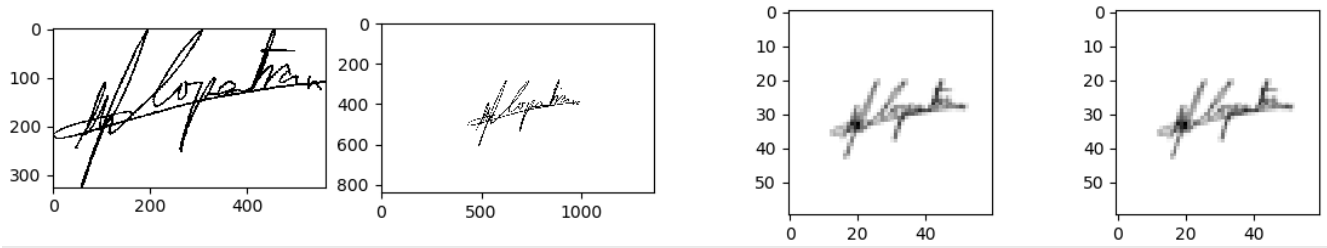


Figure 3.2 Technique 1. Preprocessed images with 64x64 pixels. (a) Original (b) Centered image in a pre-defined canvas size without noise (c) Resized (d) Cropped (Author)

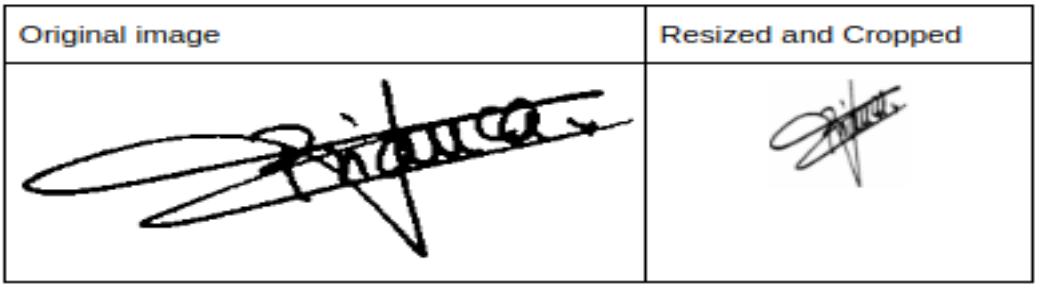


Figure 3.3 Technique 2. Original image and 64x64 pixels resized and cropped image (Author)

3.4 Model architectures

To achieve a suitable architecture model capable of generating images as close as possible to the original ones, we explored different architectural models. The first proposed model known as GAN has been successfully employed to create images of numbers in the handwritten digit MNIST dataset [19]. Nevertheless, this model did not produce high-quality signature images in our experiments. As a result, we decided to test a more stable network.

As proposed in [13], DCGAN introduces several improvements for training higher resolution and deeper networks. We tested this model as designed in the original paper with 64x64 pixel output images. Nevertheless, this size was too small to generate high-quality images for signatures. Thus, we modified the DCGAN architecture to receive and generate 128x128 pixel images adding an extra layer in both generator and discriminator. However, in both models, the generator network does not have any initial information to help it create synthetic samples, unless a uniformly sampled noise vector, impacting in the model performance. The model architecture for generating 64x64 signature images is available in the Appendix section.

To address this issue, we employed another model to improve the quality of the generated images. The CDCGAN [12] uses the information from real images by adding a label as a new parameter to the generator, and also in the discriminator to help it distinguish between real and fake images (Figure 4.4). In other words, the generator gets a hint about how to start the generative process, resulting in synthetic images inheriting the characteristics of the added extra labels. Since the results looked promising, we modified this network architecture

to receive and generate 256x256 pixels images. The model architecture for generating 256x256 pixels signature images is available in the Appendix section.

Finally, to improve the variability of the generated images, we tested the InfoDCGAN — a mix between an InfoGAN[14] and a DCGAN[13]. The main idea is to provide the generator network with latent code, which has meaningful and consistent effects on the output. Besides the training, this architecture differs from the CDCGAN in the generator input and the discriminator output layers. In CDCGAN, the network receives labels from the dataset, nevertheless in InfoDCGAN the generator network receives latent features extracted by the discriminator network. For instance, the generator input becomes the sum of z, and a latent code c composed of a discrete code (mapping the classes), and a continuous code (changeable from -2 to +2 in our case). And, the discriminator network outputs the prediction of one image being real and fake, besides the estimation of the latent code c (Figure 4.4).

3.5 Image analysis

To guarantee that the generated images are different, we used the pixel-by-pixel difference or in other words, the absolute difference between each pixel pair. Since the result of this difference becomes a black image with just a few white pixels, visually its hard to identify the difference between pixels. For this reason, to facilitate the resulted image visualization, we used the HSV (hue, saturation, value) image colormap[2]. It is important to notice that we tested two preprocessing techniques, see Section 3.3, however, this image analysis is only applied to the models with the best results which use the second preprocessed technique, resulting in RGB images. Additionally, we employed two metrics to calculate the similarity between the generated images and real images from the dataset. Firstly, the peak signal-to-noise ratio (PSNR) [31], an expression for the ratio between the maximum possible value (power) of a signal, and the power of distorting noise that affects the quality of its representation. Secondly, the Structural Similarity Index Measure (SSIM) [32], a method for measuring the fidelity between two images based on the computation of three terms, namely the luminance term, the contrast term, and the structural term. The PSNR is defined as:

$$PSNR = 20 \cdot \log_{10} \frac{MAX_x}{\sqrt{MSE}} \tag{3.1}$$

with MSE equals to:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [X(i,j) - Y(i,j)]^2 \tag{3.2}$$

where MAX_x is the maximum signal value that exists in our original image, x represents the matrix data of our original image, y represents the matrix data of our degraded image, m represents the numbers of rows of pixels of the images and i represents the index of that row, n represents the number of columns of pixels of the image and j represents the index of that column

[2]HSV is an alternative representations of the RGB color model, it is based upon how colors are organized and conceptualized in human vision in terms of other color-making attributes, such as hue, lightness, and chroma

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1) + (2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.3}$$

where $(\mu x, \sigma x)$ and $(\mu y, \sigma y)$ are the mean intensity and standard deviation set of image block x and image block y, respectively, while xy denote their cross-correlation. C1 and C2 are small constant values to avoid instability problem when the denominator is close to zero according to [32].

CHAPTER 4

Experiments and Discussion

Several experiments were performed using different models, dataset classes, parameters, and signature output sizes. In this section, we will explore the experimental protocols and results. This section is divided into the development environment, preprocessing experiments, , training the neural network, DCGAN experiments, CDCGAN experiments, InfoDCGAN experiments, and image analysis.

4.1 Development Environment

We developed the experiments using Python and the PyTorch framework [33], an open source deep learning platform for research prototyping. To run all the experiments, we used the Google Collaboratory [34], Google's free cloud service for AI developers. Collaboratory allows the use of Jupyter notebooks [1] running everything in a browser storing code in Google drive. This environment has the following hardware characteristics: Tesla K80 GPU, 2-core Intel Xeon CPU 2.30GHz, and 13GB RAM.

