JYX JYVÄSKYLÄN YLIOPISTO UNIVERSITY OF JYVÄSKYLÄ

This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.

Author(s): Terziyan, Vagan; Tiihonen, Timo

Title: Using Cloning-GAN Architecture to Unlock the Secrets of Smart Manufacturing : Replication of Cognitive Models

Year: 2024

Version: Published version

Copyright: © 2024 the Authors

Rights: CC BY-NC-ND 4.0

Rights url: https://creativecommons.org/licenses/by-nc-nd/4.0/

Please cite the original version:

Terziyan, V., & Tiihonen, T. (2024). Using Cloning-GAN Architecture to Unlock the Secrets of Smart Manufacturing : Replication of Cognitive Models. Procedia Computer Science, 232, 890-902. https://doi.org/10.1016/j.procs.2024.01.089

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 232 (2024) 890–902

Procedia Computer Science

www.elsevier.com/locate/procedia

5th International Conference on Industry 4.0 and Smart Manufacturing

Using Cloning-GAN Architecture to Unlock the Secrets of Smart Manufacturing: Replication of Cognitive Models

Vagan Terziyan [a],*, Timo Tiihonen [a]

[a] *Faculty of Information Technology, University of Jyväskylä, 40014, Jyväskylä, Finland*

Abstract

As Industry 4.0 and 5.0 evolve to be highly automated but human-centric, there is a need for process modeling based on digital replicas of physical objects including humans. Knowledge distillation and cognitive cloning offer a way to train operational copies of decision-making black boxes, or donors, without requiring additional data. In this paper, we propose an architecture and analytics for a generative adversarial network, called Cloning-GAN, which enables donor-clone knowledge transfer, including the donor's individual biases. The architecture involves generating challenging samples to be labeled by the donor and used as training data for the clone. We consider several multicriteria requirements for the generated data, including closeness to the decision boundary, uniform distribution in the decision space, maximal confusion for the donor, and challenge for the clone. We present various strategies to balance these conflicting criteria forcing the clone learning quickly the hidden cognitive skills and biases of the donor.

© 2024 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 5th International Conference on Industry 4.0 and Smart Manufacturing

Keywords: Smart Manufacturing; digital twins; cognitive clones; knowledge transfer; knowledge distillation; Generative Adversarial Network; adversarial distillation

1. Introduction

Modelling and simulation enhanced with emergent technologies is a key for efficient Industry 4.0 and beyond, including smart manufacturing [1]. In addition to digital twins [2] simulating a physical entity or operation, we can observe a strong trend towards human-centric cyber-physical production systems [3] driven by mental models and smart operators [4] aiming at human values [5], including digital cognitive clones of humans [6] and groups [7].

* Corresponding author. *E-mail address:* vagan.terziyan@jyu.fi

1877-0509 © 2024 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 5th International Conference on Industry 4.0 and Smart Manufacturing
10.1016/j.procs.2024.01.089

Emergent development of such smart assets results in significant benefits for smart manufacturing, making it more efficient, safer, and capable of producing high-quality products. Assisting technologies to enable digital replica of industrial reality (e.g., in the form of smart models, twins, clones, etc.) include machine (deep) learning [8], knowledge transfer [9], knowledge distillation [10], particularly adversarial distillation [11] among others.

In this paper, we introduce an adversarial cloning architecture capable of making digital replicas of hidden cognitive assets with respect to their individual biases. The architecture is supposed to be a kind of adversarial distillation driven by a generative adversarial network with a special configuration and objectives regarding the generator and discriminator networks. It facilitates supervised machine learning of the clone model due to a smart way to generate challenging inputs for the donor (the target smart and black-box asset for the clone), who is supposed to act as a supervisor in such an adversarial learning process. The multicriteria-driven generator (adversarial facilitator) is the key cloning enabler and the major innovation in the suggested architecture.

The following text of the paper is organized as follows: in Section 2, we present the main contribution of the paper, i.e., the intended cloning architecture; in Section 3, we present the related work; in Section 4 we discuss the added value of this study compared to our previous studies regarding cloning architecture; and we conclude in Section 5.

2. Introduction to Cloning-GAN

In this section, we are going to introduce a new architecture capable of unlocking and replication of hidden and smart industrial assets, including human decision-makers and other digital, cognitive and data-driven (e.g., machine learning based) models.

