FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Data Augmentation Using Generative Adversarial Networks
Student:	Iveta Šárfyová
Supervisor:	Ing. Magda Friedjungová
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

One of the possible ways how to generate artificial training data is to use a generative adversarial network (GAN) where one network is trained to generate real-looking training data (the generator), and the other is trained to distinguish artificial-looking data (the discriminator). Both try to beat the other and at the end of the training process, the generator network can be used for the purpose of new training datasets.

Survey common and state of the art algorithms for data augmentation. Implement at least two surveyed algorithms using GAN and compare their performance to different approaches used in the data augmentation domain (e.g. variational autoencoders and common techniques such as rotation, shift, etc.) on publicly accessible image datasets. Comparison will be done using classification models learned on both, original and synthetic data, and their combination. Examine particular errors and describe the limitations of the algorithms used.

References

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017.
Andrew Brock, Jeff Donahue, Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. ICLR 2019
Ian J. Goodfellow, Jean Pouget-Abadie†, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. NIPS 2014.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague November 26, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Data Augmentation Using Generative Adversarial Networks

Iveta Šárfyová

Department of Applied Mathematics
Supervisor: Ing. Magda Friedjungová

May 13, 2020

Acknowledgements

Firstly, I would like to express my profound gratitude to the supervisor of this thesis, Ing. Magda Friedjungová, for her exceptional guidance and insightful discussions. I could not wish for a better mentor for my bachelor thesis.

My sincere thanks also go to Tomáš Halama for his support, encouragement and daily supplies of chocolate. I am grateful for your patience and for bringing a smile to my face.

Last but not least, I would like to thank my family and friends for supporting me during my studies. I would never have been able to achieve this without you.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 13, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Šárfyová, Iveta. *Data Augmentation Using Generative Adversarial Networks*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstract

Most labelled real-world data is not uniformly distributed within classes, which can have a severe impact on the prediction quality of classification models. A general approach is to overcome this issue by modifying the original data to restore the balance of the classes. This thesis deals with balancing image datasets by data augmentation using generative adversarial neural networks. The primary focus is on generating images of underrepresented classes in imbalanced datasets, which is a process known as class balancing. The aim of this thesis is to analyse and compare data augmentation techniques including standard methods, generative adversarial networks and autoencoders. Evaluation is done using classifiers trained on the original, unbalanced and augmented datasets. The results achieved suggest how the performance of the methods proportionately deteriorates with increasing imbalance rate and diversity of datasets.

Keywords data augmentation, imbalanced dataset, data preprocessing, class balancing, neural network, generative adversarial network, autoencoder

Abstrakt

Většina dat z reálného světa není rovnoměrně rozdělena do odpovídajících tříd, ale je nevyvážená, což může mít velký vliv na kvalitu predikce klasifikačních modelů. Obecný přístup k řešení tohoto problému je modifikace původních datových sad tak, abychom dosáhli vyváženosti jednotlivých tříd. Tato práce se zaobírá vyvážením obrazových dat za pomoci generativních adversariálních sítí. Primární důraz je kladen na generování obrazových dat náležících do tříd s nedostatečným počtem reprezentantů, což je proces známý jako class balancing. Práce se zabývá analýzou a porovnáním různých technik používaných pro rozšíření dat, jako jsou geometrické metody nebo modely založené na principu neuronových sítí. Vyhodnocení je provedeno pomocí klasifikačních modelů, natrénovaných na původních, nevyvážených i uměle vyvážených datových sadách. Dosažené výsledky naznačují, jak schopnost jednotlivých metod rozšířit datové sady klesá se zvětšující mírou nevyvážení a rozmanitostí těchto sad.

Klíčová slova datová augmentace, nevyvážený dataset, předzpracování dat, vyvažování tříd, neuronová síť, generativní adversariální sítě, autoenkodér

Contents

List of figures

List of tables

Introduction

Effective data processing has become highly desirable in today's world. Over the centuries, we have collected an enormous amount of data, which grows in size day by day. However, the question that still stands is how to make them useful. People eventually discovered that analysis of data could provide them with valuable information from understanding the past to predicting future processes. This discovery has started an era of vigorous exploration of various techniques and tools for data analysis.

Motivation

Computer recognition and understanding of image data have become one of the primary areas of interest. Research in computer vision and image analysis has made tremendous progress over the last decade. It is not that long ago that we were not able to detect an animal in a picture reliably. Today we can be proud of the ability to generate faces of people who never existed or advanced car assistance software with real-time object detection. Despite this progress, we still face many difficulties when it comes to image classification. One of the main factors causing problems for image classification models is the data itself.

During training, machine learning models require a vast amount of data that we do not always have. One possible solution is to collect more data samples, but this would take too much time. Another current problem are data privacy policies, which forbid the use of any data from datasets containing personal information.

Moreover, the majority of existing image datasets have a different number of samples for each class, leading to what we call an imbalanced dataset. This fact does not bother us as long as every class has approximately the same size. It becomes a real problem when one of the classes contains a signifi-

cantly smaller number of representatives. A model trained on such a dataset can be heavily biased towards the overrepresented class. This is, for instance, a quite frequent phenomenon in medical datasets. In many studies, researchers collect and survey data from patients during various examinations. However, only a small group of patients usually has the suspected disease, and therefore most of the data comes from healthy people. Another common situation is when the datasets contain only data from some specialised diagnostic medical equipment, such as a medical scanner. In this case, the examined patients typically already have a high possibility of having a disease. Consequently, the majority of the gathered data contains information about ill patients.

A general approach to alleviating this problem is called data augmentation. There are several possibilities of augmenting datasets, from simple standard ones such as geometric transformations to more advanced approaches such as generative adversarial networks. Since their introduction in 2014 [1], they have become extremely popular thanks to their ability to generate artificial images indistinguishable from the original images. The quality of pictures created using generative adversarial networks has proven to be high, and they quickly became one of the most attractive areas in the machine learning world.

Objectives

This thesis aims to:

- introduce widely used data augmentation methods, focusing on generative models and models derived from them,
- survey state-of-the-art techniques and algorithms used to restore balance in imbalanced datasets,
- choose and reimplement two state-of-the-art generative adversarial networks based on the descriptions presented in their research papers. Experimentally compare the reimplemented algorithms and discuss their limitations.

The thesis is organized into six chapters. First, Chapter 1 discusses both fundamental and advanced methods used for the augmentation of image data. Next, Chapter 2 presents state-of-the-art techniques using deep learning models. A detailed introduction and implementation details of two generative adversarial networks, on which this thesis focuses, can be found in Chapter 3 and Chapter 4, respectively. Finally, in the last two chapters, the achieved experimental results are presented, and eventually, the contribution of this thesis and future work are stated.

Chapter 1

Data augmentation

In this chapter, both fundamental and advanced methods for image data augmentation will be described.

Thanks to extensive research in the machine learning field, there is a variety of models used for image-related problems such as image classification or image detection. The architecture of these models quickly became more elaborate and with the models' complexity also increased the number of parameters, which can reach up to several million. However, the models' parameters need to be adjusted to achieve reasonable results. This is done through training on available data.

Sophisticated models (e.g., deep neural networks) require a significant amount of data, which are crucial for training. Data augmentation techniques enable us to expand our dataset by modifying the original images [2]. However, not every method always provides us with plausible images. Augmented datasets can greatly improve the model's performance, but they can also deteriorate it. The intention is to increase the amount of data with new samples relevant to the dataset, which can be achieved by carefully applying these techniques or their combinations.

Image data augmentation is not used solely for training complex models. There are many reasons why the generation of artificial data is much needed. Firstly, we often work with imbalanced datasets, where the targeted data points form a minority, and the models often fail when it comes to classifying them. Secondly, the data can contain personal information that cannot be used (such as medical records), and therefore we need to recreate similar data [3]. Other benefits of generating synthetic data include the reduction of overfitting and the ability to avoid unsatisfactory generalisation, which are both highly undesired when training a classifying model.

Given that this thesis focuses on imbalanced image datasets, in the following sections, several popular approaches used to tackle this problem will be examined. Firstly, we will focus on more straightforward methods and gradually move on to more sophisticated generative techniques. Furthermore, special attention will be given to selected generative models.

Generative models are a subset of machine learning models which can produce new data similar to their training dataset. The models often require a huge amount of data to approximate its distribution and to be able to retrieve relevant samples from it. In general, the original distribution and learned distribution do not need to be exactly the same in order for the generated samples to be unrecognisable from the real data.

Several models are capable of achieving such results, but some of the most interesting and promising for image data synthesis have proven to be generative adversarial networks (GAN) [1] and variational autoencoders (VAE) [4]. Both models are experiencing great success nowadays and will be elaborated upon in the following text [5].

1.1 Geometric transformations

In this section, basic data augmentation methods used for minor dataset alternations will be presented. Most of them consist of various geometric transformations and some of them include modifications such as altering the colour spectrum or adding noise. The listed techniques are highly popular because they are easy to understand and simple to implement. Their usage accelerated with the release of Python libraries [6, 7] for data science, which became highly appealing for students and newcomers to the machine learning field.

The application of these methods provides us with new samples, where for the creation of one new data point only a single sample from the original data distribution is used. However, the expanded dataset does not contain any new information, just new combinations of information that we already had [8].

1.1.1 Image flipping

Images can be flipped horizontally and vertically. However, the vertical flip is used much less often because it may disrupt orientation-related features. Applying a vertical flip to text recognition datasets or datasets containing images with objects positioned on the ground or with the sky as a background would not be very helpful. In most cases, horizontal flips make much more sense within the context of the dataset.

Figure 1.1: Application of geometric transformation to the original image. The figure illustrates, from left to right, the original image and its augmentations using rotation, shifts and colour scaling.

1.1.2 Image rotation

Rotational augmentation is done by rotating images clockwise or anticlockwise. The range can be between 0 and 360 degrees, but slight rotations, such as can be seen in Figure 1.1, are more common than significant rotations. Once again, a high rotation degree may not preserve the needed information.

Another problem that occurs after applying random rotations to images are blank areas. Usually, we need to maintain the shape of the images, and therefore we cannot just crop them. This topic will be discussed further below.

1.1.3 Image shifts

Image shifts move images left, right, up or down. This method has proven to be useful for datasets where images need to be centred (face recognition), but also when we need to force our model to look at the edges (object detection). As the original image is shifted in one direction, part of the image is being cut off, and the opposite side needs to be filled with meaningful pixels as illustrated by Figure 1.1.

1.1.4 Image cropping

This method is primarily used for resizing the dataset and replacing original images. One of the reasons why this transformation is applied is that the dataset may contain images of different shapes; another reason is that we need the images to be a specific size. In this case, filling is usually not needed, because images are normally resized to smaller dimensions.

1.1.5 Image colour scaling

The resulting colour of images is basically a combination of the colour channels. By separating them, changing them and bringing them back together, the dataset can be expanded with new tinted samples. These samples can vary in different ways, such as brightness (as shown in Figure 1.1) or contrast. This

technique also enables more advanced changes such as whitening, highlighting and many other desirable image enhancements.

1.1.6 Gaussian noise injection

When the model is not successful in learning global patterns, adding Gaussian noise to the dataset is a widespread technique, which can enhance its performance. This so-called noise injection is a method in which a matrix of random values with zero mean is added to the original image. The applied amount of noise needs to be chosen carefully, as many pixels not initially belonging to the original picture may result in the model's incapability to learn any features at all.

1.1.7 Fill strategies

After applying geometric strategies such as rotations or shifts, part of the image is removed and the other part needs to be completed with reasonable content. To preserve the original shape, it would be sufficient to fill the blank areas with some random colour. Most of the tools fill the missing edges with black. This is typically not enough as the transformed images are used for the training of the model. There are several approaches to making assumptions of the blank areas [9]. The following strategies are available within the `fill_mode` argument in the `ImageDataGenerator` class [7].

- **Constant** – Filling the edges with a constant value is the default strategy, but mostly inappropriate. Images containing landscape or complex structures demand better assumptions, on the contrary, using this technique on MNIST images or photographs of objects set in a simple environment would be sufficient. The constant value is modifiable, and often it is most sensible to choose the background colour.
- **Nearest** – This method duplicates pixels neighbouring with edges of blank areas.
- **Reflect** – The boundary between missing pixels and the image forms an imaginary mirror, which can at first glance create an illusion of an ordinary image.
- **Wrap** – Every blank space is filled with pixels located near the opposite area of missing pixels.

1.2 Distance-based methods

Oversampling minority-class, in its basic form, means creating exact copies of these images, which results in classes of the same size. However, this brings us no new information. More complex oversampling methods are the synthetic minority over-sampling technique (SMOTE) [10] and the adaptive synthetic sampling approach (ADASYN) [11], which both generate new synthetic examples by combining neighbouring samples of minority-class [3].

The SMOTE is a technique, which generates new samples using under-sampled class examples and its k-nearest neighbours. Usually, only some of the nearest neighbours are used depending on the number of synthetic samples that we need. A single synthetic sample is obtained by calculating the difference between the feature vectors of the sample and its nearest neighbour. The resulting vector is multiplied by a constant between 0 and 1, and then added to the feature vector of the sample. Generating the new sample is achieved by moving the original sample in the direction of its neighbour [10].