4.2 Preprocessing experiments

The preprocessing step is crucial to the stability of the network since the generator network learns to create synthetic images from the input's signature image, and the dataset images have different sizes. Thus, we performed some tests to resize the images to the network input size without cutting any part of the signature and keeping the aspect ratio between height and width. Using the first technique described in Section 3.3, the best results for 64 x 64 pixels images are illustrated in Figure 4.1 and Figure 4.2.

Analyzing the empirical results, we deduced that 64 x 64 pixels samples do not have enough pixels to guarantee the quality of the signature. Besides the size, the image normalization provides a uniform representation for all images in the dataset, which contributes to the generative adversarial network training. However this technique has one drawback, to centralize the image and remove the noise, we need to change the image to grayscale, losing its three channels property, so it becomes grayscale, *i.e.*, with only one channel. Consequently, the image loses many features related to the RGB aspect.

In the next step, we experimented 128x128 pixel input images due to easily adaptation to the

[1] The Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code.

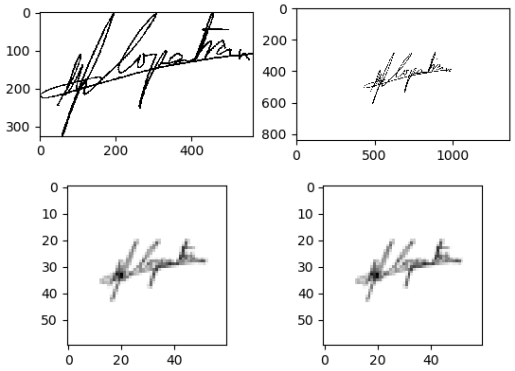


Figure 4.1 Preprocessed images with 64x64 pixels. From the top to the bottom, left to the right, original, centered, resized and cropped images (Author)

DCGAN original architecture (symmetric layers, *i.e.*, 4x4, 8x8, 16x16, 32x32, 64x64). Finally, we experimented rectangular 160x256 pixel input images. For real purposes, we realized that signatures are usually more extensive. This characteristic increases the necessity of a network which accepts rectangular inputs. In both cases, signatures had a higher resolution quality and better depicted the original ones (Figure 4.2).

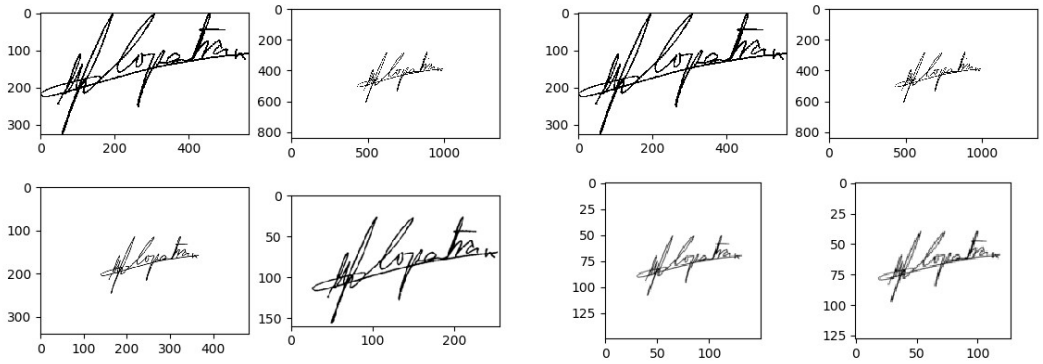


Figure 4.2 Preprocessed images with 160x256 pixels (first two columns) and 128x128 pixels (last two columns). In the first row, original and centered. In the second row, resized and croped.(Author)

Finally, to get to the results presented in the previous figures, three parameters were necessary: canvas size, resizing size and crop size. The final parameters are provided in Table 4.1 for all the experimented input sizes.

Regarding the second preprocessing technique described in Section 3.3, images were simply resized to the network input size as depicted in Figure 4.3.

DCGAN input	Canvas size	Resizing size	Crop size
64 x 64	840 x 1360	64 x 64	64 x 64
128 x 128	840 x 1360	150 x 150	128 x 128
160 x 256	840 x 1360	340 x 484	160 x 256

Table 4.1 Preprocessing final parameters in pixels

Figure 4.3 Images with 64x64 pixels, 128x128 pixels, 160x256 pixels, and 256x256 pixels (Author)

4.3 Training the neural networks

For training the DCGAN, we used most of the parameters from the original paper [13], since they already work for other purposes and then we changed them empirically to improve the network results for handwritten signature recognition. Initially, we trained the DCGAN in the GPDS300 dataset with all provided signature classes to simulate a writer-independent scenario, but after some tests, we changed the training process to a writer-dependent approach with only one class. We also trained the network with 20, 15, 10 and finally five real signature examples to define the minimum number of required signatures for the system. It is important to notice that this number of signatures was chosen empirically. We trained the model for 200 epochs. The weights were initialized with a normal distribution zero-centered and with 0.02 standard deviation. Optimization was performed in the model with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 1 due to the small number of samples, and an Adam optimizer with ß = 0.9, and 0.0001 learning rate in both generator and discriminator. In the discriminator, we set LeakyReLU slope to 0.2. Finally, we used a 100-dimensional normal distribution vector Z as the generator's input. Other parameters, such as kernel size, stride, padding, and bias for convolutional layers are available in the Appendix section. For training the CDCGAN, we used the same parameters from the DCGAN, except that we provided the number of classes. It is worth to remember that in the infoDCGAN, instead of classes, this number represents the discrete code. The others parameters for these two models can be found in the Appendix section.

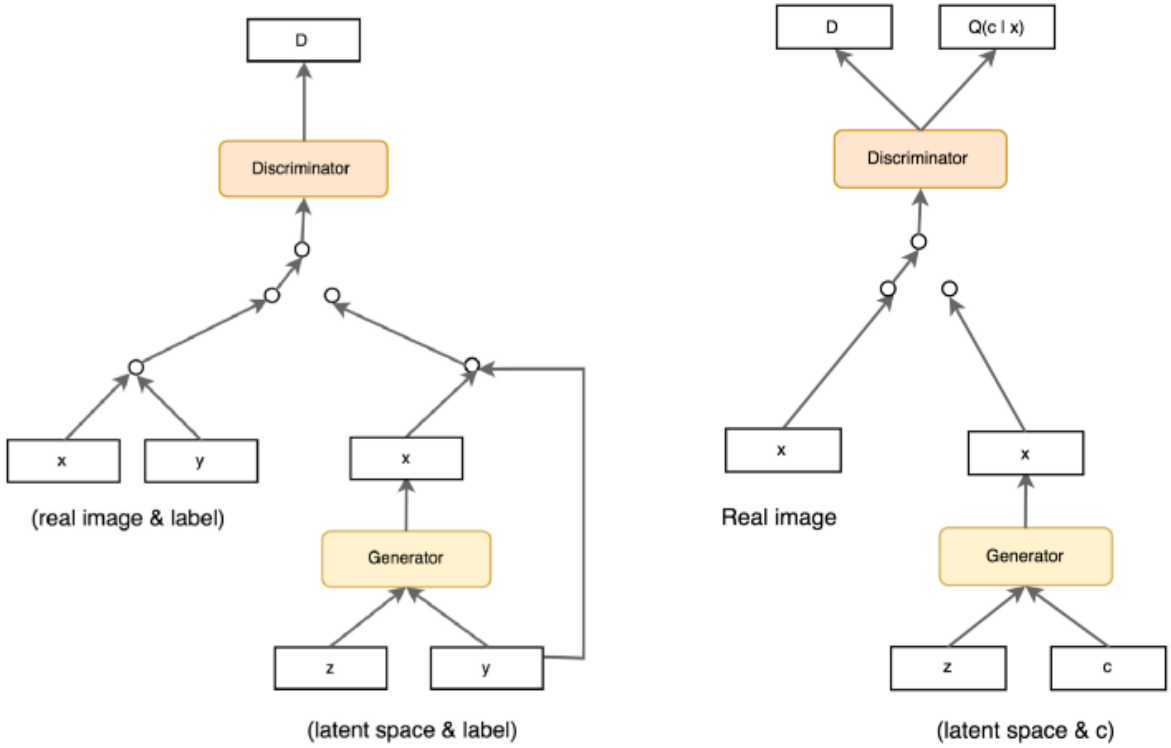


Figure 4.4 CDCGAN and infoDCGAN training schemas [35]