2.1. DONOR, cloning, and CLONE

Assume we have a cognitive model aka black box (abstract, physical, cyber, social, etc.) named as **DONOR** and represented by a hidden ***"labeling function"*** $\boldsymbol{\mathcal{F}}: [0,1]^n \longmapsto \Delta^c$ defined as follows:

(a) The domain of $\boldsymbol{\mathcal{F}}$ is defined by an Euclidean space $\mathbb{R}^n$ bounded by a $\boldsymbol{n}$-dimensional unit hypercube: $[0,1]^n$, each i-th point of which is an $\boldsymbol{n}$-dimensional vector $\vec{x}_i$ of rational values, aka coordinates of some object of potential labeling:

$$[0,1]^n = \{\forall \vec{x}_i: (x_1, x_2, \dots, x_n) \in \mathbb{R}^n | x_1 \in [0,1], x_2 \in [0,1], \dots, x_n \in [0,1]\};$$

(b) The range of $\boldsymbol{\mathcal{F}}$ is defined by the unit $\boldsymbol{c}$-simplex Δ^c (i.e., the $\boldsymbol{c}$-dimensional probability simplex), each i-th point of which is a $\boldsymbol{c}$-dimensional vector $\vec{p}_i$ of probabilities [regarding each of $\boldsymbol{c}$ possible class labels from the set $\mathbb{L} = \{l_1, l_2, \dots, l_c\}$ assigned to the corresponding object represented by point $\vec{x}_i$ within the domain space]:

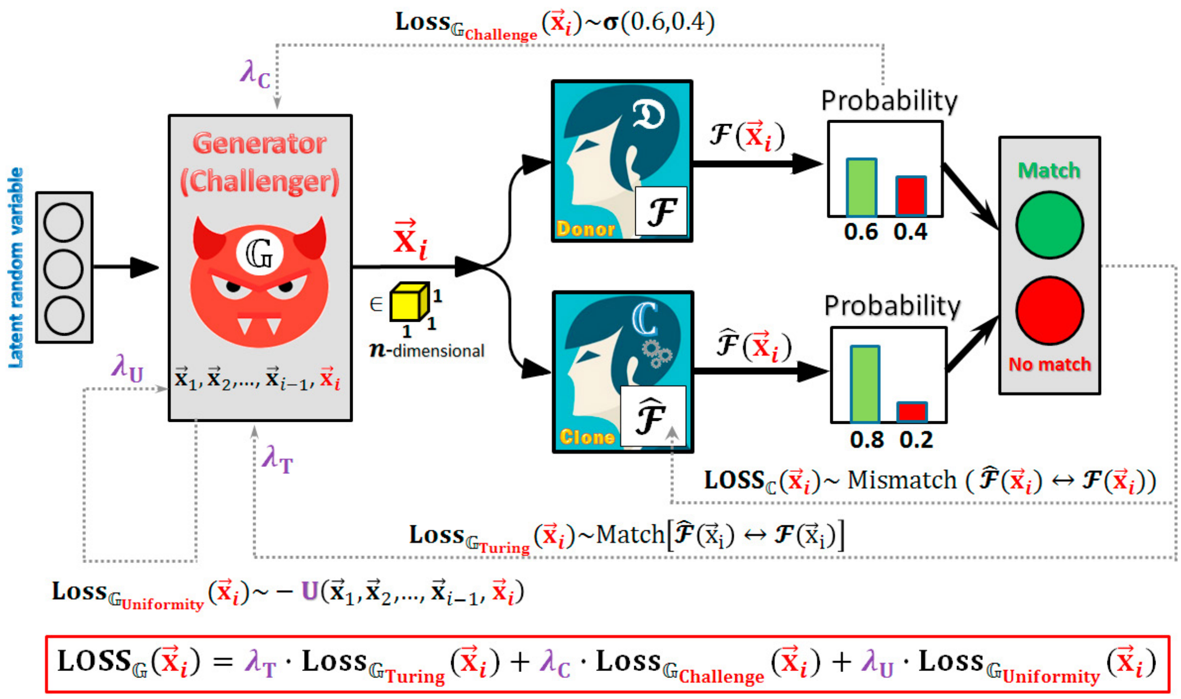
$$\Delta^c = \{\forall \vec{p}_i: (p_1, p_2, \dots, p_c) \in [0,1]^c | \textstyle\sum_{k=1}^{c} p_k = 1\};$$

c) $\forall \vec{x}_i: (x_1, x_2, \dots, x_n) \in [0,1]^n, \exists! \vec{p}_i: (p_1, p_2, \dots, p_c)_i \in \Delta^c,\ \vec{p}_i = \boldsymbol{\mathcal{F}}(\vec{x}_i).$

Simply speaking, a **DONOR** behaves as a labeling (classification) function (like a neural network), which takes a vector of numeric values (normalized to [0,1]) of some target object features as an input and outputs the probability distribution of that object having a particular label (i.e., belonging to a particular class) among several possible ones.