The ADASYN method is based on the idea of SMOTE and other algorithms derived from it. The main improvement is that most synthetic data is generated from minority-class samples, which the model finds hard to classify [11].

The opposite method to oversampling is undersampling, which cuts down images of majority-class. Although our dataset is balanced, this leaves us with even less information.

1.3 Generative adversarial networks

Goodfellow et al. [1] introduced GAN as a model based on the min-max game theory. The framework is resembling a two-player zero-sum game, the solution of which is known as Nash equilibrium. In our case, the players are two neural networks called generator and discriminator. As we already know what the GAN abbreviation stands for, one can assume that this game is competitive, instead of cooperative.

The generator, a generative model, has noise from some distribution on the input, and it outputs a sample from the model distribution. This noise is usually a vector from Gaussian distribution, which has proven to be beneficial. The generator learns to map the input distribution into the model distribution, which is done via adversarial training. His task is to learn the mapping well enough to sample data from the model distribution, which are indistinguishable from the real data.

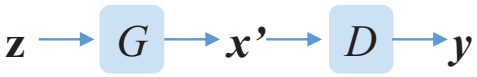


Figure 1.2: The architecture of a GAN. Source: [12]

The discriminative model's responsibility is to verify the authenticity of the data. On its input, the discriminator may receive both original and generated data. The goal is to predict whether the data are real or artificial. However, the output is not a binary answer, but rather the probability that the input originates from the real data distribution.

GAN's training is a simultaneous process in which both of the networks try to improve their performance. As usual for training machine learning models, the training consists of epochs in which GAN goes through the whole dataset. The original data are split into batches, which are evaluated in individual steps. For each step, a batch of vectors $\boldsymbol{z}$ is randomly generated from the latent space $p_{\boldsymbol{z}}$. The generator G receives noise $\boldsymbol{z}$ on the input and generates a sample $G(\boldsymbol{z})$ from the generative distribution $p_{\boldsymbol{g}}$. This sample is fed into the discriminator D for its evaluation. At the same time, it also receives samples from the real data $\boldsymbol{x}$. The discriminator yields a number between 0 and 1, which represents the network's confidence in the data authenticity.

Both generator G and discriminator D parameters are being tweaked via back-propagation with an intention to improve their performance. The discriminator D is led to differentiate between the original $\boldsymbol{x}$ and generated data $G(\boldsymbol{z})$, in other words, to maximise the probability that data are correctly classified. On the contrary, the generator G is trying to minimise this probability. Based on the discriminator's D opinion, generator G determines how to improve the credibility of the produced data $G(\boldsymbol{z})$. Both networks simultaneously optimise the following function F:

$$F(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{1.1}$$

Even though the type of data is not restricted, the author himself trained the GAN on image datasets. The original architecture includes fully connected networks, which were replaced by convolutional layers with the introduction of deep convolutional generative adversarial networks (DCGAN) [13]. They are built on top of convolutional neural networks (CNN) [14] that contribute to the success of computer vision tasks. The DCGAN architecture has proven to be more appropriate for image data because convolutions are capable of finding spatial correlations. Nowadays, when talking about GAN concerning image datasets, we often mean a DCGAN based model, as it became a standard for extensions of the GAN architecture.

1.4 Variational autoencoders

An autoencoder is a model capable of converting the input data into an internal latent space representation and reconstructing them. Depending on the size of the dimension of the internal latent space, we can divide autoencoders into two categories: overcomplete and undercomplete. We will focus on undercomplete autoencoders, which transform the input data into a smaller dimension. This process is handled by two parts of the network: encoder and decoder [2].

The encoder E transforms the data sample $\boldsymbol{x}$ to a vector $\boldsymbol{z}$ from lower dimensional space by finding and extracting only the most substantial information. The decoder G receives this vector $\boldsymbol{z}$ on its input and puts together the contained information. Decoder's output $\boldsymbol{x'}$ has the same format as the original data $\boldsymbol{x}$, which has the encoder E on its input. The only difference is that the output data were compressed and then decompressed by passing through a bottleneck latent layer. Due to the reduction of dimension, we essentially lose some of the less significant information.

Autoencoders belong to the category of unsupervised learning methods, which means that both networks need to learn the transformations by themselves. During the training, the encoder E learns how to perceive the information and shrink it into a hidden lower-dimensional representation, and the decoder G learns to use this information in order to re-create the input data $\boldsymbol{x}$. They use a reconstruction loss $\mathcal{L}(\boldsymbol{x}, G(E(\boldsymbol{x})))$, which tells them how much the data differ.

For calculating the dissimilarity between the original and reconstructed data, various reconstruction loss functions $\mathcal{L}$ can be used, such as binary cross-entropy or mean squared error. Information from such loss functions is distributed using backpropagation, which pushes the networks in the direction of reducing the loss value. Over time, they improve their performance and effectively generate data from their compressed form. Considering that the data $\boldsymbol{x'}$ are usually lower-quality approximations rather than duplicates, an autoencoder can be understood as a lossy compression function. These approximations can still be nearly indistinguishable from the real data to the human eye.

One of the variations of the autoencoder architecture is a variational autoencoder (VAE) [15], which, unlike the vanilla (i.e. standard as was described above) autoencoder, can generate synthetic data. Several changes had to be made to make this possible.

In order to be able to generate diverse data, we need to have a distribution to sample from. Thus, it would be useful for the hidden latent space to act like

Figure 1.3: The architecture of a VAE. Source: [12]

a normal distribution. This is achieved by the encoder E yielding two sets of parameters: a vector of means $\boldsymbol{\mu}$ and a vector of standard deviations $\boldsymbol{\sigma}$. To ensure that the distribution on the encoder's output $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ approximates the standard normal distribution $\mathcal{N}(0, 1)$, we use a measure called KL-divergence [16] as another loss function (Equation 1.2). KL-divergence measures the similarity of two probabilistic distributions, in our case, we minimise the difference between $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ and $\mathcal{N}(0, 1)$ [15, 17].

$$D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}) \,\|\, \mathcal{N}(0, 1)) = \sum_{i=1}^{n} \boldsymbol{\sigma}_i^2 + \boldsymbol{\mu}_i^2 - \log \boldsymbol{\sigma}_i - 1 \tag{1.2}$$

We sample from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$, but because the process of sampling a latent vector $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ is stochastic, we cannot run backpropagation through it. This problem is resolved by sampling from $\boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$ and transforming the sample to $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$. Using the reparametrisation trick, we represent $\boldsymbol{z}$ as $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$. This makes it possible to differentiate the sampling operation and backpropagate through the whole network.

The decoder G learns to transform the sampled vector $\boldsymbol{z}$ to an output, which resembles the input data point $\boldsymbol{x}$. Moreover, thanks to KL-divergence loss, once the model is trained and we want to generate new samples we can simply sample $\boldsymbol{z}$ from $\mathcal{N}(0, 1)$, with no need for additional reparametrisation.

Chapter 2

State-of-the-art

This chapter contains a theoretical overview of the surveyed state-of-the-art image data augmentation techniques based on deep learning. The methods will be presented chronologically according to the year they were introduced.

The restricted Boltzmann machine (RBM) [18], also known as a harmonium, is one of the earliest graphical probabilistic models used for deep learning. The architecture of RBMs consists of only two layers, so they do not meet the criteria for being deep models themselves. However, they are used as components of other deep models. The layers in RBM models are one of two types, either a visible layer or a hidden one. The connections between the model's units are analogous to an undirected bipartite graph. In other words, the units in the visible layer are connected to every unit in the hidden layer, but the layer's units are not connected among themselves. The model is derived from the Boltzmann machine (BM) [19], which is basically a network of visible and invisible nodes with connections among all nodes. In this sense, the RBM can be understood as a restricted variation of the BM. Another difference is that RBMs are categorized as generative models [2].

There are many extensions of RBMs, which contain duplicated layers, modified connections or other changes compared to the original architecture [2]. Moreover, the RBMs are often compared to GANs and VAEs, with which they share a similar idea. It should be noted that even though the mentioned models are more recent, RBMs can still outperform them in some cases [20].

The success and growth of generative models served as an impulse for developing techniques for sampling and visualising the latent space. Several captivating ideas can be found in [21], where the author proposes the following methods. One of the frequently used techniques to demonstrate a model's effectiveness is linear interpolation within the latent space. The problem is

that it traverses between locations that are highly unlikely considering the prior distribution. As a solution, the author proposes to use spherical linear interpolation. To find and visually represent analogies, the author devised a tool called J-Diagram. Moreover, it can be useful in comparing the generated results over epochs during training. Another problem that arises when training generative models, is that the manifold of latent projections of trained samples is usually only a small subset of the latent space. This leads to a sparse latent space, which contains many reoccurring dead zones. To forego this situation, we can precompute dense locations on the manifold and then discover the nearest known neighbours in the latent space. Manifold interpolated neighbour embedding (MINE) combines this strategy with an interpolation mechanism, which enables discovering new data points in latent space with good structure and visualising them [21].

One of the many tasks that generative models are used for is mapping images from one representation to another. However, it is unpleasant that for every variation of image translation between two domains, we need to design a new model for these specific domains even if it is essentially the same problem in all cases. This obstacle inspired the authors of pix2pix [22], a model with a general approach to image-to-image translations. The model is fed with a labelled input image from one domain and expected to generate an output image from the other domain. This is the reason why the framework is highly influenced by the architecture of a conditional GAN [23], which has the ability to learn the conditional generative model of data. The framework achieved sound results with no need to hand-engineer the loss functions. Thanks to the available interactive demo, the idea behind pix2pix also attracted the broad public [22].

The authors of cycle consistent GAN (CycleGAN) [24] took image-to-image translation one step further. They are aware of the lack of paired image datasets, so the model is built to be trained on unpaired training data. Instead of learning the mapping between the specific images, the model learns to capture the characteristics of domains. Moreover, the model has another impressive enhancement; it achieves cycle consistency of translation between domains — the ability to translate the images to the target domain and convert them back to the source domain. In order to deliver this consistency, the authors tailored the cycle consistency loss. There are two losses of this kind used in the model, one forward and one backward consistency loss. The model consists of two generators and two discriminators. The losses encourage these networks so that the translations between the domains are inverse to each other. Combined with an adversarial loss, the CycleGAN achieved impressive results [24].

Another generative model that builds upon the idea of conditional GANs [23] is called data augmentation generative adversarial network (DAGAN) [25].

The main aim of the proposed model is to improve learning for some specific problem thanks to learning on other similar ones. DAGAN enables training in low-data regimes by pre-training the network on a source domain with a large number of data points and generating new data for the target domain with fewer representants. With the aim of not being limited by the classes of the source domain, the generator is not supplied with the image's label. Instead of learning the manifold for a specific class, the generator learns only on features present in the image, not influenced by the class label. Images from the same class are generated by sampling from the learned manifold. As a result of being independent of the classes, the model can capture cross-class transformations and be used to obtain data for unseen novel classes [25].

Both generative models presented in the previous chapter, GAN and VAE, were extended and combined into a model called CVAE-GAN [12]. By class conditioning a VAE we get an extension called conditional variational autoencoder (CVAE) [26]. Using additional label information, we are able to direct the model to generate samples for a given class. To improve the quality of the generated pictures, the authors added two networks. One for discriminating the authenticity of images and the other for classifying them. In addition to reconstruction loss, we try to match the features of reconstructed images against the genuine image features during the model's training. The authors achieve competitive results even for challenging tasks such as face generation [12].

An innovative idea for data augmentation was introduced by balancing GAN (BAGAN) [27], which can generate high-quality images even when trained with an imbalanced dataset. Authors solve the problem by using all (both majority and minority) images during training, where the model learns features from majority classes and uses them to produce minority-class images. A major architectural enhancement consists of the initialising GAN using an autoencoder. Authors first train the autoencoder to learn characteristic features of all images in the dataset. The next training step involves transferring the autoencoder's knowledge into the GAN. The generator is initialised with the weights of the decoder and the discriminator with the weights of the encoder. The last step is the adversarial training itself. This combination of GAN and autoencoder enables class conditioning and helps avoid mode collapse [27].

Transferring knowledge by pre-training and fine-tuning the discriminative networks served as inspiration for trying this technique with generative models. Authors of transferring GANs [28] have studied the domain adaptation and came to several conclusions. During the network's configuration, multiple combinations of the pre-trained generator and discriminator were tried. The experiments have shown that the best results are obtained by pre-training both of the GAN's networks. While transferring only the discriminator improved

the results, transferring only the generator degraded the GAN's performance. Moreover, this configuration made training more stable. The authors also examined various source and target domains and concluded that it is better to choose a dataset with few densely sampled classes as a source domain. Furthermore, the paper's results suggest that the density of the dataset is more important than its diversity. As stated before, it is recommended to choose a dataset with one or few densely sampled classes as a source domain rather than diverse datasets. The paper has proved that images generated by already pre-trained GANs are of higher quality, even within a shorter training time and limited datasets [28].

Data augmentation using deep learning models is undoubtedly a popular area of research. The already large group of models is expanding daily with new frameworks and techniques. Many of them present innovative ideas and original approaches to the problem of limited data. There is an increasing amount of up-to-date research papers, but their detailed introduction is beyond the scope of this thesis. Based on the surveyed algorithms and a consultation with the supervisor we decided to focus on CVAE-GAN and BAGAN.