4.4 DCGAN

After the preprocessing step, we trained the DCGAN with 64x64 pixels input images and both preprocessing approaches to understanding if this model can create new synthetic signatures from the real samples. We divided this test into two parts: a writer-independent test and a writer-dependent test. In the first case, we performed an analysis with all 300 classes of the dataset. The objective was to understand if the DCGAN can generalize its results to any signature and create new signatures for any new sample in the network. In the second case, we performed a test with only one class of the dataset. The objective was to create new signatures for one writer. For testing purposes, we split the dataset into 90% of the images for training and 10% for testing, mainly due to the number of samples for each class, and to avoid underfitting[2]. Providing as many samples as possible. One class contains 24 real images and 30 fake images, so for training with all samples, we would have at least five to six testing samples. On the other hand, in the case of training only with real images, for instance, 20, two to three genuine signatures would be sufficient to test the model.

In the first test during the training process, the network had an unstable behavior, and the generator did not create images close to the genuine ones for all epochs (Figure 4.5).

Analyzing the training losses in the first plot of Figure 4.6 and Table 4.2, we observed

[2]Underfitting occurs if the model or algorithm shows low variance but high bias. Underfitting is often a result of an excessively simple model leading to poor predictions.

Figure 4.5 Generated sample after 60 epochs and real sample (Author)

that after the first epoch, the generator network kept a high training loss, and it was not able to decrease its loss and get generated images closer to the real ones. On the other hand, the discriminator had a low training loss which means that it learned to recognize features of a genuine signature and it was able to discriminate between generated samples and real samples resulting in a D training loss equals to zero, after the network stabilization. The second plot in Figure 4.6 confirms this hypothesis. D(x) is the probability of an image x being considered real, where x is a real image, while D(G(z)) is the probability of an image generated by G with input noise z being considered as real, where G(z) is a fake image. According to this plot, the discriminator gave a high probability for real images x, most of the time 1, while it gave low probabilities to generated images G(z), close to 0. Finally, these results show that the network did not achieve equilibrium and that the discriminator network was too powerful compared to the generator. As a result, generated images did not reach the expected outputs. We consider that some of the main problems of this architecture are: small input size, white pixels in the image, imbalance between generator and discriminator networks and the grayscale input images. Consequently, we empirically considered that the original DCGAN with this preprocessing step was not a good candidate as a data augmentation technique for handwritten signature verifications systems.

Epoch	Loss D	Loss G	D(x)	D(G(z))
0	0.7063	19.9575	0.6973	0.2923
15	0	37.4578	1	0
45	0	37.8084	1	0
53	0	23.1873	1	0
60	44.2783	47.5395	0	0

Table 4.2 DCGAN training statistics

After this attempt, we adjusted some parameters of the network to obtain a better performance. Initially, we changed the Adam optimizer learning rate. We increased the learning rate of the generator to twice the discriminator, but we did not succeed. Also, we increase the generator rate in ten times more than the discriminator, but the model did not converge. As a result, we investigated the influence of other parameters and the preprocessing technique in the results, and we discovered that the DCGAN model was not working correctly to images in

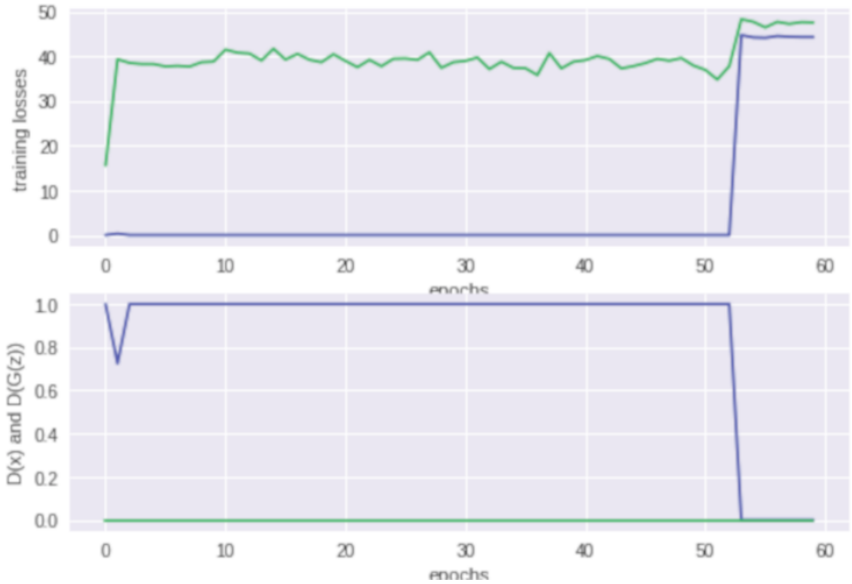


Figure 4.6 Generator training loss in green and discriminator training loss in blue. In the second plot D(x) in blue and D(G(z)) in green. (Author)

grayscale, but only in RGB format.

So, in the next experiments, for preprocessing, we employed scaling and resizing techniques over the signatures to match the 64x64 pixels input size of the DCGAN. Consequently, signatures heigh and width had a little distortion when fitting into the network.

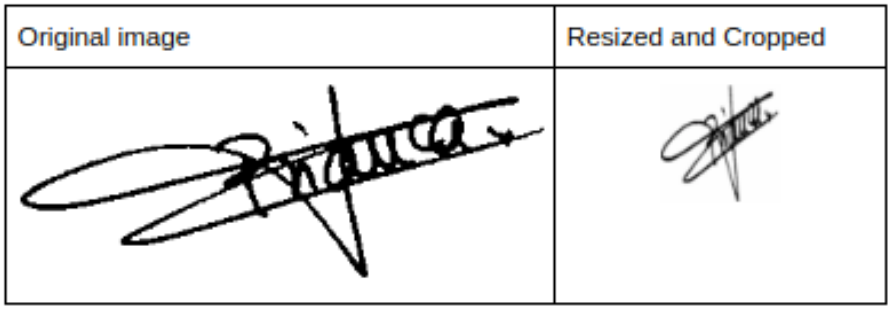


Figure 4.7 Original image and 64x64 Preprocessed image (Author)

In Figure 4.9 and Figure 4.10, it is possible to identify some of the generated images from this model. After training for 250 epochs, the network converged to one distribution which creates signatures close to the original ones. However, during the training process, the actual weights of the network could be already used to generated new data samples, for instance in the epochs 100 and 175, generated samples already looked like real samples. To see more examples of generated samples during the training process, refer to Figure A.5. We noticed that the generated samples have different shapes due to the intraclass variability. Signatures from the same user can differ even if they are real samples, an intrinsic characteristic of signatures.