Cloning is a supervised learning process, in which the **DONOR** is a supervisor (i.e., labels the training set samples), and the task is to train the neural network model as a **CLONE** of the **DONOR**, i.e., to learn the hidden labeling function $\boldsymbol{\mathcal{F}}$ defined above, as precisely as possible. For similar tasks, the traditionally used term is "knowledge distillation". However, we are using the "cloning" term to follow consistently the terminology from our former articles where the ultimate objective was formulated as designing digital cognitive clones of humans (as decision-makers) and groups of collective hybrid intelligence (see, e.g. [12], [7], [13], [6], and [14]).

Cloning-GAN architecture (see **Fig. 1**), is a special kind of Generative Adversarial Network (GAN) architecture (see review on GANs in [15]), and it is suggested to **facilitate** cloning process by smart generation of special (challenging or adversarial) samples-as-queries for the **DONOR** for labeling and then synchronously and incrementally train the **CLONE** on the basis of generated and labeled samples.



$\mathbf{Loss}_{\mathbb{G}_{\text{Challenge}}}(\vec{\mathbf{x}}_i) \sim \sigma(0.6, 0.4)$
λ_C
Generator (Challenger)
Latent random variable
λ_U
$\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{i-1}, \vec{x}_i$
$\vec{\mathbf{x}}_i$
$\in$ n-dimensional
Donor
$\mathcal{F}(\vec{\mathbf{x}}_i)$
Probability
0.6 0.4
Clone
$\hat{\mathcal{F}}(\vec{\mathbf{x}}_i)$
Probability
0.8 0.2
Match
No match
λ_T
$\mathbf{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i) \sim \text{Mismatch}(\hat{\mathcal{F}}(\vec{\mathbf{x}}_i) \leftrightarrow \mathcal{F}(\vec{\mathbf{x}}_i))$
$\mathbf{Loss}_{\mathbb{G}_{\text{Turing}}}(\vec{\mathbf{x}}_i) \sim \text{Match}[\hat{\mathcal{F}}(\vec{x}_i) \leftrightarrow \mathcal{F}(\vec{x}_i)]$
$\mathbf{Loss}_{\mathbb{G}_{\text{Uniformity}}}(\vec{\mathbf{x}}_i) \sim -\mathbf{U}(\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_{i-1}, \vec{\mathbf{x}}_i)$
$\mathbf{LOSS}_{\mathbb{G}}(\vec{\mathbf{x}}_i) = \lambda_T \cdot \mathbf{Loss}_{\mathbb{G}_{\text{Turing}}}(\vec{\mathbf{x}}_i) + \lambda_C \cdot \mathbf{Loss}_{\mathbb{G}_{\text{Challenge}}}(\vec{\mathbf{x}}_i) + \lambda_U \cdot \mathbf{Loss}_{\mathbb{G}_{\text{Uniformity}}}(\vec{\mathbf{x}}_i)$

Fig 1. Cloning-GAN architecture

GENERATOR vs **CLONE** "game" in **Cloning-GAN** is driven by the conflicting objectives of the two synchronously trained adversaries. The objective of the **GENERATOR** training is to generate the most challenging (adversarial, puzzling) training samples for **CLONE**. The objective of the **CLONE** training is the capability to predict (guess, imitate) the labeling outcomes (especially biased ones) from the **DONOR** as close as possible in the challenging cases. The loss function for the **CLONE** is clear – it provides punishment feedback for the mismatch between its own outcomes and corresponding outcomes from the **DONOR**. Therefore, the most sophisticated task in the **Cloning-GAN** architecture is to define the loss function for the **GENERATOR**, which is naturally more complex one because it has to take into account several different criteria for the quality of generated samples.

Let us provide more details on the loss functions regarding **GENERATOR** vs **CLONE** training.