Chapter 3

Implemented methods

In this chapter we will discuss the two image data augmentation methods we decided to focus on. Their brief description can already be found in the previous chapter introducing the state-of-the-art methods surveyed within this thesis. Both of them have complex architectures and present advanced concepts, which is why they deserve further examination and a detailed introduction. The thesis assumes the reader has a fundamental understanding of machine learning concepts; hence we will leave out their explanation, which can be found in [29].

3.1 Task definition

Imbalanced datasets have become a concerning problem and have been extensively studied recently. The aim of this study is to examine and demonstrate the application of deep learning models derived from GANs on a class balancing task. Based on the survey, we use CVAE-GAN and BAGAN to generate minority-class images. The models are implemented according to the framework presented in [12, 27] and tailored to suit our datasets. Both these networks enable class-conditioning, which is the key characteristic needed for generating images of a specific class. The uniform distribution of images between the classes is ensured by augmenting the imbalanced dataset with the created minority-class images. As a result, we obtain a balanced dataset, which is more suitable for further use, for instance, the training of predictive and classification models. We decided to present the possible benefits by investigating the accuracy of classifiers trained on the augmented datasets.

3.2 CVAE-GAN

CVAE-GAN [12] was not designed for the imbalanced dataset problem in particular, but it can suit the balancing task appropriately. Initially, the model was constructed to generate highly-detailed, super-resolution images, e.g., human faces, flowers and birds. For this purpose, the authors took inspiration from two advanced deep learning models, VAE [4] and GAN [1]. They merged them into one framework under a conditioned generative process, as illustrated in Figure 3.1. In addition, the model comes with several novel ideas. The image is modelled as a combination of label and latent attributes representing image features. Secondly, the authors adopt sophisticated loss functions tailored to push the model towards its best performance.

The model's framework consists of four networks: encoder E, generator G, discriminator D and classifier C. No architectural changes disrupting the initial framework are applied to the networks themselves. The general task that each of the networks had in the original models (VAE and GAN) remains but is not sufficient. To make the training more stable, the authors introduced two mean feature matching objectives for the generator G. Both use a feature representation acquired through the other network's layers.

Another common problem appearing in relation to generative adversarial networks is mode collapse. The encoder E and the generator G are given a new objective to prevent this problem. They are used to obtain a mapping between the original $\boldsymbol{x_r}$ and reconstructed $\boldsymbol{x_f}$ image samples, analogous to the VAE approach. After acquiring this mapping, we are able to calculate the difference between $\boldsymbol{x_r}$ and $\boldsymbol{x_f}$ using L_2 reconstruction loss and a pair-wise feature matching loss using features from the other network's layers. Both of the above mentioned novel feature matching mechanisms will be presented in more detail further in the text. The main reason for their introduction was to make the training converge faster and to increase its stability.

3.2.1 Training

As usual, during the training, the images are processed in batches. Although this time, we differentiate between three distinct types of image batches, where the first type is the original pictures $\boldsymbol{x_r}$ themselves. The second category is represented by the reconstructed images $\boldsymbol{x_f} = G(\boldsymbol{z}, \boldsymbol{c_r})$, where $\boldsymbol{z} = E(\boldsymbol{x_r}, \boldsymbol{c_r})$ and $\boldsymbol{c_r}$ is the class label of $\boldsymbol{x_r}$. The last batch type consists of images $\boldsymbol{x_p} = G(\boldsymbol{z_p}, \boldsymbol{c_p})$ generated using random samples. Two parameters are needed for the generation of a single image, a vector randomly sampled from the normal distribution $\boldsymbol{z_p} \sim \mathcal{N}(0, 1)$ and a randomly generated class label $\boldsymbol{c_p}$.

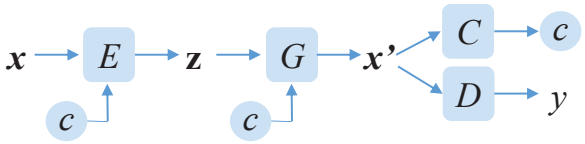


Figure 3.1: The architecture of a CVAE-GAN. Source: [12]

Due to the presented feature matching mechanisms, the generator's G training process changes significantly. The rest of the networks keep the training the same as in VAE and GAN. However, because of having to differentiate between reconstructed and generated outputs $\boldsymbol{x_f}$ and $\boldsymbol{x_p}$, a slight adjustment is necessary. Moreover, we newly use these networks as a source for the values needed for the computation of several loss functions.

As presented in Section 1.3, the training of the GAN's components can be seen as a min-max game. Such a relationship between two networks can be described with a shared loss function defined in Equation 1.1. The discriminator's D loss function remains almost unchanged. The function used in CVAE-GAN is only extended through the inclusion of $\boldsymbol{x_p}$, so we end up with a loss function $\mathcal{L}_D$ as described in Equation 3.1.

$$\mathcal{L}_D = -(\log(D(\boldsymbol{x_r})) + \log(1 - D(\boldsymbol{x_f})) + \log(1 - D(\boldsymbol{x_p}))) \tag{3.1}$$

Since the introduction of the mean feature matching objective for the generator G, the network G does not preserve the loss function used in GAN. Instead, its new task is to minimise the difference between the discriminator D's features of real and generated image samples, $f_D(\boldsymbol{x_r})$ and $f_D(\boldsymbol{x_p})$, respectively. These features $f_D(\boldsymbol{x})$ are obtained from the intermediate layer of the network D. Therefore, the generator G tries to minimise

$$\mathcal{L}_{GD} = \frac{1}{2}\left\| \frac{1}{m}\sum_{i}^{m} f_D(\boldsymbol{x_r}) - \frac{1}{m}\sum_{i}^{m} f_D(\boldsymbol{x_p}) \right\|_2^2, \tag{3.2}$$

where m is the batch size. The loss function $\mathcal{L}_{GD}$ might suggest that the mean features are estimated using the images from the current batch only. However, the calculated centres of features are biased with the data from the previous epochs using the moving average method in order to make the means of the features more meaningful.

The proposed mean feature matching for conditional image generation brought another loss function $\mathcal{L}_{GC}$, which the network G tries to minimise (Equation 3.3). Its role is to drive the generator G towards reducing the difference between the features of the original and the synthesised images. The features

$f_C(\boldsymbol{x})$ are extracted from the intermediate layer of the classifier C. These features are not simply averaged through the whole batch, like in the previously mentioned mean feature objective. Both image batches are broken down into groups according to their class. With k being the number of classes in the dataset, we end up having two sets of k groups, one set containing real images $\boldsymbol{x_r}$ and the other generated images $\boldsymbol{x_p}$. After obtaining the corresponding features, the mean of features $f_C^{\boldsymbol{c}}(\boldsymbol{x})$ of the images from the specific class $\boldsymbol{c}$ are calculated by averaging the features within one group only. Since a uniform distribution of classes within the batch is not ensured, the relevant feature means are achieved using moving averages again. Let us define $\mathcal{L}_{GC}$ as:

$$\mathcal{L}_{GC} = \frac{1}{2}\sum_{\boldsymbol{c}}^{k}\left\|\frac{1}{m_{\boldsymbol{c_r}}}\sum_{i}^{m_{\boldsymbol{c_r}}} f_C^{\boldsymbol{c}}(\boldsymbol{x_r}) - \frac{1}{m_{\boldsymbol{c_p}}}\sum_{i}^{m_{\boldsymbol{c_p}}} f_C^{\boldsymbol{c}}(\boldsymbol{x_p})\right\|_2^2 \tag{3.3}$$

Here, $m_{\boldsymbol{c_r}}$ stands for the number of real images $\boldsymbol{x_r}$ with label $\boldsymbol{c_r}$ and $m_{\boldsymbol{c_p}}$ for the number of synthesised images $\boldsymbol{x_p}$ with label $\boldsymbol{c_p}$.

As presented in Section 1.4, to ensure the mapping between the original $\boldsymbol{x_r}$ and the reconstructed images $\boldsymbol{x_f}$, we first need to obtain a mapping between the real pictures $\boldsymbol{x_r}$ and vectors $\boldsymbol{z}$ from latent space. For this purpose, the encoder E is assigned with the loss function $\mathcal{L}_{KL}$ analogous to Equation 1.2. Once again, the principles of the reparametrisation trick apply in this scenario as well. After obtaining this mapping, we can produce the reconstructions $\boldsymbol{x_f}$ using the generator network G, jointly training both of the networks together. Having both types of images $\boldsymbol{x_r}$, $\boldsymbol{x_f}$, we can measure their difference using L_2 reconstruction loss and pair-wise feature matching loss. The encoder E and generator G minimise this difference using the $\mathcal{L}_G$ loss function.

$$\mathcal{L}_G = \frac{1}{2}(\|\boldsymbol{x_r} - \boldsymbol{x_f}\|_2^2 + \|f_D(\boldsymbol{x_r}) - f_D(\boldsymbol{x_f})\|_2^2 + \|f_C(\boldsymbol{x_r}) - f_C(\boldsymbol{x_f})\|_2^2) \tag{3.4}$$

In summary, the generative network G attempts to diminish the value of three loss functions: $\mathcal{L}_{GD}$, $\mathcal{L}_{GC}$ and $\mathcal{L}_G$. The encoder network E is affected by loss functions $\mathcal{L}_{KL}$ and $\mathcal{L}_G$.

The network C receives an original image $\boldsymbol{x_r}$ on its input and outputs the probability of the sample belonging to each of k classes. The improvement of its capability to classify the images into the correct category $\boldsymbol{c_r}$ is ensured by using $\mathcal{L}_C$ loss.

$$\mathcal{L}_C = -\log(P(\boldsymbol{c_r}|\boldsymbol{x_r})) \tag{3.5}$$

To sum up, the model is trained end-to-end and aims to minimise:

$$\mathcal{L} = \mathcal{L}_D + \mathcal{L}_C + \lambda_1\mathcal{L}_{KL} + \lambda_2\mathcal{L}_G + \lambda_3\mathcal{L}_{GD} + \lambda_4\mathcal{L}_{GC}, \tag{3.6}$$

where λ_i is tunable and influences the weights of the particular loss functions.

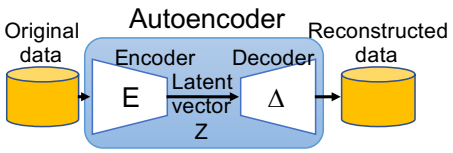
3.3 BAGAN

Unlike the previously presented model, the BAGAN [27] was explicitly designed for restoring balance in image datasets. The traditional approach to training generative models consists of gathering the minority-class images and only using them to create new ones. Nevertheless, minority-class images are mostly small in number, which leads to the models being provided with an insufficient amount of information. Consequently, it is hard to train GANs to generate new diverse images. Since the images within the dataset are typically related, the BAGAN proposed to utilise the information from all available classes in order to train a generative model.

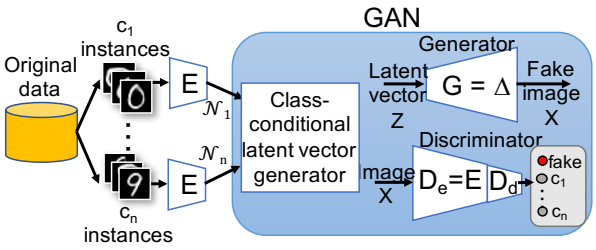
This is guaranteed by coupling the GAN with an autoencoder. After the autoencoder's training, the knowledge about the image features is transferred into a GAN. This handover of information is achieved thanks to GAN being initialised with the autoencoder's weights. Such initialisation is provided by re-using the autoencoder's networks architecture analogously in the GAN's networks as presented in Figures 3.2a and 3.2b. The decoder Δ and the generator G share the exact same topology. At the same time, the discriminator D matches its first layers D_e with the encoder E and completes them with a dense layer followed by an activation function D_c, as depicted in Figure 3.2b. Moreover, the initialisation helps to prevent some problems occurring during the training of vanilla GANs, such as mode collapse. In addition, a novel mechanism for class-specific latent vector generation, which enables us to class-condition the generative process, is introduced. We will discuss this mechanism in detail in the following paragraphs.

However, the mentioned enhancements are still not sufficient to make the BAGAN architecture suitable for a class-balancing task. At this point, in an attempt to fool the discriminator, the generator would instead focus on producing the majority-class images. Having more information about them, the generated images from majority-class would be more realistic, and the value of the adversarial loss function would decrease faster. In order to avoid this behaviour, the generator is forced to generate images from all classes during training uniformly.

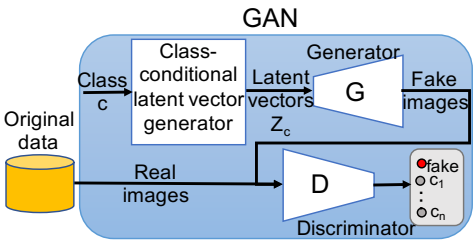
Furthermore, artificial pictures are required to look real and match the class label at the same time. In this case, the authors took inspiration from ACGAN's [30] discriminator, which has two outputs. One for judging the authenticity of an image, and the other for determining its class. However, using the very same discriminator architecture has proven to be inappropriate. The BAGAN discriminator has been adjusted to output a single vector, which represents the discriminator's opinion of the image. The discriminator can either label the image with a corresponding class or discard it as being artificially gener-



(a) Autoencoder training.