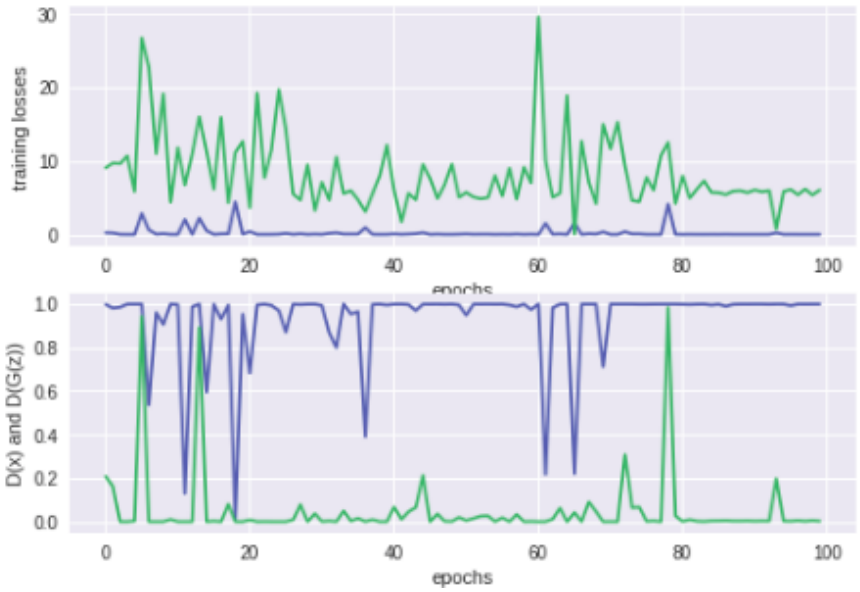


Figure 4.8 Generator training loss in green and discriminator training loss in blue. In the second plot D(x) in blue and D(G(z)) in green (Author)

Epoch	Real sample	Generated sample
100		
175		
238		

Figure 4.9 DCGAN 64x64 results per epoch (Author)

Figure 4.10 DCGAN 64X64 images for one signature. In the first row real images, in the second row generated images (Author)

Analyzing the plots from the training process in Figure 4.8, we realized that the training loss of the generator network decreased during the time, while the discriminator loss always kept low. Even with examples close to the generated ones, the generator network did not converge to a zero training loss. In our point of view, this behavior is explained by the intra-class variability and the background color of the images. Signatures of the same user are not equal, so each time the network updates its weights through backpropagation to fit one image well, it fails for other ones, which is considered an overfitting problem. On the other hand, generated images do not have a white background causing the discriminator to not give higher values of D(G(z)) and lower G training loss (Figure 4.8).

Finally, this model is suitable for data augmentation for handwritten verification systems, where the network input is a 64x64 input image.

After verifying the consistency of the model to 64x64 images, we tested it for bigger images, for instance, the sizes that we considered previously 128x128, and 160x256. For the 128x128 architecture, we added one more layer to the generator and discriminator architectures. The 160x256 architecture was a little more tricky because ordinary models usually consider only square images, however, to get the expected input and output size, we added one more layer with a rectangular kernel size in the first convolutional layer of the generator and a fully connected layer to the output of the discriminator. Nevertheless, for both architectures, within 200 epochs and several changes in the learning rate and the number of parameters, the models did not converge, and the generated samples were meaningless as we can see in Figure 4.11.

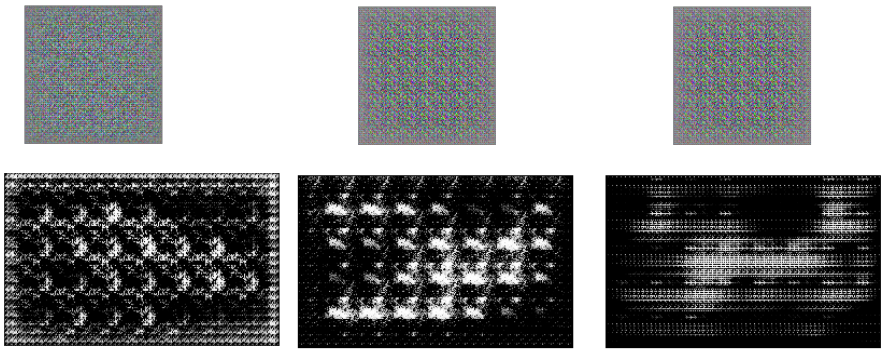


Figure 4.11 Modified DCGAN generated samples to 128x128 and 160x256 sizes in different epochs (Author)

4.5 CDCGAN

DCGANs demonstrated to be a promising technique for generating 64x64 pixels images. However, the model failed to scale for bigger images, for instance, 128x128 pixels. Since we designed our method to signature verification systems, in the training process, the signature's labels are assured to be true, because a contract or an official document provide them. So, an approach used to problems where labels are already known beforehand is the ConditionalGAN or in the case of our experiments a modified version called conditionalDCGAN. This model

uses the extra information provided by the images and encodes it into a 1-hot vector feeding it together with the noise z to the generator. By adding this additional parameter, the generator can start its generative process with more information than only a uniform distribution.

Firstly, we used this model to generate 64x64 pixels images, and as expected, the model created images compared as the previous DCGANs results. After such findings, we increased the network's input and output, by adding more layers to the generator and discriminator. We started with 128x128 pixels images with success, and finally, we tested the model with 256x256 pixels images with the architecture described in Table A.3 and Table A.4. In this experiment, for preprocessing, we employed scaling and resizing techniques over the signatures to match the 256x256 input size of the CDCGAN. Consequently, signatures heigh and width had a little distortion when fitting into the network. However, images with this size demonstrated to have a better image-quality than 64x64 ones.

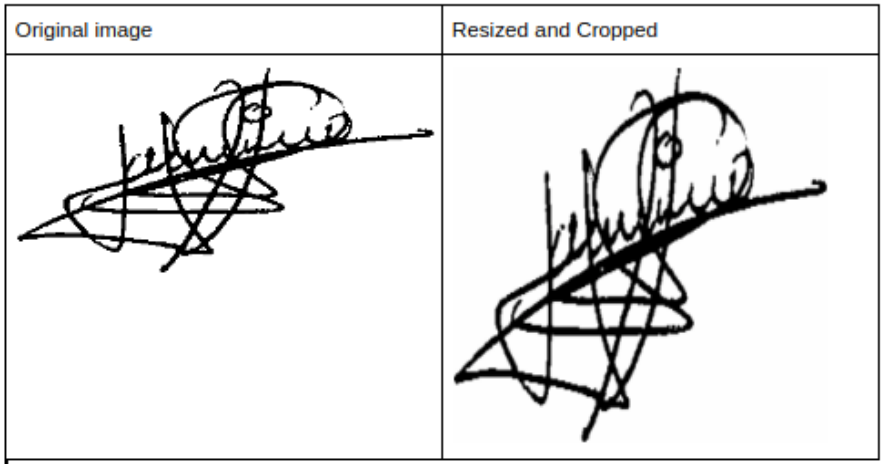


Figure 4.12 Original image and 256x256 pixels Preprocessed image (Author)

Analyzing the plot from the loss per iteration[3] in the training process for one class with 256x256 pixels images Figure 4.13, we realized that both training losses decreased during the time converging to a zero loss. In our point of view, this demonstrates the equilibrium of the proposed method — these metrics were also reflected in the results. In the epoch 120, we already could see a clear definition of the image and use it as an augmented data. The problem of gray background demonstrated in the DCGAN approach also decreased significantly, and the model was able to generate images from a dataset with only five real images. To see more examples of generated samples during the training process, refer to Table A.6.