2.2. CLONE loss in Cloning-GAN

The loss of the **CLONE** in each sample $\vec{\mathbf{x}}_i$ (denoted as $\boldsymbol{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i)$) is a normalized measure ($\boldsymbol{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i) \in [0,1]$) of the two probability distribution vectors' mismatch (aka "Turing" mismatch): how **CLONE** addresses $\vec{\mathbf{x}}_i$ (i.e., $\hat{\mathcal{F}}(\vec{\mathbf{x}}_i)$: $(\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_c)$) vs how **DONOR** addresses $\vec{\mathbf{x}}_i$ (i.e., $\mathcal{F}(\vec{\mathbf{x}}_i)$: $(p_1, p_2, \ldots, p_c)$):

$$\boldsymbol{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i) = \text{Mismatch}\left[\hat{\mathcal{F}}(\vec{x}_i) \leftrightarrow \mathcal{F}(\vec{x}_i)\right] = \frac{1}{\sqrt{2}} \cdot \boldsymbol{d}\left(\hat{\mathcal{F}}(\vec{\mathbf{x}}_i), \mathcal{F}(\vec{\mathbf{x}}_i)\right) = \sqrt{\frac{(p_1-\hat{p}_1)^2+(p_2-\hat{p}_2)^2+\cdots+(p_c-\hat{p}_c)^2}{2}}, \quad (1)$$

where $\boldsymbol{d}$ – Euclidian distance.

Another, more computationally expensive but also more solid, option would be use of Jensen-Shannon Distance instead of Euclidean distance in formula (1). It is a metric [16] bounded to [0,1]; it is based on the concept of information entropy; and it is specifically designed to measure distance between probability distributions, while Euclidean distance is a general-purpose measure. Therefore, another option of the **CLONE**'s loss formula would be:

$$\boldsymbol{LOSS}_{\mathbb{C}}^{\text{JSD}}(\vec{\mathbf{x}}_i) = \mathbf{JSD}\left(\hat{\mathcal{F}}(\vec{\mathbf{x}}_i), \mathcal{F}(\vec{\mathbf{x}}_i)\right) = \sqrt{\frac{1}{2} \cdot \sum_{k=1}^{c} \left[\hat{p}_k \cdot \log_2 \frac{2 \cdot \hat{p}_k}{\hat{p}_k + p_k} + p_k \cdot \log_2 \frac{2 \cdot p_k}{\hat{p}_k + p_k}\right]}, \quad (1^*)$$

where **JSD** – Jensen-Shannon Distance defined as a metric in [16].

See some examples below:

$\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: (1,0,0); $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: (0,0,1) $\Rightarrow \boldsymbol{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i) = \sqrt{\frac{(1-0)^2+(0-0)^2+(0-1)^2}{2}} = 1$;

$\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: (1,0,0); $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: (0,0,1) $\Rightarrow \boldsymbol{LOSS}_{\mathbb{C}}^{\mathrm{JSD}}(\vec{\mathbf{x}}_i) = \sqrt{\frac{\left[1\cdot log_2\frac{2\cdot 1}{1}+0\cdot log_2\frac{2\cdot 0}{1}\right]+\left[0\cdot log_2\frac{2\cdot 0}{0}+0\cdot log_2\frac{2\cdot 0}{0}\right]+\left[0\cdot log_2\frac{2\cdot 0}{1}+1\cdot log_2\frac{2\cdot 1}{1}\right]}{2}} \to 1$;

$\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: (0.5,0.3,0.2); $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: (0.1,0.2,0.7) $\Rightarrow \boldsymbol{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i) = \sqrt{\frac{(0.5-0.1)^2+(0.3-0.2)^2+(0.2-0.7)^2}{2}} \approx 0.458$;

$\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: (0.5,0.3,0.2); $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: (0.1,0.2,0.7) $\Rightarrow \boldsymbol{LOSS}_{\mathbb{C}}^{\mathrm{JSD}}(\vec{\mathbf{x}}_i) =$

$= \sqrt{\frac{\left[0.5\cdot log_2\frac{2\cdot 0.5}{0.5+0.1}+0.1\cdot log_2\frac{2\cdot 0.1}{0.5+0.1}\right]+\left[0.3\cdot log_2\frac{2\cdot 0.3}{0.3+0.2}+0.2\cdot log_2\frac{2\cdot 0.2}{0.3+0.2}\right]+\left[0.2\cdot log_2\frac{2\cdot 0.2}{0.2+0.7}+0.7\cdot log_2\frac{2\cdot 0.7}{0.2+0.7}\right]}{2}} \approx 0.467.$