(b) GAN initialisation.



(c) Training of the GAN.

Figure 3.2: Three stages of the BAGAN's training. Source: [27]

ated. As a result, the generator is penalised for not matching the class label, even if the image looks real.

3.3.1 Training

Apart from the mentioned architectural changes, BAGAN differs from the vanilla GANs in its training process as well. The training is divided into three stages shown in Figure 3.2.

The first step consists of the autoencoder's training. The autoencoder receives all images without their labels, and its only task is to learn to reconstruct them as illustrated in Figure 3.2a. The reconstructions get better in quality thanks to making use of the L_2 loss function.

The second stage incorporates the GAN initialisation and the preparation of the class-conditional latent vector generator. The setting described in the previous paragraphs allows the weights of the already trained autoencoder to

be transferred to the GAN. After doing so, the discriminator D possesses information about relevant features, which are substantial in the classification process. Due to the decoder Δ and generator G having the same network architecture, the vectors on their inputs $\boldsymbol{z}$ are equivalent. The authors have taken advantage of this fact and introduced a function called class-conditional latent vector generator. Firstly, all pictures in the dataset are separated into groups $X_{\boldsymbol{c}}$ according to their class label $\boldsymbol{c}$ (Figure 3.2b). One by one, every group is processed by the encoder $E(X_{\boldsymbol{c}})$. For each group two values describing the distribution $\mathcal{N}_{\boldsymbol{c}}$ of the encoder's outputs $\boldsymbol{z}_{\boldsymbol{c}} = E(X_{\boldsymbol{c}})$ are computed. These values are a mean vector $\boldsymbol{\mu}_{\boldsymbol{c}}$ and a covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{c}}$. By random sampling from the corresponding normal distribution $\mathcal{N}_{\boldsymbol{c}} = (\boldsymbol{\mu}_{\boldsymbol{c}}, \boldsymbol{\Sigma}_{\boldsymbol{c}})$ this function later supplies the generator with plausible latent vectors $Z_{\boldsymbol{c}}$ appropriate for the creation of images of a specific class $\boldsymbol{c}$.

The final step is the adversarial training of the generator G and the discriminator D, as illustrated in Figure 3.2c. The training consists of epochs, in which the data are processed in batches. The generator G is provided with a batch of latent vectors $\boldsymbol{z}_{\boldsymbol{c}}$ obtained by the class-conditional latent vector generator, which receives a batch of labels on its input. This batch has an even amount of vectors of each class. Therefore, the generator G can produce a batch of images uniformly distributed between n classes. The generator G is optimised to fool the discriminator D, so that the synthetic images are matching the class labels $\boldsymbol{c}$ instead of being labelled as fake. The amount of improvement needed to increase the quality of generated images is calculated using a sparse categorical cross-entropy loss function.

The discriminator D receives all of the counterfeit pictures for evaluation, but it learns only from a part of them. The batch learnt by the discriminator cannot be larger than the batch learnt by the generator. Consequently, the fake images make up only $\frac{1}{(n+1)}$ of the discriminator's batch size. The rest consists of real images from the dataset. These two subsets each have images uniformly distributed between classes, so the discriminator D receives both fake and real representatives of each class. Again, the discriminator D learns using the same loss function as the generator G. However, the goal is to recognize whether the image is synthetic or a real sample from a specific class $\boldsymbol{c}$.

Chapter 4

Implementation

In this chapter, the used technologies and implementation details of this thesis will be discussed and summarised.

4.1 Technologies

There is a wide range of various technologies that can be used for developing machine learning projects. One of the first and most important decisions is the selection of a programming language that will meet the requirements for the specific project.

To fulfil the purpose of this thesis, we opted for Python, one of the leading programming languages in the machine learning field. Python has simple syntax and good readability, which are one of the main reasons why this language is often the choice of newcomers to this field. However, Python is also appealing to skilled developers for many more reasons. It has become widely used thanks to the release of tools and libraries suitable for data science.

In the machine learning world, we usually work with large datasets. To go through each data point in order to get a clear idea about its content would be challenging and time-consuming. Instead, it is well-established to describe the dataset in a summarising table or visualise the information it contains. Visualising the data is also used during preprocessing or after obtaining the results. Powerful Python libraries for data visualisation are Matplotlib[1] and its extensional derivative Seaborn[2]. We mainly used Matplotlib, which is easier to use and provides clear and informative visualisations.

[1] `https://matplotlib.org/`
[2] `https://seaborn.pydata.org/`

The NumPy[3] library is popular for its intuitive interface, which enables both basic and complex numerical operations. Besides the available operations with multidimensional arrays and matrices, the library also contains many scientific computational tools used for various transformations and linear algebra.

A popular technology we use is Tensorflow [6], which has an ecosystem of machine and deep learning tools. Their availability and ease of use simplifies the development of new methods. One of the most popular technologies built on top of Tensorflow is Keras[4] high-level API. Tools available within the Keras library are explicitly designed for deep neural networks.

4.2 Specifics of the CVAE-GAN implementation

In this section we will discuss the implementation details of the CVAE-GAN [12] model. Besides describing our specific changes, we will also provide a comparison against the original implementation.

As already stated before in Section 3.2, the authors of CVAE-GAN focused on complex images of high resolution. Therefore, they reached for models with robust architectures. Since we work with different datasets and we are limited by computational power, we chose to use a simplified architecture. We built three CVAE-GAN models, each tailored for the generation of images from different datasets described in Section 5.1. The replication of the same model structure presented in the original paper is beyond the scope of this thesis.

For the encoder E, the authors adapted GoogleNet [31], the winning architecture of the renowned ImageNet Large Scale Visual Recognition Competition (ILSVRC), which took place in 2014. The model was also extended by a batch normalisation layer applied after each convolutional layer. Our encoder network E is a simplified version of the convolution network VGG16 [32], which placed second in the very same competition.

The encoder E is built with components, one of which consists of a convolution layer followed by an activation function and a batch normalization layer. These modules are then cascaded together, and some of them also followed by a max pooling layer. The encoder E receives on its input an image $\boldsymbol{x}$ and its label $\boldsymbol{c}$, which are concatenated together. In the original implementation, this merging is done in the last fully connected layer. We followed the same methodology in our implementation of the CVAE-GAN for generating MNIST [33] images, which is shown in Table B.4.

[3] `https://numpy.org/`
[4] `https://keras.io/`

The generator G is described in the paper as a model consisting of two fully connected layers followed by six sequences, each made up of a transposed convolutional layer with a batch normalisation layer and an upsampling layer. In the convolutional layers, the number of channels gradually decreases from 256 to 3 while the filter size increases from being size 3×3 to 5×5.

However, our adaptation of the model contains fewer layers and different parameters. The model G receives two values, a label $\boldsymbol{c}$ and a latent vector $\boldsymbol{z}$, which are merged in the first layers. Transposed convolutional layers with activation functions form the rest of the model architecture, as presented in Table B.1.

For the discriminator D network, both the original implementation and our implementation followed the architectural guidelines presented in DCGAN [13], as can be seen in Table B.2. Among the recommendations is to avoid fully connected and pooling hidden layers. It is also suggested to include batch normalisation layers and to use the leaky rectified linear unit (Leaky ReLU) [34] as the activation function. Moreover, we included a few dropout layers in the network D used in our model implemented for the CIFAR-10 [35] and Vehicles [36] datasets.

The original architecture used for the classifier C is based on Alexnet [37], which won the ILSVRC competition in 2012. The same properties which GoogleNet holds apply here, the architecture is sophisticated and requires a long training. Our set up is once again reminiscent of the VGG16 network, which results in it being more or less identical to our encoder E network. A thorough overview of our classifier C for one of the datasets is presented in Table B.3.

Unfortunately, the source code for the CVAE-GAN model is unpublished, and the paper provided us with only little information about the hyperparameters. Their choice of optimisers was unknown to us, so we experimented with the Adam [38] and RMSprop [39] optimisers with varying learning rates. In our case, it turned out that RMSprop subjectively outperformed the other in each of the three CVAE-GAN models. Within a single model, we set the same learning rate for every submodel's optimiser. We have discovered that good learning rates are 0.0002 for the MNIST and Vehicles datasets and 0.0003 for the CIFAR-10 dataset.

In the paper, the experiments were done on datasets having high resolution images, so they set the latent vector to be from 256-dimensional space. Our configuration of the latent vector dimension is set to 55 for the MNIST and 128 for the CIFAR-10 and Vehicles datasets.

The parameters in the loss function $\mathcal{L}$, as shown in Equation 3.6, were originally fixed to $\lambda_1 = 3$, $\lambda_2 = 1$, $\lambda_3 = 10^{-3}$ and $\lambda_4 = 10^{-3}$. We found this configuration inappropriate, primarily because of the low emphasis on minimising the $\mathcal{L}_{KL}$ loss described in Equation 1.2. We had difficulties establishing the distribution described by the encoder's outputs to approximate a standard normal distribution $\mathcal{N}(0, 1)$. Consequently, we could not just sample a latent vector $\boldsymbol{z}$ from $\mathcal{N}(0, 1)$ and feed them into the generator because the synthesised images were of insufficient quality and did not resemble original images from the dataset. Therefore, we tested the efficiency of higher λ_1 values. Sound results were achieved with λ_1 being 300, 50 and 100 for the MNIST, CIFAR-10 and Vehicles datasets respectively.

We lack the information about the number of epochs needed for the network to converge. We tried several configurations and found it was adequate to train the networks for 1500, 5000 and 3000 epochs for the MNIST, CIFAR-10 and Vehicles datasets respectively. We set the batch size to 300, 256 and 200 image samples.

4.3 Specifics of the BAGAN implementation

This section will describe the BAGAN [27] architectural details and differences between ours and the original implementation.

As opposed to the model presented in the previous section, the authors did not aim at generating images of fine-grained categories. The performed experiments were focused on restoring balance in imbalanced datasets, some of which were the same as ours. The implementation source code is publicly available in the GitHub repository[5]. This fact enabled us to examine the original architecture in detail. The acquired information helped us understand the model and re-implement it more accurately.

We followed the same procedure seen in the provided code and firstly prepared the shared encoder part. Its architecture consists of blocks, each containing a convolutional layer, a Leaky ReLU activation function and a dropout layer. The features at the output of the last layer are fed into the dense layer, which provides us with a complete encoder architecture, as detailed in Table B.7. The encoder E has a single input, a picture $\boldsymbol{x}$, and a single output, a latent vector $\boldsymbol{z}$.

As previously explained in Section 3.3, the decoder Δ and the generator G are equivalent $\Delta = G$. The first layer is a dense layer with a rectified linear unit (ReLU) activation function [40], which inputs the latent vector $\boldsymbol{z}$. The rest

[5] `https://github.com/IBM/BAGAN/`

of the architecture is made up of transposed convolutional layers with ReLU activations. For the datasets normalised between -1 and 1 the last layer has a tanh activation function, as indicated in Table B.5. Since the model is trained in the role of decoder first, it starts its training as a generator already pre-trained.

The discriminator D's architecture consists of a shared encoder part and a dense layer with a softmax activation function [2], as presented in Table B.6. The number of neurons in the dense layer depends on the number of classes in the given dataset. However, the discriminator D also does not start training from the very beginning. The shared encoder part is already initialised with the weights of the encoder, which was trained in the first training step of BAGAN, as illustrated in Figure 3.2.

Thanks to the code being published, we could inspect the values of the hyperparameters. They decided to use Adam with a learning rate of 0.00005 as an optimiser and also changed the values of some other optimiser parameters. We tried out this setup for optimisers and learning rates and found it to be sufficient, but we left the other parameters at their default values. However, for the Vehicles [36] dataset the training was done using RMSprop optimisers with a learning rate of 0.0002.

The dimensionality of the latent vector $\boldsymbol{z}$ was set to 100 by default in the original implementation. We empirically set the dimension to 10, 40 and 20 for the MNIST [33], CIFAR-10 [35] and Vehicles datasets respectively reflecting their subjective complexity.

In the first training stage the model was trained for 300, 800 and 1200 epochs and in the last stage for 800, 2000 and 2500 epochs for the MNIST, CIFAR-10 and Vehicles datasets respectively. The batch size was 275 for the MNIST and CIFAR-10 datasets and 220 for the Vehicles dataset.

Chapter 5

Experiments

One of the aims of this thesis is to reimplement and apply two state-of-the-art generative models in the task of balancing an imbalanced dataset. The implemented models, CVAE-GAN [12] and BAGAN [27], were trained to generate images of three datasets described in Section 5.1. For the comparison, we also created images using CVAE [26] and geometric transformations. In this chapter we will provide the conducted experiments and evaluation of the mentioned techniques.

5.1 Datasets

The experiments were conducted using three publicly available datasets. Two of them contain thousands of images and are easily accessible within the Keras library [7]. The third dataset is smaller and was manually assembled and preprocessed.

5.1.1 MNIST

The MNIST [33] is a well-known and widely used dataset in the machine learning world. The dataset overall contains $70,000$ grayscale images of handwritten digits in range 0 to 9. The representatives of these categories are uniformly distributed within the dataset. All images are of size 28×28 pixels.