In Figure 4.14 we can see the results of the CDCGAN generative process in different epochs for different classes of signatures. As a matter of comparison, original rescaled images were added to show the resemblance with the generated samples. We realized that the model was able to create images, despite the signature's shape or trace. Nevertheless, according to the results, images did not vary significantly in the different epochs, an aspect that could be further

[3]The number of iterations in one epoch is equals to the number of images. As a consequence, for our system with five images, we can compare 1000 iterations with 200 epochs.

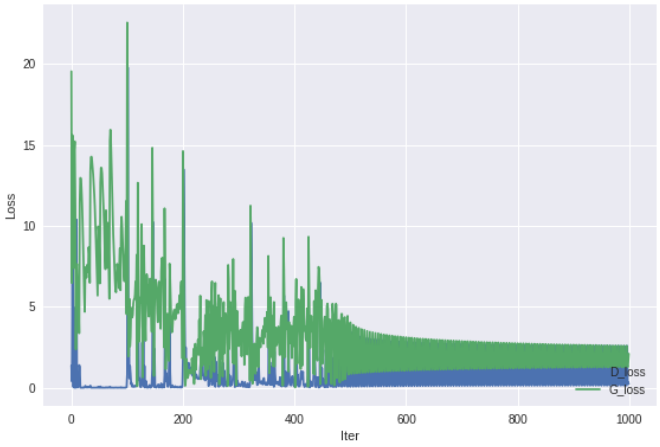


Figure 4.13 CDCGAN loss per iteration with 256X256 pixels images(Author)

improved. Addressing the number of generated images, we considered empirically that every image after the epoch 150 was able to be used as an augmented data. For this reason, we believe that for a model with five genuine signatures as input, this model can generate 50 synthetic ones. Finally, in our point of view, this model demonstrated to be a suitable approach for data augmentation for handwritten verification systems in a scenario with bigger images, such as 256x256 pixels as depicted in Figure 4.14.

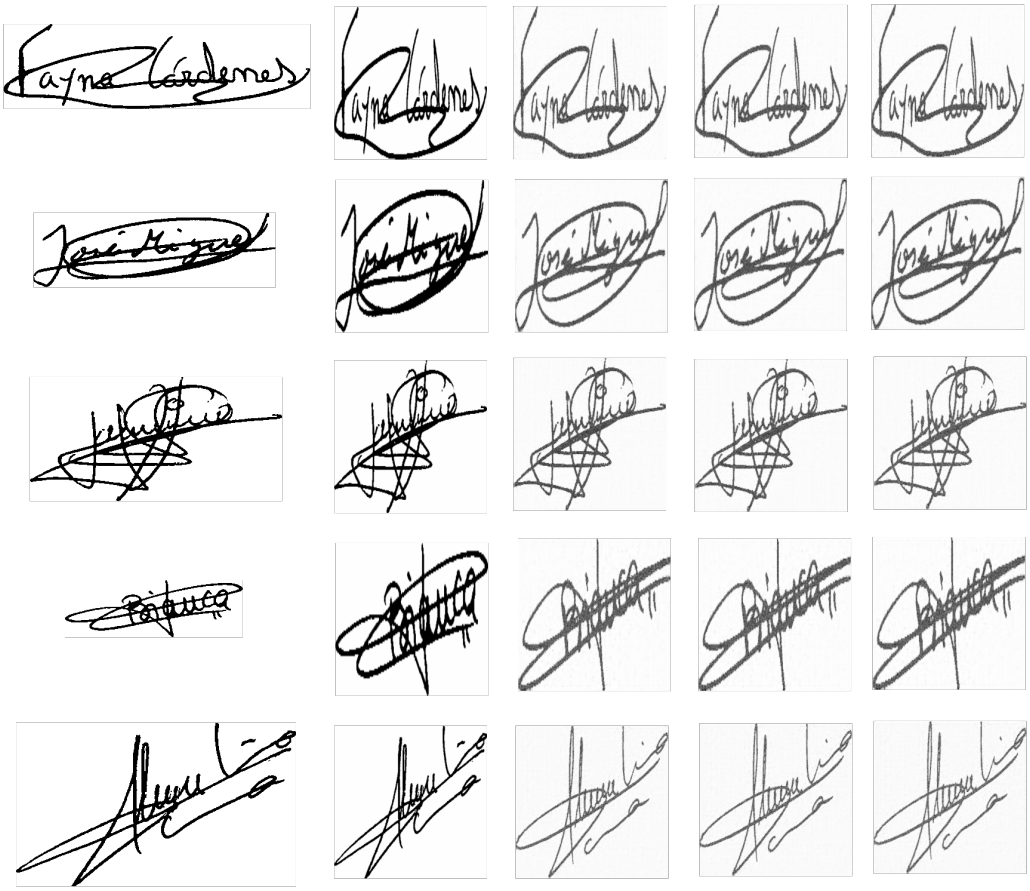


Figure 4.14 256X256 generated images. From the left to the right, original, 256x256 pixels resized, and three generated samples (Author)

4.6 InfoDCGAN

Finally, to improve the variability of our model, we tested the InfoDCGAN [14], an information-theoretic extension to the GAN able to learn disentangled representations. We trained the InfoDCGAN with the same parameters described in Section 4.3, except for the length of the continuous code. In our case, we changed this value to one, because we were working with one class.

In the first experiment, we tested our model with five genuine signatures as inputs in 200 epochs. We realized that this number of epochs was not enough to make the model converge. For this reason, we doubled the number of epochs to achieve convergence. Even so, the model did not converge. Consequently, we adjusted the InfoDCGAN model structure to resemble the CDCGAN one, since this last model showed promising results. We removed the latent feature output from the discriminator and its loss update from the training process. Besides that, we added labels in both generator and discriminator as in the CDCGAN Figure 4.15.