2.3. GENERATOR overall loss and its components in Cloning-GAN

The loss of the **GENERATOR** in each generated sample $\vec{\mathbf{x}}_i$ (denoted as $\boldsymbol{LOSS}_{\mathbb{G}}(\vec{\mathbf{x}}_i)$) is constructed from three loss components as follows:

$$LOSS_{\mathbb{G}}(\vec{\mathbf{x}}_i) = \boldsymbol{\lambda}_{\mathbf{T}} \cdot \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{\mathbf{x}}_i) + \boldsymbol{\lambda}_{\mathbf{C}} \cdot \mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\mathbf{x}}_i) + \boldsymbol{\lambda}_{\mathbf{U}} \cdot \mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\vec{\mathbf{x}}_i). \quad (2)$$

The ***"Turing" component*** of the **GENERATOR**'s loss on each generated sample $\vec{\mathbf{x}}_i$ (denoted as $\mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{\mathbf{x}}_i)$) is a normalized measure ($\in [0,1]$) of the two probability distribution vectors' match (aka "Turing" match): how **CLONE** addresses $\vec{\mathbf{x}}_i$ (i.e., $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: $(\hat{p}_1, \hat{p}_2, \dots, \hat{p}_c)$) vs how **DONOR** addresses $\vec{\mathbf{x}}_i$ (i.e., $\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: $(p_1, p_2, \dots, p_c)$):

$$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{\mathbf{x}}_i) = \mathrm{Match}\left[\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathrm{x}}_\mathrm{i}) \leftrightarrow \boldsymbol{\mathcal{F}}(\vec{\mathrm{x}}_\mathrm{i})\right] = \mathbf{1} - \boldsymbol{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i) = 1 - \sqrt{\frac{(p_1-\hat{p}_1)^2+(p_2-\hat{p}_2)^2+\dots+(p_c-\hat{p}_c)^2}{2}}. \quad (3)$$

One may notice that "Turing" component of the **GENERATOR**'s loss on each sample $\vec{\mathbf{x}}_i$ is opposite to the loss of the **CLONE** on the same sample, because one of the **GENERATOR**'s objectives is to generate such samples, which would be the most difficult ones for the **CLONE** in predicting the **DONOR**'s outcomes.

For the same purpose, we can also use a modification of formula (3) if we apply the Jensen-Shannon Distance defined by formula (1*):

$$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}^{\mathrm{JSD}}(\vec{\mathbf{x}}_i) = \mathbf{1} - \boldsymbol{LOSS}_{\mathbb{C}}^{\mathrm{JSD}}(\vec{\mathbf{x}}_i) = 1 - \sqrt{\frac{1}{2} \cdot \sum_{k=1}^{c}\left[\hat{p}_k \cdot \log_2 \frac{2\cdot\hat{p}_k}{\hat{p}_k+p_k} + p_k \cdot \log_2 \frac{2\cdot p_k}{\hat{p}_k+p_k}\right]}. \quad (3^*)$$

See some examples below:

$\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: (1,0,0); $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: (0,0,1) $\Rightarrow \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{\boldsymbol{x}}_i) = \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}^{\mathrm{JSD}}(\vec{\mathbf{x}}_i) = 1 - 1 = 0$;

$\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: (0.3,0.4,0.3); $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: (0.2,0.4,0.4) $\Rightarrow \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{\boldsymbol{x}}_i) = 1 - 0.1 = 0.9$;

$\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)$: (0.3,0.4,0.3); $\widehat{\boldsymbol{\mathcal{F}}}(\vec{\mathbf{x}}_i)$: (0.2,0.4,0.4) $\Rightarrow \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}^{\mathrm{JSD}}(\vec{\boldsymbol{x}}_i) \approx 1 - 0.111 = 0.889$.

Further in the paper, we will use only terms $\boldsymbol{LOSS}_{\mathbb{C}}(\vec{\mathbf{x}}_i)$ to represent **CLONE** loss and $\mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{\mathbf{x}}_i)$ to represent the "Turing" component of the **GENERATOR**'s loss, assuming that particular computing options for it, either (1) and (3) or (1*) and (3*), could be chosen depending on the task.