5.1.2 CIFAR-10

The CIFAR-10 [35] also belongs among popular machine learning datasets. It has a total of $60,000$ colour images of 10 classes. Each class has 6000 representatives and depicts objects of categories such as horse, ship or car. The resolution of CIFAR-10 images is 32×32 pixels.

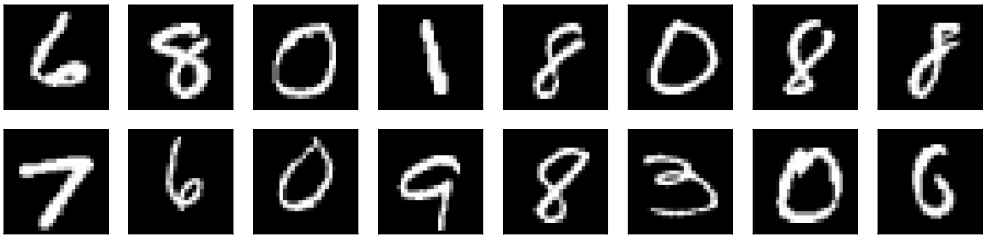


Figure 5.1: Image samples from the MNIST dataset.

Figure 5.2: Image samples from the CIFAR-10 dataset.

Figure 5.3: Image samples from the Vehicles dataset.

5.1.3 Vehicles

The Computational Vision Group at Caltech has images they used for their experiments published on their web page [36], and we found some of them interesting. Vehicles is the name given to the dataset we created by joining other smaller datasets. We created a dataset containing 2552 images overall by merging four sets of pictures called Cars 2001 (Rear), Cars 199 (Rear) 2, Motorcycles 2001 (Side) and Airplanes (Side) into three classes of vehicles. The images are of various sizes, so we resized them to 64×64 to suit our purpose.

5.2 Design of experiments

We manually imbalanced each original dataset by selecting and dropping a subset of a single specific class. Firstly, the datasets were split into train and test data in a ratio of 70:30. Addressing the amount of minority-class images, we kept two scenarios. In the first case, we dropped 50% and in the second, 75% of pictures of the selected class in the train data.

Since we use three datasets (MNIST, CIFAR-10 and Vehicles), three types of network architectures (CVAE, CVAE-GAN and BAGAN) and two imbalance scenarios, counting all the possible combinations gives us two groups of nine models. Each model in the first group is trained on data with minority-class containing circa 50% of original images. The second group of nine models is trained on data, where the minority-class is reduced to approximately 25% of original class representatives. Train data used for one group of models differs from the train data in the other group only in the amount of minority-class samples.

In the MNIST dataset, we decided to drop the representatives of the fourth class formed by handwritten number four. For the unbalanced CIFAR-10 dataset we have dropped representatives of the eighth class containing photos of ships. From the Vehicles dataset, the samples from the class containing images of cars were removed. The dropped images were selected at random. Nevertheless, to ensure that train and test datasets have the same composition, we always used the same random seed values. This way, the dataset split was deterministic.

All the trained models are applied to generating images of minority-class to restore the balance in the unbalanced datasets used for their training. Creating images using geometric transformations does not require training any model. The augmented datasets are evaluated using methods presented in the following section.

5.3 Evaluation

In the course of the experiments, we analyse the datasets using two methods. For the images of one dataset, we examined the original, unbalanced and four augmented variations of the dataset.

The first technique was used to measure the variety of the minority-class images called the structural similarity (SSIM) index [41]. There are several parameters that can be adjusted, but we used the standard SSIM implementation, as shown in Equation 5.3, which is available within the TensorFlow

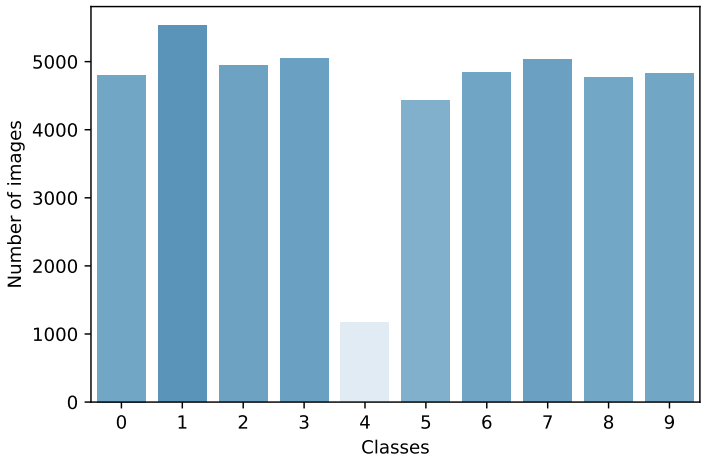


Figure 5.4: Class distribution of the unbalanced MNIST train dataset with 75% of minority-class images.

ecosystem. This metric takes two images $\boldsymbol{x}, \boldsymbol{y}$, each with N components, as its input and outputs a number between -1 and 1 representing how similar the images are. Nevertheless, the negative values are obtained in cases when the local image structures are inverted [42]. Otherwise, the values on the output are between 0 and 1. The closer the score is to zero, the smaller the resemblance between the images.

$$\mu_{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i \tag{5.1}$$

$$\sigma_{\boldsymbol{x}} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\boldsymbol{x}_i - \mu_{\boldsymbol{x}})^2} \tag{5.2}$$

The mean value and standard deviation are calculated analogously for the components of the image $\boldsymbol{y}$.

$$\mathrm{SSIM}(\boldsymbol{x}, \boldsymbol{y}) = \frac{(2\mu_{\boldsymbol{x}}\mu_{\boldsymbol{y}} + C_1) + (2\sigma_{\boldsymbol{xy}} + C_2)}{(\mu_{\boldsymbol{x}}^2 + \mu_{\boldsymbol{y}}^2 + C_1)(\sigma_{\boldsymbol{x}}^2 + \sigma_{\boldsymbol{y}}^2 + C_2)}, \tag{5.3}$$

where $\mu_{\boldsymbol{x}}$ is the mean value of components of $\boldsymbol{x}$ computed as shown in Equation 5.1, $\mu_{\boldsymbol{y}}$ is the mean value of components of $\boldsymbol{y}$, $\sigma_{\boldsymbol{xy}}$ is the covariance of components of $\boldsymbol{x}$ and $\boldsymbol{y}$, $\sigma_{\boldsymbol{x}}^2$ is the variance of components of $\boldsymbol{x}$ computed as shown in Equation 5.2 and $\sigma_{\boldsymbol{y}}^2$ is the variance of components of $\boldsymbol{y}$. C_1 and C_2 are adjustable hyperparameters.

Instead of calculating the SSIM for a pair of images, we needed to obtain the SSIM value for the entire sets of images. To achieve this goal, we randomly paired the images used to balance the dataset, calculated SSIM for each pair and averaged the values. We interpret the result as the capability of the model, that created the evaluated images, to generate diverse samples. Moreover, to have a reference point for the results, we measured the SSIM of the original minority-class images. To calculate their average SSIM, we used the same number of images as was used to balance the unbalanced datasets.

The second evaluation method depends on using a classifier. For each type of dataset (MNIST, CIFAR-10 and Vehicles), we constructed a simple classifier architecture and trained one classifier for each of the six mentioned dataset variations. The architecture details of the classifiers can be found in Tables B.8 to B.10. The classifier networks for the MNIST and Vehicles datasets are inspired by a Tensorflow tutorial[6]. The classifier for CIFAR-10 is based on the classifier network used for this dataset in CVAE-GAN.

Firstly, we split the original dataset into train and test data the same way we did when training the generative models. In addition, we split the training data, so that we use 75% of them as the train set for the first classifier, and the rest forms the validation set.

The second classifier needed to be trained on the unbalanced dataset. For this purpose, we employed the same data as we used to train the generative models. In other words, the train data were deprived of minority-class images using the same seed as we used before. Again, 25% of this data was used as a validation set.

For each of the other classifiers, we followed the same method of data preparation. We balanced the unbalanced dataset, which was previously used to train the generative models, using geometric transformations, CVAE, CVAE-GAN and BAGAN. Altogether, we have four balanced datasets, each augmented using a different method. Each dataset was split into train and validation data in a ratio of 75:25. Every pair of train and validation data is used to train and tune one classifier.

For each dataset, we train one classifier on original balanced data (MNIST, CIFAR-10 and Vehicles) and two classifiers on original unbalanced data. One for 50% and one for 75% imbalance scenario. Moreover, in every scenario, we train four classifiers within one dataset. In total, we have 3 classifiers trained on original balanced data, 6 classifiers trained on unbalanced data and 24

[6] `https://www.tensorflow.org/tutorials/images/cnn`

classifiers trained on augmented data. The classifiers' ability to correctly classify the test data is measured using two metrics, accuracy and F1 score.

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}} \tag{5.4}$$

$$\text{F1 score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{5.5}$$

where the recall is the ratio of true positives and the sum of true positives and false negatives. The precision is calculated as the number of true positives divided by the sum of true positives and false positives.

The accuracy represents the classifier's ability to classify the images from the test dataset correctly. The F1 score is computed as a harmonic mean of precision and recall for the minority-class. This metric is considered to be suitable for imbalanced datasets. For the evaluation, we always used the balanced test set containing original images.

Chapter 6

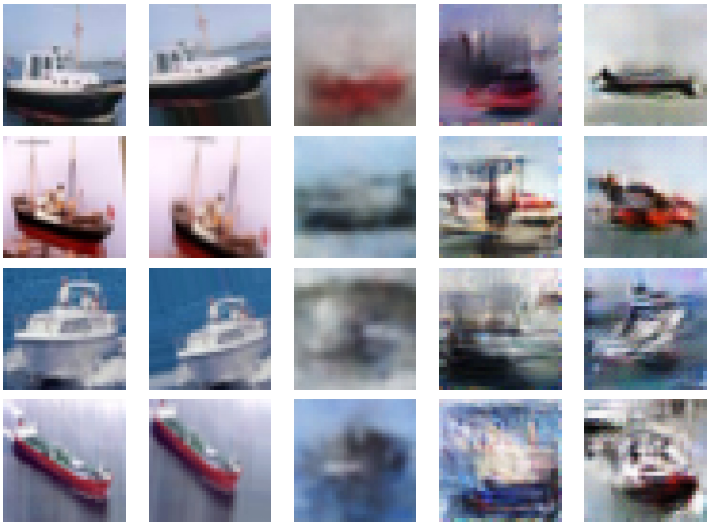
Results and discussion

This chapter presents the obtained results along with their possible interpretation. The main objective of the experiments was to analyse the quality of images generated by our reimplemented models, CVAE-GAN [12] and BAGAN [27], and compare them against original images and images produced using other methods.

The experimental results are of two types as follows. The first type surveys the diversity of groups of images produced by different methods. In our tables, the SSIM index [41] represents the diversity of images within one group. The obtained values for 50% imbalance scenario are compared in Table C.1, and for 75% imbalance scenario in Table C.2.

It is noteworthy that the diversity of geometrically transformed images is higher in almost every case than the diversity of original images. This phenomenon might be caused by creating a less homogenous dataset, especially when talking about MNIST [33] images. The handwritten digits are initially positioned into the centre of the images, so slight transformations such as rotations and shifts move them to more unusual locations. In CIFAR-10 [35] and Vehicles [36], the transformations have a lesser impact because the datasets themselves are more heterogeneous than the MNIST dataset.

The SSIM values also suggest a poor performance of CVAE [26] models. The results are not bad enough to suggest a mode collapse as the cause, but rather the CVAE architecture apparently was unable to discover many underlying features and incorporate them into the generated images. Besides, as can be seen in Figure 6.1, the images are significantly blurred. Using CVAE, we obtained worse results than with other methods except for one case, which was generating CIFAR-10 images. In this situation, the CVAE model was even able to outperform some more elaborate models.



(a) Original images (b) Geometric transformation (c) CVAE (d) CVAE-GAN (e) BAGAN

Figure 6.1: Visualisation comparison of original images (a), geometricly transformed images (b) and images generated by CVAE (c), CVAE-GAN (d) and BAGAN (e). The models were trained on data with 50% of the minority-class missing.

However, the diversity of images generated by CVAE-GAN is not excellent either. We faced many difficulties during building, training and tuning of the model, which likely reflected on its performance. Producing CIFAR-10 images was especially troublesome, likely caused by the dataset not being as homogenous as the other two datasets.

The BAGAN model performed surprisingly well for this metric. The diversity of generated images is close to the degree of the diversity of the original images. Moreover, the created images visually quite resemble the real ones, as illustrated in Figure 6.1. Indeed, the BAGAN images did not beat the images augmented by geometric transformations, which look almost indistinguishable from the original images.

The comparison of measurements of original images between Table C.1 and Table C.2 indicates that with an increasing number of images, the variety of the MNIST dataset samples decreases while the diversity of the CIFAR-10 dataset samples increases. As we already stated before, the CIFAR-10 dataset is more diverse than the MNIST dataset. The SSIM values from the Vehicles dataset remain almost the same for both imbalance scenarios.