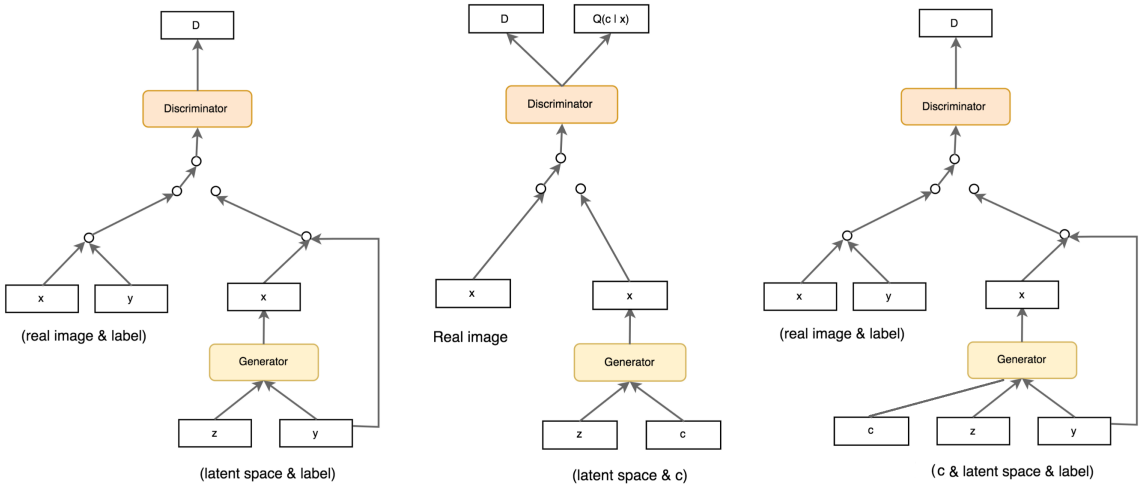


Figure 4.15 (a) Original CDCGAN (b) InfoDCGAN (c) adjusted InfoDCGAN [35]

Additionally, we used 400 epochs to analyze the generated results longer. We demonstrate the results from this experiment in Figure 4.16. According to the generated images depicted in the figure, examples did not vary significantly with the modified InfoDCGAN implementation even though we introduced the continuous code, a variational latent code, which should help in this variance.

Figure 4.16 Generated images from different epochs from one class using five signatures as input (Author)

As a result, we investigated the continuous code contribution in the generated images. We multiplied this code by many constants (0.01, 0.1, 10, 100, 1000) to understand what was its impact. However, analyzing the results, we realized that the continuous codes were not directly influencing the signature variation through the epochs as we expected.

Following that, we investigated the data influence over the signature's variation. Our central hypothesis was that the five genuine signatures were too identical and in an insufficient number to create a considerable variety in the generated examples. Thus, we added more samples to the data input. Firstly, we tested our model with 20 genuine signatures — additionally, we also checked it with 20 genuine signatures and 25 forgeries. Figure 4.17

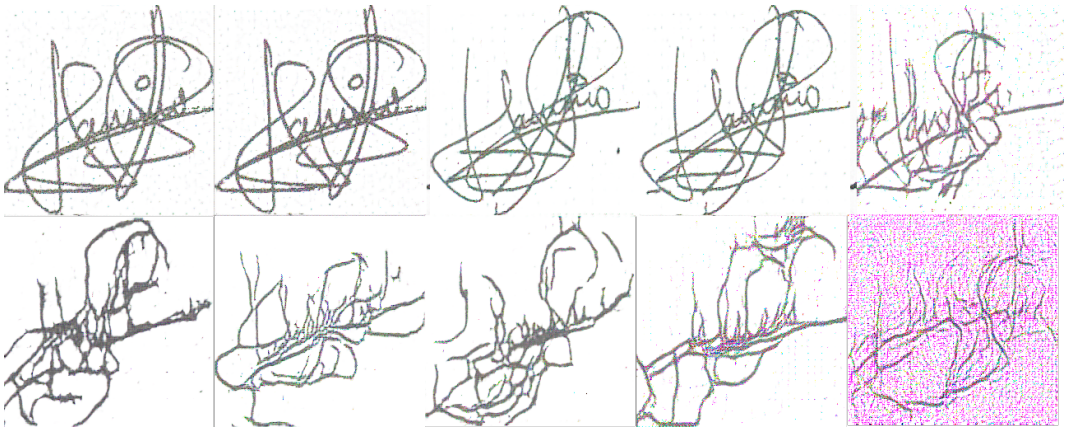


Figure 4.17 First row, examples generated with 20 genuine images as input in epochs 90, 120, 220, 250, and 400. Second row, examples generated with 20 genuine images and 25 forgeries in epochs 80, 90, 120, 150, and 400 (Author)

Finally, we experienced that the tests with five signatures did not have a significant variation. On the other hand, in tests with more samples, such as 20 genuine signatures, the model generated more than one variety of examples. However, for the model with 20 genuine signatures and 26 forgeries, there was not a convergence, probably due to a considerable difference between the structures of genuine signatures and copies. It is important to notice that this would not be the real situation since the application receives only genuine signatures, and the results behaved as we expected. Consequently, we deduced that the number of data input samples influences directly in the model variance. Five signatures might not be enough to achieve the expected degree of variation to the generated images, and increasing the number of signatures tends to increase variability. However, adding samples with a different structure can result in a

lack of convergence.

4.7 Image Analysis

Understanding the quality of the generated images is an essential step in the data augmentation process. For this reason, we analyzed and compared the generated images with themselves and with the real ones using some metrics. Firstly, since there was not a considerable variability between samples, we performed a test to check that generated images were different among themselves. Thus, we calculated the pixel-by-pixel difference from six images separated in three different classes (Figure 4.19), and applied the HSV colormap according to the color scheme in Figure 4.18, provided by [36] .

Figure 4.18 HSV image colormap

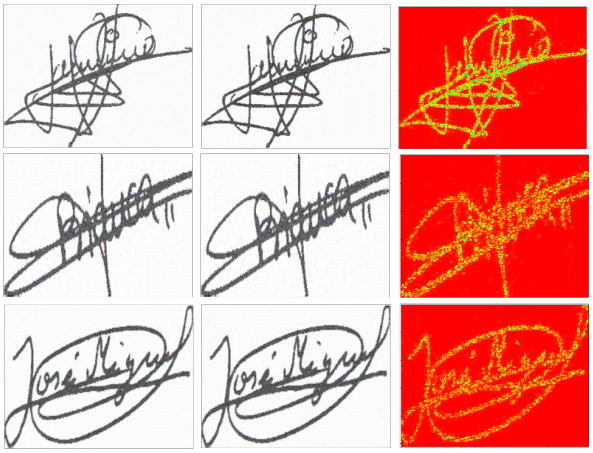


Figure 4.19 Pixel-by-pixel difference between two images for three classes in epochs 180 and 200 with HSV image colormap (Author)

The figure is the subtraction of color values from every pixel in both images. So, according to the picture, we noticed that there were some differences between pixels from generated images in different epochs. The pixel difference in the third column where red means no difference and green means some variation according to the color scheme, demonstrates that pixels related to the edges of the signatures have some differences, meaning that our method is changing features related to the signatures format and showing that the images through different epochs are indeed different.

After analyzing this difference, we calculated the peak signal-to-noise ratio (PSNR) (3.1) between generated, real and fake images. This ratio is usually used as a quality measurement between an original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image. There is an inverse relationship between PSNR and MSE.

So, a higher PSNR value indicates the higher quality of the image. The MSE metric measures the average squared difference between the estimated image and the estimation, corresponding to the expected value of the squared error loss.

Despite the PSNR values, large distances between pixel intensities do not necessarily mean the contents of the images are different [31]. Thus, to get another measure of the real difference between images, we employed another metric, the Structural Similarity Index Measure (SSIM) (3.3). The SSIM is a perception-based model that considers image degradation as perceived change in structural information, such as directional pixel intensity. In other words, it recognizes the difference in the structural information of an image. The SSIM values vary between -1 and 1, where the 1 indicates perfect similarity, while 0 shows the opposite, finally -1 is only achieved theoretically.