The ***"Challenge" component*** of the **GENERATOR**'s loss on each generated sample $\vec{\mathbf{x}}_i$ (denoted as $\mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\mathbf{x}}_i)$) is a normalized measure ($\in [0,1]$) of how easy it would be for the **DONOR** to confidently label the generated sample or, therefore, how far is the sample from being the "adversarial" one and being able to confuse the **DONOR** (i.e., how far is the probability distribution provided by **DONOR** on sample $\vec{\mathbf{x}}_i$ from the uniform one):

$$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\mathrm{x}}_\mathrm{i}) = \frac{c}{\sqrt{c-1}} \cdot \boldsymbol{\sigma}(p_1, p_2, \dots, p_c) = \sqrt{\frac{c\cdot(p_1^2+p_2^2+\dots+p_c^2)-1}{c-1}}, \quad (4)$$

where $\boldsymbol{\sigma}$ – standard deviation.

$$\boldsymbol{\sigma}(p_1, p_2, \ldots, p_c) = \sqrt{\frac{\left(p_1-\frac{1}{c}\right)^2+\left(p_2-\frac{1}{c}\right)^2+\cdots+\left(p_c-\frac{1}{c}\right)^2}{c}} = \cdots = \frac{\sqrt{c\cdot(p_1^2+p_2^2+\cdots+p_c^2)-1}}{c}.$$

$$\mathbf{MAX}[\boldsymbol{\sigma}(p_1, p_2, \ldots, p_c)] = \boldsymbol{\sigma}(1,\ 0, \ldots, 0) = \boldsymbol{\sigma}(0, \ldots, 0, 1) = \boldsymbol{\sigma}(0,\ \ldots, 1, \ldots, 0) = \frac{\sqrt{c\cdot(0+\cdots+1+\cdots+0)-1}}{c} = \frac{\sqrt{c-1}}{c}.$$

$$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\mathbf{x}}_i) = \frac{\sigma(p_1,p_2,\ldots,p_c)}{\mathbf{MAX}[\sigma(p_1,p_2,\ldots,p_c)]} = \frac{c}{\sqrt{c-1}}\cdot\boldsymbol{\sigma}(p_1, p_2, \ldots, p_c) = \frac{c}{\sqrt{c-1}}\cdot\frac{\sqrt{c\cdot(p_1^2+p_2^2+\cdots+p_c^2)-1}}{c} = \sqrt{\frac{c\cdot(p_1^2+p_2^2+\cdots+p_c^2)-1}{c-1}}.$$

Therefore, $\mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\mathbf{x}}_i) \in [\mathbf{0}, \mathbf{1}]$ - aka normalized standard deviation $\boldsymbol{\sigma}$. The more standard deviation $\boldsymbol{\sigma}$ – the less confusion for the **DONOR** – the more loss for the **GENERATOR**.

The “Challenge” component of the **GENERATOR**’s loss ensures appearance of challenging-and-adversarial (close to decision boundaries and corner cases) samples, helping the **CLONE** to learn faster the individual biases of the **DONOR**.

See some examples below:

$$\boldsymbol{\mathcal{F}}(\vec{\boldsymbol{x}}_i)\colon (1,0,0) \Rightarrow \mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\boldsymbol{x}}_i) = \sqrt{\frac{3\cdot(1^2+0^2+0^2)-1}{2}} = 1;$$

$$\boldsymbol{\mathcal{F}}(\vec{\boldsymbol{x}}_i)\colon (0.5,0.3,0.2) \Rightarrow \mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\boldsymbol{x}}_i) = \sqrt{\frac{3\cdot(0.5^2+0.3^2+0.2^2)-1}{2}} \approx 0.26;$$

$$\boldsymbol{\mathcal{F}}(\vec{\boldsymbol{x}}_i)\colon (\tfrac{1}{3},\tfrac{1}{3},\tfrac{1}{3}) \Rightarrow \mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\boldsymbol{x}}_i) = \sqrt{\frac{3\cdot3\cdot\left(\frac{1}{3}\right)^2-1}{2}} = 0.$$

It would be *important to mention* here that the “Challenge” component of the **GENERATOR**’s loss makes sense only in such “optimistic” cases when the **DONOR** responds with the so-called “soft targets”, i.e., not only indicates the “winning” class but provides the complete “ranking list” with the scores-as-probabilities for each of the classes involved (the universe here is a disointUnionOf c classes). This case is a typical outcome from the **DONOR**, which is a hidden neural network classifier with disclosed output (e.g., softmax) layer. In “the-winner-takes-all” classification output cases, our probability vector (so called “hard target”) will be represented by one-hot encoded vector containing one “1” with the rest “0” and, therefore, the “challenge” component of the loss function will always be constant and equal to $\frac{\sqrt{c-1}}{c}$ (see above). However, such “blind” cases will be approached similarly (although less efficient), i.e., by applying the same formula (2) but without the “Challenge” component in it.