The second evaluation type studies how the datasets balanced with augmented images impact the performance of classifiers trained on them. We summarised the results for each combination of dataset and imbalance scenario, which are provided in Tables C.3 to C.8.

As expected, the highest accuracy and F1 score values were measured on classifiers for the MNIST dataset. The values never dropped under 0.97, not even the F1 score of a classifier trained on an unbalanced dataset with 25% of minority-class images. It should be pointed out, that classifiers trained on datasets where their imbalance was restored using BAGAN performed almost as well as the classifiers trained on original data. The images created using other techniques were also of sufficient subjective quality, resembling the real samples.

The Vehicles classifiers also provided sound results, despite the fact that the dataset contains real-life photos. We interpret the cause of this to be the dataset having only three classes and the photos within one class being similar. Motorcycles and airplanes are mostly depicted from the side, the cars from the rear. Their surroundings are also usually consistent within a class, e.g. the airplanes usually have the sky or an airport as a background, and cars are mainly placed on a road.

The CVAE model, the simplest of the three types of generative models, generated blurred non-realistic images. The objects resembled true samples but looked like they were behind an opaque curtain or a frosted glass. It is important to note that even though the images of cars produced by CVAE are of low quality, the classifiers trained on datasets balanced with CVAE-GAN performed slightly worse. The synthesised images did not remind us of the photos of cars. Nevertheless, because the accuracy and F1 score were still close to the measurements of other methods, the necessary information is likely encoded in the generated images despite us being unable to see them. Once again, the BAGAN outperformed the two other generative models and provided us with plausible images.

The classification of CIFAR-10 images is a challenging task. Predictably, we were not as successful as with the other datasets. The classifier trained on original balanced data achieved accuracy of 0.7177, which is the highest accuracy that was measured. Furthermore, the minority-class F1 score achieved using this classifier is 0.8312, while the F1 score values measured with MNIST and Vehicles classifiers never dropped under 0.9333.

Not only classifying but also generating CIFAR-10 images is a difficult task. We spent much time training and tuning the generative models, even though we consider the produced images disappointing. We did not expect that CVAE

would give us plausible results, but it was especially CVAE-GAN, that we assumed would perform excellently. According to the paper, the model is supposed to learn and produce complex images with many details [12]. As illustrated in Figure 6.1, our results do not align with this statement.

Surprisingly, even though our results are not visually appealing, the measured values are higher than expected. However, the differences between the F1 score columns for CIFAR-10 in Tables C.5 and C.6 are more significant than in tables for other datasets. This might be caused by the models not being able to encompass and generalise all the important features, which are vital for the classifier's learning process.

Finally, we would like to summarise our empirical findings and what we experienced with the CVAE-GAN and BAGAN models. The designing, training and tuning of CVAE-GAN models was troublesome. The proposed network's parts were too intricate for us to replicate in this thesis. We had to tailor a similar, but simpler architecture for each dataset. Apart from that, it was hard to minimise the loss function consisting of six parts. The purpose of the proposed lambda weight values was to tune the emphasis for each of the loss functions. For instance, we had to find a balance between good reconstructions and a well structured latent space, both of which are needed to generate plausible images.

We consider the combination of CVAE-GAN and the CIFAR-10 dataset to be especially unfortunate. As we already pointed out, the dataset has colourful images divided into ten classes representing various real-life objects. The images vary greatly both inter-class and intra-class. We observed that CVAE-GAN has problems with synthesising images of datasets with high diversity. To confirm our suspicion, we verified that the datasets used in the original experiments were heavily preprocessed, to the point of being consistent with the spatial placement of all the crucial features.

One of the significant advantages of BAGAN is definitely the available code. Moreover, the published architecture was designed for MNIST and CIFAR-10, which suited our needs. Thanks to this fact, we did not have to improvise as much as with CVAE-GAN during the implementation. Still, we had to make some educated guesses. We struggled with finding a suitable dimension of latent vectors, which carry the encoded information, as we had to take in consideration the complexity and variance of the datasets.

During the training of these models, we encountered some problems such as mode collapse or the networks being unable to converge. These issues were mitigated using hyperparameter tuning or minor architectural changes.

Conclusion

The main focus of this thesis was data augmentation using generative adversarial networks. Specifically, we addressed the task of restoring balance in imbalanced datasets and surveyed up-to-date techniques that can be used to tackle this obstacle. We dived into the problem and applied two state-of-the-art methods, CVAE-GAN and BAGAN. Our contribution lies in adapting these techniques and comparing them on multiple datasets. In the following text, we would like to review the steps that we took to fulfil our goals and propose possible future work directions.

Contribution

As stated at the beginning of the thesis, we declared three main goals. This section discusses their fulfilment and reviews the work that was done.

The first goal was to introduce widely used data augmentation methods. In the first chapter of this thesis, we have made an introduction to this field, presenting both simple and advanced methods. Among the more straightforward techniques used to produce image samples, there are numerous types of geometric transformations. Some of the mentioned transformations were used in the experimental part of this thesis to obtain new images. Nevertheless, as a result of this thesis focusing on generative models and models derived from them, we put an emphasis on presenting the concepts of GAN and VAE.

The second stated objective involves the survey of state-of-the-art techniques and algorithms that might have the potential to be used to restore balance in imbalanced datasets. After the preceded study of currently used techniques, we examined the most relevant of them, which can be found in Chapter 2. For each technique, we gave a brief introduction of its objective.

Another goal that we established was to reimplement two of the introduced state-of-the-art GANs based on the descriptions presented in their research papers. The decision was made to implement the CVAE-GAN and BAGAN models. We presented their architecture, training process and their implementation details along with the changes we used to tailor the models to suit our needs in Chapters 3 and 4. We demonstrated their ability to generate images on various datasets and compared them with other data augmentation methods.

As expected, the most realistic images are mostly those augmented using geometric transformations. On the other hand, the images generated using CVAE are significantly blurred. Considering that CVAE was the simplest of the generative models that we used, it is not surprising. The images obtained using CVAE-GAN and BAGAN are of reasonable quality. However, CVAE-GAN requires more training and tuning.

Future work

Due to the temporal restrictions and computational limitations, many experiments are still left to be performed. The following section suggests several ideas that can be realised to extend this thesis.

There is a significant number of state-of-the-art methods appropriate for the balancing task, and we tested only two of them. We consider that some other models are also worth being examined and applied to balance our datasets. We also acknowledge that using only two metrics for the evaluation of such a complex task might not be sufficient. Addressing this issue, we suggest using more techniques to evaluate the applicability of the generated images, e.g. the Inception Score [43] or the Fréchet Inception Distance [44].

Even though we found images produced by CVAE-GAN slightly disappointing and its training troublesome, it is desirable to verify the results presented in its paper. Authors worked in a different setting, focusing on synthesising images of homogenous datasets with fine details. We suggest building a more robust architecture and training the model on a significantly pre-processed dataset with high-resolution images.

Unlike what we experienced with CVAE-GAN, designing and tuning BAGAN was more manageable. The results obtained from the evaluation metrics met our expectations, and the generated images were visually appealing. This finding is quite encouraging and can serve as an incentive for further research. Future investigations could experiment with modifications to the model's architecture and applying them to more challenging datasets.

It is generally known that training deep learning models on big data is time-consuming and requires high computational power. We hope for future work without time limitations that would allow encompassing data from the real world.

Bibliography

1. GOODFELLOW, Ian et al. Generative Adversarial Nets. In: GHAHRAMANI, Z.; WELLING, M.; CORTES, C.; LAWRENCE, N. D.; WEINBERGER, K. Q. (eds.). *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, pp. 2672–2680. Available also from: `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.
2. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep learning*. MIT press, 2016. ISBN 0262035618.
3. TANAKA, Fabio Henrique Kiyoiti dos Santos; ARANHA, Claus. Data augmentation using GANs. *arXiv preprint arXiv:1904.09135*. 2019. Available also from: `https://arxiv.org/abs/1904.09135`.
4. KINGMA, Diederik P; WELLING, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*. 2013. Available also from: `https://arxiv.org/abs/1312.6114`.
5. YEH, Raymond. *Variational Autoencoders and Generative Adversarial Network* [online]. 2016 [visited on 2020-03-03]. Available from: `https://courses.engr.illinois.edu/ece544na/fa2016/guest_lecture4.pdf`.
6. ABADI, Martín et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Available also from: `https://www.tensorflow.org/`.
7. CHOLLET, François et al. *Keras* [software]. 2015. Available also from: `https://keras.io`.
8. SHORTEN, Connor; KHOSHGOFTAAR, Taghi M. A survey on image data augmentation for deep learning. *Journal of Big Data*. 2019, vol. 6, no. 1, pp. 60. Available from DOI: `10.1186/s40537-019-0197-0`.

9. *NanoNets, Machine Learning API* [online] [visited on 2020-03-21]. Available from: https://nanonets.com/blog/data-augmentation-how-to-use-deeplearning-when-you-have-limited-data-part-2/.

10. CHAWLA, Nitesh V; BOWYER, Kevin W; HALL, Lawrence O; KEGELMEYER, W Philip. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*. 2002, vol. 16, pp. 321–357. Available from DOI: 10.1613/jair.953.

11. HE, Haibo; BAI, Yang; GARCIA, Edwardo A; LI, Shutao. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In: *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. 2008, pp. 1322–1328. Available from DOI: 10.1109/IJCNN.2008.4633969.

12. BAO, Jianmin; CHEN, Dong; WEN, Fang; LI, Houqiang; HUA, Gang. CVAE-GAN: fine-grained image generation through asymmetric training. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2745–2754. Available from DOI: 10.1109/ICCV.2017.299.

13. RADFORD, Alec; METZ, Luke; CHINTALA, Soumith. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In: BENGIO, Yoshua; LECUN, Yann (eds.). *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016. Available also from: http://arxiv.org/abs/1511.06434.

14. O'SHEA, Keiron; NASH, Ryan. An Introduction to Convolutional Neural Networks. *CoRR*. 2015, vol. abs/1511.08458. Available from arXiv: 1511.08458.

15. DOERSCH, Carl. Tutorial on Variational Autoencoders. *arXiv preprint arXiv:1606.05908*. 2016. Available also from: https://arxiv.org/abs/1606.05908.

16. KULLBACK, Solomon; LEIBLER, Richard A. On information and sufficiency. *The annals of mathematical statistics*. 1951, vol. 22, no. 1, pp. 79–86. Available from DOI: 10.1214/aoms/1177729694.

17. GARAY-MAESTRE, Unai; GALLEGO, Antonio-Javier; CALVO-ZARAGOZA, Jorge. Data Augmentation via Variational Auto-Encoders. In: *Iberoamerican Congress on Pattern Recognition*. 2018, pp. 29–37. Available from DOI: 10.1007/978-3-030-13469-3_4.

18. SMOLENSKY, P. Information Processing in Dynamical Systems: Foundations of Harmony Theory. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 194–281. ISBN 026268053X.

19. HINTON, Geoffrey E; SEJNOWSKI, Terrence J; ACKLEY, David H. *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, 1984. Available also from: `http://www.csri.utoronto.ca/~hinton/absps/bmtr.pdf`.

20. MAROÑAS, Juan; PAREDES, Roberto; RAMOS, Daniel. Generative Models For Deep Learning with Very Scarce Data. In: *Iberoamerican Congress on Pattern Recognition*. 2018, pp. 20–28. Available from DOI: `10.1007/978-3-030-13469-3_4`.

21. WHITE, Tom. Sampling Generative Networks: Notes on a Few Effective Techniques. *CoRR*. 2016, vol. abs/1609.04468. Available from arXiv: `1609.04468`.

22. ISOLA, Phillip; ZHU, Jun-Yan; ZHOU, Tinghui; EFROS, Alexei A. Image-to-Image Translation with Conditional Adversarial Networks. *CoRR*. 2016, vol. abs/1611.07004. Available from arXiv: `1611.07004`.

23. MIRZA, Mehdi; OSINDERO, Simon. Conditional Generative Adversarial Nets. *CoRR*. 2014, vol. abs/1411.1784. Available from arXiv: `1411.1784`.

24. ZHU, Jun-Yan; PARK, Taesung; ISOLA, Phillip; EFROS, Alexei A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *CoRR*. 2017, vol. abs/1703.10593. Available from arXiv: `1703.10593`.

25. ANTONIOU, Antreas; STORKEY, Amos; EDWARDS, Harrison. *Data Augmentation Generative Adversarial Networks*. 2017. Available from arXiv: `1711.04340`.

26. SOHN, Kihyuk; LEE, Honglak; YAN, Xinchen. Learning Structured Output Representation using Deep Conditional Generative Models. In: *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 3483–3491. Available also from: `http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf`.

27. MARIANI, Giovanni; SCHEIDEGGER, Florian; ISTRATE, Roxana; BEKAS, Costas; MALOSSI, A. Cristiano I. BAGAN: Data Augmentation with Balancing GAN. *CoRR*. 2018, vol. abs/1803.09655. Available from arXiv: `1803.09655`.

28. WANG, Yaxing et al. Transferring GANs: generating images from limited data. *CoRR*. 2018, vol. abs/1805.01677. Available from arXiv: `1805.01677`.