To serve as a benchmark, we calculated the PSNR and SSIM, first comparing 300 real images with other 300 real images from the same classes, and then comparing 300 real images with 300 forgeries from the same classes (Table 4.3). From this table, we analyzed that despite being from the same class, one real signature differs from another real signature from the same user. Furthermore, when comparing real images with forgeries, the PSNR and the SSIM decrease, since the images are less similar.

Metric	Real/Real	Real/Forgery
PSNR	7.35 ± 1.27	7.02 ± 1.15
SSIM	0.52 ± 0.10	0.50 ± 0.09

Table 4.3 PSNR and SSIM benchmark comparing real images with other real images and forgeries from the same classes

Using the described metrics, experiments were performed with fifty generated images from five classes, each class containing ten generated images from epochs 155, 160, 165, 170, 175, 180, 185, 190, 195, 200. Following, we compared the generated images with real ones and forgeries from the same classes. The results are depicted in Table 4.4. Comparing these values with the values from Table 4.3, we realized that the value of PSNR between generated images and real ones (10.53 +- 3.04) is in the same range compared to real images and other real images from the same classes (7.35 +- 1.27) considering the mean and standard deviation. Comparing to forgeries, the results showed that the value of PSNR in the generated images (8.96 +- 0.40) is higher than the value from real ones (7.02 +- 1.15), meaning that generated images have less influence of noise than real ones.

Furthermore, the SSIM measure which recognizes local variations and the image structure was employed. Comparing the values from generated images with real ones (0.56 +- 0.17), and real ones with other real ones from the same classes (0.52 +- 0.10), the SSIM value was considered equal, according to the mean and standard deviation demonstrating that the structure of generated images is as good as the structure of real images from the dataset. Additionally, the value of the comparison between generated images and forgeries (0.46 +- 0.03) is in the same range as the value from real ones compared to forgeries (0.50 +- 0.09).

Finally, according to the results, we consider that the image quality of the synthetic samples is acceptable when compared to the range of images from the dataset.

Metric	Generated/Real	Generated/Forgery
PSNR	10.53 ± 3.04	8.96 ± 0.40
SSIM	0.56 ± 0.17	0.46 ± 0.03

Table 4.4 PSNR and SSIM comparing generated images with real ones and forgeries from the same classes

CHAPTER 5

Conclusions

Handwritten signature verification systems are used in a wide variety of security systems to verify the identity of a person. One of the biggest challenges in this field is the limited number of samples per user. Generally, the amount of information about each person is limited to three or four signatures presented in one official document, which makes the biometric verification a challenging task and restricting the performance of real applications. In this work, we proposed a data augmentation technique using the CDCGAN architecture, and an adjusted InfoDCGAN architecture, modified versions of a DCGAN to increase the number of signatures for such systems. In our experiments, we considered that this model is capable of generating high-quality synthetic signatures to be used as extra data to handwritten signature verification systems, creating ten times more signatures than the input, and finally achieving the proposed objective. We consider that one of the main advantages of our method is the automatic generation of new samples with reasonable structural information, and meaningly features collaborating with the training of verification systems. As research, this work mainly contributes to open a different view to the research community about the application of deep learning methods in the creation of synthetic samples, and the enhancement of handwritten signature verification systems.

5.1 Limitations

We developed a technique for data augmentation restricted to writer-dependent verification systems. Scaling this model to a writer-independent approach would demand further improvements without guarantees. Furthermore, our neural network is limited to 256x256 pixels images.

5.2 Future work

We believe that there are several points to improve. Firstly, working on enhancements to the variability of the generated images, then create a network capable of receiving and producing images of different sizes, furthermore; testing other generative models are important investigations to be made. Additionally, it is essential to examine the proposed data augmentation technique in already-known state-of-the-art algorithms in the signature verification field to understand what is the impact in their final results.

APPENDIX A

appendix

Layer	Size	Parameters
Input	100x1x1	input z = 100x1x1
Transposed Convolution	512x4x4	kernel size=(4, 4) stride=(1, 1) bias=False
Batch normalization	512x4x4	eps=1e-05 momentum=0.1
ReLU	512x4x4	
Transposed Convolution	256x8x8	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	256x8x8	eps=1e-05 momentum=0.1
ReLU	256x8x8	
Transposed Convolution	128x16x16	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	128x16x16	eps=1e-05 momentum=0.1
ReLU	128x16x16	
Transposed Convolution	64x32x32	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	64x32x32	eps=1e-05 momentum=0.1
ReLU	64x32x32	
Transposed Convolution	3x64x64	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Tanh	3x64x64	

Table A.1 DCGAN 64x64 Generator architecture

Layer	Size	Parameters
Input	3x64x64	
Convolution	64x32x32	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	64x32x32	eps=1e-05 momentum=0.1
LeakyReLU	64x32x32	negative slope=0.2
Convolution	128x16x16	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	128x16x16	eps=1e-05 momentum=0.1
LeakyReLU	128x16x16	negative slope=0.2
Convolution	256x8x8	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	256x8x8	eps=1e-05 momentum=0.1
LeakyReLU	256x8x8	negative slope=0.2
Convolution	512x4x4	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	512x4x4	eps=1e-05 momentum=0.1
LeakyReLU	512x4x4	negative slope=0.2
Convolution	1x1x1	kernel size=(4, 4) stride=(1, 1) bias=False
Sigmoid	1x1x1	

Table A.2 DCGAN 64x64 Discriminator architecture

Layer	Size	Parameters
Input	101x1x1	input z = 100x1x1 , number of classes = 1x1x1
Transposed Convolution	2048x4x4	kernel size=(4, 4) stride=(1, 1) bias=False
Batch normalization	2048x4x4	eps=1e-05 momentum=0.1
ReLU	2048x4x4	
Transposed Convolution	1024x8x8	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	1024x8x8	eps=1e-05 momentum=0.1
ReLU	1024x8x8	
Transposed Convolution	512x16x16	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	512x16x16	eps=1e-05 momentum=0.1
ReLU	512x16x16	
Transposed Convolution	256x32x32	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	256x32x32	eps=1e-05 momentum=0.1
ReLU	256x32x32	
Transposed Convolution	128x64x64	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	128x64x64	eps=1e-05 momentum=0.1
ReLU	128x64x64	
Transposed Convolution	64x128x128	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	64x128x128	eps=1e-05 momentum=0.1
ReLU	64x128x128	
Transposed Convolution	3x256x256	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Tanh	3x256x256	

Table A.3 CDCGAN 256x256 Generator architecture

Layer	Size	Parameters
Input	3x256x256	
Convolution	64x128x128	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	64x128x128	eps=1e-05 momentum=0.1
LeakyReLU	64x128x128	negative_slope=0.2
Convolution	128x64x64	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	128x64x64	eps=1e-05 momentum=0.1
LeakyReLU	128x64x64	negative_slope=0.2
Convolution	256x32x32	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	256x32x32	eps=1e-05 momentum=0.1
LeakyReLU	256x32x32	negative_slope=0.2
Convolution	512x16x16	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	512x16x16	eps=1e-05 momentum=0.1
LeakyReLU	512x16x16	negative_slope=0.2
Convolution	1024x8x8	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	1024x8x8	eps=1e-05 momentum=0.1
LeakyReLU	1024x8x8	negative_slope=0.2
Convolution	2048x4x4	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Batch normalization	2048x4x4	eps=1e-05 momentum=0.1
LeakyReLU	2048x4x4	negative_slope=0.2
Convolution	1x1x1	kernel size=(4, 4) stride=(2, 2) padding=(1, 1) bias=False
Sigmoid	1x1x1	