29. MURPHY, Kevin P. *Machine learning: a probabilistic perspective*. MIT press, 2012. ISBN 0262018020.

30. ODENA, Augustus; OLAH, Christopher; SHLENS, Jonathon. Conditional Image Synthesis with Auxiliary Classifier GANs. In: PRECUP, Doina; TEH, Yee Whye (eds.). *Proceedings of the 34th International Conference on Machine Learning*. International Convention Centre, Sydney, Australia: PMLR, 2017, vol. 70, pp. 2642–2651. Proceedings of Machine Learning Research. Available also from: `http://proceedings.mlr.press/v70/odena17a.html`.

31. SZEGEDY, C. et al. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. Available from DOI: `10.1109/CVPR.2015.7298594`.

32. SIMONYAN, Karen; ZISSERMAN, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: BENGIO, Yoshua; LECUN, Yann (eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. Available also from: `http://arxiv.org/abs/1409.1556`.

33. LECUN, Yann; CORTES, Corinna. MNIST handwritten digit database. 2010. Available also from: `http://yann.lecun.com/exdb/mnist/`.

34. MAAS, Andrew L; HANNUN, Awni Y; NG, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In: *Proc. icml*. 2013, vol. 30, p. 3. No. 1. Available also from: `https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf`.

35. KRIZHEVSKY, Alex; HINTON, Geoffrey, et al. Learning multiple layers of features from tiny images. *University of Toronto*. 2009. Available also from: `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

36. *Computational Vision: Archive* [online] [visited on 2020-04-05]. Available from: `http://www.vision.caltech.edu/html-files/archive.html`.

37. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105. Available from DOI: `10.1145/3065386`.

38. KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization. In: BENGIO, Yoshua; LECUN, Yann (eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. Available also from: `http://arxiv.org/abs/1412.6980`.

39. TIELEMAN, Tijmen; HINTON, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*. 2012, vol. 4, no. 2, pp. 26–31. Available also from: `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

40. XU, Bing; WANG, Naiyan; CHEN, Tianqi; LI, Mu. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR*. 2015, vol. abs/1505.00853. Available from arXiv: `1505.00853`.

41. WANG, Zhou; BOVIK, Alan C; SHEIKH, Hamid R; SIMONCELLI, Eero P. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*. 2004, vol. 13, no. 4, pp. 600–612. Available from DOI: `10.1109/TIP.2003.819861`.

42. WANG, Zhou; BOVIK, Alan C; SIMONCELLI, Eero P. Structural approaches to image quality assessment. *Handbook of Image and Video Processing*. 2005, vol. 7, pp. 18. Available from DOI: `10.1016/B978-012119792-6/50119-4`.

43. BARRATT, Shane; SHARMA, Rishi. *A Note on the Inception Score*. 2018. Available from arXiv: `1801.01973`.

44. HEUSEL, Martin; RAMSAUER, Hubert; UNTERTHINER, Thomas; NESSLER, Bernhard; HOCHREITER, Sepp. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In: GUYON, I. et al. (eds.). *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 6626–6637. Available also from: `http://papers.nips.cc/paper/7240-gans-trained-by-a-two-time-scale-update-rule-converge-to-a-local-nash-equilibrium.pdf`.

Appendix A

Acronyms

ADASYN Adaptive synthetic sampling approach

BAGAN Balancing generative adversarial network

BM Boltzmann machine

CNN Convolutional neural network

CVAE Conditional variational autoencoder

CycleGAN Cycle consistent generative adversarial network

DAGAN Data augmentation generative adversarial network

DCGAN Deep convolutional generative adversarial network

FID Fréchet Inception Distance

GAN Generative adversarial network

ILSVRC ImageNet Large Scale Visual Recognition Competition

IS Inception Score

Leaky ReLU Leaky rectified linear unit

MINE Manifold interpolated neighbour embedding

ReLU Rectified linear unit

RBM Restricted Boltzmann machine

SMOTE Synthetic minority over-sampling technique

SSIM Structural similarity

VAE Variational autoencoder

Appendix B

Network architecture

The following tables are generated using the code from the publicly available GitHub repository[7].

Table B.1: CVAE-GAN generator G for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_3 (InputLayer)	(1)		0	
1	embedding_1 (Embedding)	(1, 55)		550	input_3
2	reshape_2 (Reshape)	(55,)		0	embedding_1
3	input_4 (InputLayer)	(55)		0	
4	concatenate_1 (Concatenate)	(110,)	**Axis:** -1	0	reshape_2, input_4
5	flatten_1 (Flatten)	(110,)		0	concatenate_1
6	dense_2 (Dense)	(3136,)	**#Neurons:** 3136 **Activation:** linear	348 096	flatten_1
7	re_lu (ReLU)	(3136,)	**Activation:** relu	0	dense_2
8	reshape_3 (Reshape)	(7, 7, 64)		0	re_lu
9	conv2d_transpose (Conv2DTranspose)	(7, 7, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	36 928	reshape_3
10	re_lu_1 (ReLU)	(7, 7, 64)	**Activation:** relu	0	conv2d_transpose
11	conv2d_transpose_1 (Conv2DTranspose)	(14, 14, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	18 464	re_lu_1
12	re_lu_2 (ReLU)	(14, 14, 32)	**Activation:** relu	0	conv2d_transpose_1
13	conv2d_transpose_2 (Conv2DTranspose)	(28, 28, 16)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	4624	re_lu_2
14	re_lu_3 (ReLU)	(28, 28, 16)	**Activation:** relu	0	conv2d_transpose_2
15	conv2d_transpose_3 (Conv2DTranspose)	(28, 28, 1)	**Activation:** tanh **Kernel Size:** [5, 5] **Stride:** [1, 1]	401	re_lu_3

[7] `https://github.com/fablukm/keras-reports`

Table B.2: CVAE-GAN dicsriminator D for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_5 (InputLayer)	(28, 28, 1)		0	
1	conv2d_7 (Conv2D)	(13, 13, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	640	input_5
2	leaky_re_lu_7 (LeakyReLU)	(13, 13, 64)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_7
3	batch_normalization_7 (BatchNormalization)	(13, 13, 64)		256	leaky_re_lu_7
4	conv2d_8 (Conv2D)	(6, 6, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	73 856	batch_normalization_7
5	leaky_re_lu_8 (LeakyReLU)	(6, 6, 128)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_8
6	batch_normalization_8 (BatchNormalization)	(6, 6, 128)		512	leaky_re_lu_8
7	conv2d_9 (Conv2D)	(2, 2, 256)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	295 168	batch_normalization_8
8	leaky_re_lu_9 (LeakyReLU)	(2, 2, 256)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_9
9	batch_normalization_9 (BatchNormalization)	(2, 2, 256)		1024	leaky_re_lu_9
10	max_pooling2d_3 (MaxPooling2D)	(1, 1, 256)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	batch_normalization_9
11	flatten_2 (Flatten)	(256,)		0	max_pooling2d_3
12	dropout (Dropout)	(256,)	**Dropout Rate:** 0.3	0	flatten_2
13	dense_3 (Dense)	(1,)	**#Neurons:** 1 **Activation:** sigmoid	257	dropout

Table B.3: CVAE-GAN classifier C for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_6 (InputLayer)	(28, 28, 1)		0	
1	conv2d_10 (Conv2D)	(26, 26, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	320	input_6
2	re_lu_4 (ReLU)	(26, 26, 32)	**Activation:** relu	0	conv2d_10
3	batch_normalization_10 (BatchNormalization)	(26, 26, 32)		128	re_lu_4
4	conv2d_11 (Conv2D)	(24, 24, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	9248	batch_normalization_10
5	re_lu_5 (ReLU)	(24, 24, 32)	**Activation:** relu	0	conv2d_11
6	batch_normalization_11 (BatchNormalization)	(24, 24, 32)		128	re_lu_5
7	max_pooling2d_4 (MaxPooling2D)	(12, 12, 32)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	batch_normalization_11
8	conv2d_12 (Conv2D)	(10, 10, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	18 496	max_pooling2d_4
9	re_lu_6 (ReLU)	(10, 10, 64)	**Activation:** relu	0	conv2d_12
10	batch_normalization_12 (BatchNormalization)	(10, 10, 64)		256	re_lu_6
11	conv2d_13 (Conv2D)	(8, 8, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	36 928	batch_normalization_12
12	re_lu_7 (ReLU)	(8, 8, 64)	**Activation:** relu	0	conv2d_13
13	batch_normalization_13 (BatchNormalization)	(8, 8, 64)		256	re_lu_7
14	max_pooling2d_5 (MaxPooling2D)	(4, 4, 64)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	batch_normalization_13
15	conv2d_14 (Conv2D)	(2, 2, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	73 856	max_pooling2d_5
16	re_lu_8 (ReLU)	(2, 2, 128)	**Activation:** relu	0	conv2d_14
17	batch_normalization_14 (BatchNormalization)	(2, 2, 128)		512	re_lu_8
18	conv2d_15 (Conv2D)	(2, 2, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	147 584	batch_normalization_14
19	re_lu_9 (ReLU)	(2, 2, 128)	**Activation:** relu	0	conv2d_15
20	batch_normalization_15 (BatchNormalization)	(2, 2, 128)		512	re_lu_9

Table B.3 Continued: CVAE-GAN classifier C for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
21	conv2d_16 (Conv2D)	(2, 2, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	147 584	batch_normalization_15
22	re_lu_10 (ReLU)	(2, 2, 128)	**Activation:** relu	0	conv2d_16
23	batch_normalization_16 (BatchNormalization)	(2, 2, 128)		512	re_lu_10
24	max_pooling2d_6 (MaxPooling2D)	(1, 1, 128)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	batch_normalization_16
25	flatten_3 (Flatten)	(128,)		0	max_pooling2d_6
26	dropout_1 (Dropout)	(128,)	**Dropout Rate:** 0.3	0	flatten_3
27	dense_4 (Dense)	(10,)	**#Neurons:** 10 **Activation:** linear	1290	dropout_1
	activation (Activation)	(10,)	**Activation:** softmax	0	dense_4

Table B.4: CVAE-GAN encoder E for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_1 (InputLayer)	(28, 28, 1)		0	
1	conv2d (Conv2D)	(26, 26, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	320	input_1
2	leaky_re_lu (LeakyReLU)	(26, 26, 32)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d
3	batch_normalization (BatchNormalization)	(26, 26, 32)		128	leaky_re_lu
4	conv2d_1 (Conv2D)	(24, 24, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	9248	batch_normalization
5	leaky_re_lu_1 (LeakyReLU)	(24, 24, 32)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_1
6	batch_normalization_1 (BatchNormalization)	(24, 24, 32)		128	leaky_re_lu_1
7	max_pooling2d (MaxPooling2D)	(12, 12, 32)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	batch_normalization_1
8	conv2d_2 (Conv2D)	(10, 10, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	18 496	max_pooling2d
9	leaky_re_lu_2 (LeakyReLU)	(10, 10, 64)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_2
10	batch_normalization_2 (BatchNormalization)	(10, 10, 64)		256	leaky_re_lu_2
11	conv2d_3 (Conv2D)	(8, 8, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	36 928	batch_normalization_2
12	leaky_re_lu_3 (LeakyReLU)	(8, 8, 64)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_3
13	batch_normalization_3 (BatchNormalization)	(8, 8, 64)		256	leaky_re_lu_3
14	max_pooling2d_1 (MaxPooling2D)	(4, 4, 64)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	batch_normalization_3
15	conv2d_4 (Conv2D)	(2, 2, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	73 856	max_pooling2d_1
16	leaky_re_lu_4 (LeakyReLU)	(2, 2, 128)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_4
17	batch_normalization_4 (BatchNormalization)	(2, 2, 128)		512	leaky_re_lu_4

Table B.4 Continued: CVAE-GAN encoder E for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
18	conv2d_5 (Conv2D)	(2, 2, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	147 584	batch_normalization_4
19	leaky_re_lu_5 (LeakyReLU)	(2, 2, 128)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_5
20	batch_normalization_5 (BatchNormalization)	(2, 2, 128)		512	leaky_re_lu_5
21	conv2d_6 (Conv2D)	(2, 2, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	147 584	batch_normalization_5
22	leaky_re_lu_6 (LeakyReLU)	(2, 2, 128)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_6
23	input_2 (InputLayer)	(1)		0	
24	batch_normalization_6 (BatchNormalization)	(2, 2, 128)		512	leaky_re_lu_6
25	embedding (Embedding)	(1, 128)		1280	input_2
26	max_pooling2d_2 (MaxPooling2D)	(1, 1, 128)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	batch_normalization_6
27	reshape (Reshape)	(128, 1)		0	embedding
28	reshape_1 (Reshape)	(128, 1)		0	max_pooling2d_2
29	concatenate (Concatenate)	(128, 2)	**Axis:** -1	0	reshape, reshape_1
30	flatten (Flatten)	(256,)		0	concatenate
31	dense (Dense)	(55,)	**#Neurons:** 55 **Activation:** linear	14 135	flatten
32	dense_1 (Dense)	(55,)	**#Neurons:** 55 **Activation:** linear	14 135	flatten