Table A.4 CDCGAN 256x256 Discriminator architecture

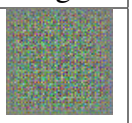
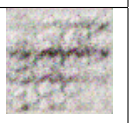
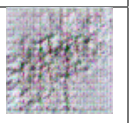
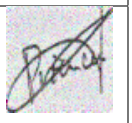
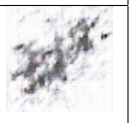
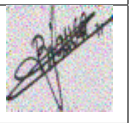
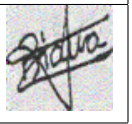
Epoch	Loss D	Loss G	D(x)	D(G(z))	Image
0	1.0591	4.6433	3.799	872	
25	100	5.1842	9.990	89	
50	529	7.5555	9.492	7	
75	73	5.0984	9.991	64	
100	53	8.3251	9.999	52	
150	1.329	11.1238	8.767	8.012	
175	71	5.0125	1.2000	70	
200	51	7.0938	9.996	47	
225	52	5.4688	9.992	44	
250	337	3.8009	9.999	330	

Table A.5 DCGAN 64x64 training process

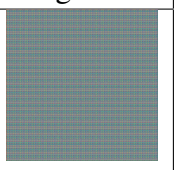
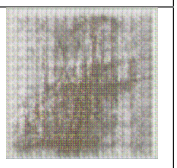
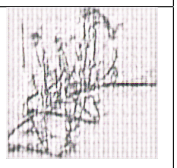
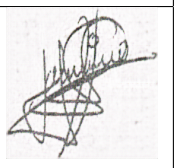
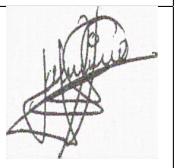
Epoch	Loss D	Loss G	Image
0	1.36	19.53	
25	4.90	8.75	
50	1.42	4.25	
75	0.37	2.83	
100	0.50	0.81	
125	0.10	3.0	
150	0.11	2.80	
175	0.12	2.67	
200	0.22	02.09	

Table A.6 CDCGAN 256x256 training process

Bibliography

[1] L. G. Hafemann, R. Sabourin, and L. S. Oliveira. Offline handwritten signature verification — literature review. In *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–8, Nov 2017.

[2] Z. Zhang, X. Liu, and Y. Cui. Multi-phase offline signature verification system using deep convolutional generative adversarial networks. In *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*, volume 2, pages 103–107, Dec 2016.

[3] L. G. Hafemann, R. Sabourin, and L. S. Oliveira. Analyzing features learned for offline signature verification using deep cnns. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2989–2994, Dec 2016.

[4] Meenakshi K. Kalera, Sargur N. Srihari, and Aihua Xu. Offline signature verification and identification using distance statistics. *IJPRAI*, 18:1339–1360, 2004.

[5] J. Fierrez-Aguilar, N. Alonso-Hermira, G. Moreno-Marquez, and J. Ortega-Garcia. An off-line signature verification system based on fusion of local and global information. In Davide Maltoni and Anil K. Jain, editors, *Biometric Authentication*, pages 295–306, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[6] M. A. Ferrer, M. Diaz-Cabrera, and A. Morales. Synthetic off-line signature image generation. In *2013 International Conference on Biometrics (ICB)*, pages 1–7, June 2013.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[8] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[9] D. R. Kisku, A. Rattani, P. Gupta, and J. K. Sing. Offline signature verification using geometric and orientation features with multiple experts fusion. In *2011 3rd International Conference on Electronics Computer Technology*, volume 5, pages 269–272, April 2011.

[10] M. A. Ferrer, M. Diaz-Cabrera, and A. Morales. Synthetic off-line signature image generation. In *2013 International Conference on Biometrics (ICB)*, pages 1–7, June 2013.

[11] M. A. Ferrer, M. Diaz-Cabrera, and A. Morales. Static signature synthesis: A neuromotor inspired approach for biometrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):667–680, March 2015.

[12] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *ArXiv e-prints*, November 2014.

[13] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ArXiv e-prints*, November 2015.

[14] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *ArXiv e-prints*, page arXiv:1606.03657, June 2016.

[15] Kai Huang and Hong Yan. Off-line signature verification based on geometric feature extraction and neural network classification. *Pattern Recognition*, 30(1):9 – 17, 1997.

[16] R. Sabourin and J. . Drouhard. Off-line signature verification using directional pdf and neural networks. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 321–325, Aug 1992.

[17] Mustafa Berkay Yılmaz and Berrin Yanıkoğlu. Score level fusion of classifiers in off-line signature verification. *Information Fusion*, 32:109 – 119, 2016. SI Information Fusion in Biometrics.

[18] L. G. Hafemann, R. Sabourin, and L. S. Oliveira. Writer-independent feature learning for offline signature verification using deep convolutional neural networks. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2576–2583, July 2016.

[19] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[20] Al Gharakhanian. Gans: One of the hottest topics in machine learning, 12 2016. [Online; accessed 1-November-2018] URL: `https://www.linkedin.com/pulse/gans-one-hottest-topics-machine-learning-al-gharakhanian/?trk=pulse_spock-articles`.

[21] Yann Lecun, Bernhard Boser, John Denker, Don Henderson, R E. Howard, Wayne E. Hubbard, and Larry Jackel. Handwritten digit recognition with a back-propagation network. *Neural Information Processing Systems*, 2:396–404, 01 1989.

[22] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, Aug 2003.

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *ArXiv e-prints*, June 2011.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[25] Kai Huang and Hong Yan. Off-line signature verification based on geometric feature extraction and neural network classification. *Pattern Recognition*, 30(1):9 – 17, 1997.

[26] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. In *NIPS*, 2017.

[27] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *ArXiv e-prints*, March 2017.

[28] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *ArXiv e-prints*, September 2016.

[29] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017.

[30] F. Vargas, M. Ferrer, C. Travieso, and J. Alonso. Off-line handwritten signature gpds-960 corpus. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 764–768, Sept 2007.

[31] Z. Wang and A. C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1):98–117, Jan 2009.

[32] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[33] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[34] Google. Google colaboratory. [Online; accessed 1-November-2018] URL: `https://colab.research.google.com/`.

[35] Jonathan Hui. Gan cgan infogan (using labels to improve gan), 11 2018. [Online; accessed 15-November-2018] URL: `https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d`.

[36] matplotlib. color example code: colormaps reference, 1 2012. [Online; accessed 14-December-2018] URL: `https://matplotlib.org/examples/color/colormaps_reference.html`.

[37] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[38] M. Diaz, M. A. Ferrer, G. S. Eskander, and R. Sabourin. Generation of duplicated off-line signature images for verification systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(5):951–964, May 2017.

[39] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. In *NIPS*, 2017.

This volume has been typeset in LaTeXwith the UFPEThesis class (www.cin.ufpe.br/~paguso/ufpethesis).