Table B.5: BAGAN generator G and decoder Δ for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_1 (InputLayer)	(10)		0	
1	dense (Dense)	(3136,)	**#Neurons:** 3136 **Activation:** linear	34 496	input_1
2	re_lu (ReLU)	(3136,)	**Activation:** relu	0	dense
3	reshape (Reshape)	(7, 7, 64)		0	re_lu
4	conv2d_transpose (Conv2DTranspose)	(7, 7, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [1, 1]	36 928	reshape
5	re_lu_1 (ReLU)	(7, 7, 64)	**Activation:** relu	0	conv2d_transpose
6	conv2d_transpose_1 (Conv2DTranspose)	(14, 14, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	18 464	re_lu_1
7	re_lu_2 (ReLU)	(14, 14, 32)	**Activation:** relu	0	conv2d_transpose_1
8	conv2d_transpose_2 (Conv2DTranspose)	(28, 28, 16)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	4624	re_lu_2
9	re_lu_3 (ReLU)	(28, 28, 16)	**Activation:** relu	0	conv2d_transpose_2
10	conv2d_transpose_3 (Conv2DTranspose)	(28, 28, 1)	**Activation:** tanh **Kernel Size:** [5, 5] **Stride:** [1, 1]	401	re_lu_3

Table B.6: BAGAN discriminator D for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_2 (InputLayer)	(28, 28, 1)		0	
1	conv2d_4 (Conv2D)	(14, 14, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	320	input_2
2	leaky_re_lu_4 (LeakyReLU)	(14, 14, 32)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_4
3	dropout_4 (Dropout)	(14, 14, 32)	**Dropout Rate:** 0.3	0	leaky_re_lu_4
4	conv2d_5 (Conv2D)	(7, 7, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	18 496	dropout_4
5	leaky_re_lu_5 (LeakyReLU)	(7, 7, 64)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_5
6	dropout_5 (Dropout)	(7, 7, 64)	**Dropout Rate:** 0.3	0	leaky_re_lu_5
7	conv2d_6 (Conv2D)	(4, 4, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	73 856	dropout_5
8	leaky_re_lu_6 (LeakyReLU)	(4, 4, 128)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_6
9	dropout_6 (Dropout)	(4, 4, 128)	**Dropout Rate:** 0.3	0	leaky_re_lu_6
10	conv2d_7 (Conv2D)	(2, 2, 256)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	295 168	dropout_6
11	leaky_re_lu_7 (LeakyReLU)	(2, 2, 256)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_7
12	dropout_7 (Dropout)	(2, 2, 256)	**Dropout Rate:** 0.3	0	leaky_re_lu_7
13	flatten_1 (Flatten)	(1024,)		0	dropout_7
14	dense_1 (Dense)	(11,)	**#Neurons:** 11 **Activation:** sigmoid	11 275	flatten_1

Table B.7: BAGAN encoder E for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_1 (InputLayer)	(28, 28, 1)		0	
1	conv2d (Conv2D)	(14, 14, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	320	input_1
2	leaky_re_lu (LeakyReLU)	(14, 14, 32)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d
3	dropout (Dropout)	(14, 14, 32)	**Dropout Rate:** 0.3	0	leaky_re_lu
4	conv2d_1 (Conv2D)	(7, 7, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	18 496	dropout
5	leaky_re_lu_1 (LeakyReLU)	(7, 7, 64)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_1
6	dropout_1 (Dropout)	(7, 7, 64)	**Dropout Rate:** 0.3	0	leaky_re_lu_1
7	conv2d_2 (Conv2D)	(4, 4, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	73 856	dropout_1
8	leaky_re_lu_2 (LeakyReLU)	(4, 4, 128)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_2
9	dropout_2 (Dropout)	(4, 4, 128)	**Dropout Rate:** 0.3	0	leaky_re_lu_2
10	conv2d_3 (Conv2D)	(2, 2, 256)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	295 168	dropout_2
11	leaky_re_lu_3 (LeakyReLU)	(2, 2, 256)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_3
12	dropout_3 (Dropout)	(2, 2, 256)	**Dropout Rate:** 0.3	0	leaky_re_lu_3
13	flatten (Flatten)	(1024,)		0	dropout_3
14	dense (Dense)	(10,)	**#Neurons:** 10 **Activation:** linear	10 250	flatten

Table B.8: Evaluation classifier for MNIST dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_2 (InputLayer)	(28, 28, 1)		0	
1	conv2d_3 (Conv2D)	(14, 14, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	320	input_2
2	re_lu_4 (ReLU)	(14, 14, 32)	**Activation:** relu	0	conv2d_3
3	conv2d_4 (Conv2D)	(7, 7, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	18 496	re_lu_4
4	re_lu_5 (ReLU)	(7, 7, 64)	**Activation:** relu	0	conv2d_4
5	conv2d_5 (Conv2D)	(4, 4, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	36 928	re_lu_5
6	re_lu_6 (ReLU)	(4, 4, 64)	**Activation:** relu	0	conv2d_5
7	flatten_1 (Flatten)	(1024,)		0	re_lu_6
8	dense_2 (Dense)	(64,)	**#Neurons:** 64 **Activation:** linear	65 600	flatten_1
9	re_lu_7 (ReLU)	(64,)	**Activation:** relu	0	dense_2
10	dense_3 (Dense)	(10,)	**#Neurons:** 10 **Activation:** linear	650	re_lu_7
	activation_1 (Activation)	(10,)	**Activation:** softmax	0	dense_3

Table B.9: Evaluation classifier for Vehicles dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_2 (InputLayer)	(64, 64, 3)		0	
1	conv2d_4 (Conv2D)	(32, 32, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	896	input_2
2	re_lu_5 (ReLU)	(32, 32, 32)	**Activation:** relu	0	conv2d_4
3	conv2d_5 (Conv2D)	(16, 16, 32)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	9248	re_lu_5
4	re_lu_6 (ReLU)	(16, 16, 32)	**Activation:** relu	0	conv2d_5
5	conv2d_6 (Conv2D)	(8, 8, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	18 496	re_lu_6
6	re_lu_7 (ReLU)	(8, 8, 64)	**Activation:** relu	0	conv2d_6
7	conv2d_7 (Conv2D)	(4, 4, 64)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	36 928	re_lu_7
8	re_lu_8 (ReLU)	(4, 4, 64)	**Activation:** relu	0	conv2d_7
9	flatten_1 (Flatten)	(1024,)		0	re_lu_8
10	dense_2 (Dense)	(64,)	**#Neurons:** 64 **Activation:** linear	65 600	flatten_1
11	re_lu_9 (ReLU)	(64,)	**Activation:** relu	0	dense_2
12	dense_3 (Dense)	(10,)	**#Neurons:** 10 **Activation:** linear	650	re_lu_9
	activation_1 (Activation)	(10,)	**Activation:** softmax	0	dense_3

Table B.10: Evaluation classifier for CIFAR-10 dataset.

№	Layer (Type)	Output shape	Config	#Parameters	Inbound layers
0	input_2 (InputLayer)	(32, 32, 3)		0	
1	conv2d_4 (Conv2D)	(16, 16, 128)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	3584	input_2
2	leaky_re_lu_4 (LeakyReLU)	(16, 16, 128)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_4
3	batch_normalization_4 (BatchNormalization)	(16, 16, 128)		512	leaky_re_lu_4
4	dropout_4 (Dropout)	(16, 16, 128)	**Dropout Rate:** 0.3	0	batch_normalization_4
5	conv2d_5 (Conv2D)	(8, 8, 256)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	295 168	dropout_4
6	leaky_re_lu_5 (LeakyReLU)	(8, 8, 256)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_5
7	batch_normalization_5 (BatchNormalization)	(8, 8, 256)		1024	leaky_re_lu_5
8	dropout_5 (Dropout)	(8, 8, 256)	**Dropout Rate:** 0.3	0	batch_normalization_5
9	max_pooling2d_1 (MaxPooling2D)	(4, 4, 256)	**Pool size:** [2, 2] **Strides:** [2, 2]	0	dropout_5
10	conv2d_6 (Conv2D)	(2, 2, 256)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	590 080	max_pooling2d_1
11	leaky_re_lu_6 (LeakyReLU)	(2, 2, 256)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_6
12	batch_normalization_6 (BatchNormalization)	(2, 2, 256)		1024	leaky_re_lu_6
13	dropout_6 (Dropout)	(2, 2, 256)	**Dropout Rate:** 0.3	0	batch_normalization_6
14	conv2d_7 (Conv2D)	(1, 1, 512)	**Activation:** linear **Kernel Size:** [3, 3] **Stride:** [2, 2]	1 180 160	dropout_6
15	leaky_re_lu_7 (LeakyReLU)	(1, 1, 512)	**Activation:** leakyrelu **Alpha:** 0.3	0	conv2d_7
16	batch_normalization_7 (BatchNormalization)	(1, 1, 512)		2048	leaky_re_lu_7
17	dropout_7 (Dropout)	(1, 1, 512)	**Dropout Rate:** 0.3	0	batch_normalization_7
18	flatten_1 (Flatten)	(512,)		0	dropout_7
19	dense_2 (Dense)	(64,)	**#Neurons:** 64 **Activation:** linear	32 832	flatten_1
20	re_lu_1 (ReLU)	(64,)	**Activation:** relu	0	dense_2
21	dense_3 (Dense)	(10,)	**#Neurons:** 10 **Activation:** linear	650	re_lu_1
	activation_1 (Activation)	(10,)	**Activation:** softmax	0	dense_3

Appendix C

Experimental results

Table C.1: Measurement of SSIM of original and generated minority-class image couples produced by models trained on datasets with 50% imbalance.

Dataset	Original images	Generated images using			
		geom. trans.	CVAE	CVAE-GAN	BAGAN
MNIST	0.1678	0.1265	0.2350	0.2058	0.1727
CIFAR-10	0.0322	0.0275	0.0380	0.0746	0.0343
Vehicles	0.1079	0.0860	0.3632	0.2030	0.1627

Table C.2: Measurement of SSIM of original and generated minority-class image couples produced by models trained on datasets with 75% imbalance.

Dataset	Original images	Generated images using			
		geom. trans.	CVAE	CVAE-GAN	BAGAN
MNIST	0.1744	0.1255	0.2131	0.2091	0.1686
CIFAR-10	0.0278	0.0318	0.0394	0.0762	0.0395
Vehicles	0.1031	0.0909	0.3499	0.2041	0.1510

Table C.3: Evaluation of the dataset having 50% of minority-class images dropped using the classifier network presented in Table B.8.

MNIST			
Dataset type		Test data	
		Accuracy	F1 score
Unbalanced original		0.9845	0.9813
Balanced original		0.9860	0.9858
Balanced with	geom. trans.	0.9839	0.9731
	CVAE	0.9834	0.9774
	CVAE-GAN	0.9846	0.9833
	BAGAN	0.9830	0.9842

Table C.4: Evaluation of the dataset having 75% of minority-class images dropped using the classifier network presented in Table B.8.

MNIST			
Dataset type		Test data	
		Accuracy	F1 score
Unbalanced original		0.9820	0.9793
Balanced original		0.9860	0.9858
Balanced with	geom. trans.	0.9833	0.9789
	CVAE	0.9819	0.9735
	CVAE-GAN	0.9831	0.9745
	BAGAN	0.9842	0.9806

Table C.5: Evaluation of the dataset having 50% of minority-class images dropped using the classifier network presented in Table B.10.

CIFAR-10			
Dataset type		Test data	
		Accuracy	F1 score
Unbalanced original		0.7074	0.8021
Balanced original		0.7177	0.8312
Balanced with	geom. trans.	0.6974	0.8035
	CVAE	0.6892	0.7797
	CVAE-GAN	0.6866	0.7823
	BAGAN	0.7039	0.8043

Table C.6: Evaluation of the dataset having 75% of minority-class images dropped using the classifier network presented in Table B.10.

CIFAR-10			
Dataset type		Test data	
		Accuracy	F1 score
Unbalanced original		0.7023	0.7484
Balanced original		0.7177	0.8312
Balanced with	geom. trans.	0.7175	0.7936
	CVAE	0.6966	0.7544
	CVAE-GAN	0.6882	0.7539
	BAGAN	0.7054	0.7547

Table C.7: Evaluation of the dataset having 50% of minority-class images dropped using the classifier network presented in Table B.9.

Vehicles			
Dataset type		Test data	
		Accuracy	F1 score
Unbalanced original		0.9661	0.9491
Balanced original		0.9726	0.9561
Balanced with	geom. trans.	0.9817	0.9741
	CVAE	0.9739	0.9574
	CVAE-GAN	0.9700	0.9547
	BAGAN	0.9765	0.9632

Table C.8: Evaluation of the dataset having 75% of minority-class images dropped using the classifier network presented in Table B.9.

Vehicles			
Dataset type		Test data	
		Accuracy	F1 score
Unbalanced original		0.9661	0.9457
Balanced original		0.9726	0.9561
Balanced with	geom. trans.	0.9621	0.9482
	CVAE	0.9661	0.9496
	CVAE-GAN	0.9569	0.9333
	BAGAN	0.9687	0.9556

Appendix D

Contents of enclosed SD card

- `readme.txt` ..... the file with SD card contents description
- `src` ..... the directory of source codes
  - `evaluation` ..... the directory of evaluated methods
  - `implementation` ..... the directory of implemented models
  - `vehicles` ..... the directory of Vehicles dataset
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format