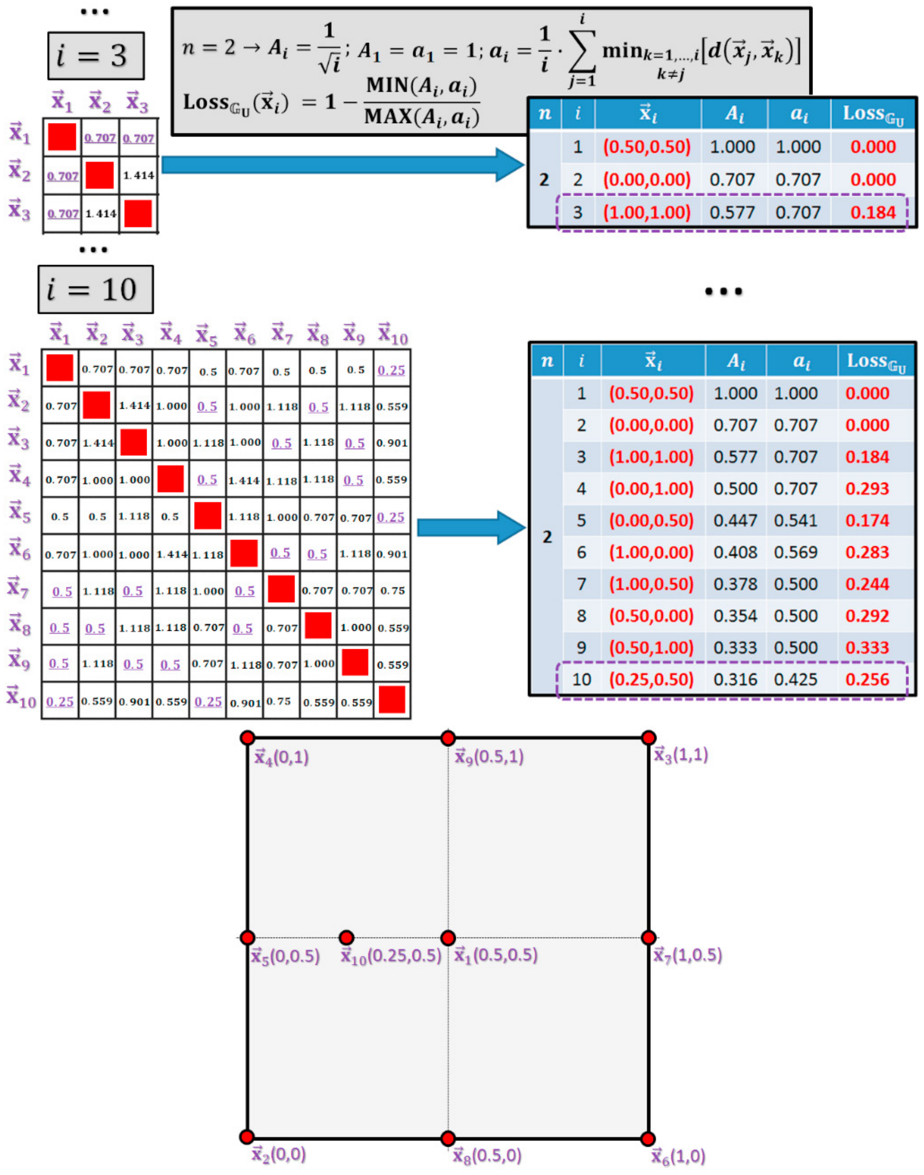
The ***“Uniformity” component.*** One of the **GENERATOR**’s objectives is to generate adversarial samples “everywhere” (close to uniform distribution) within the decision space bounded by the unit hypercube. “Uniformity” component of the **GENERATOR**’s loss regarding each generated sample $\vec{\mathbf{x}}_i$ (denoted as $\mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\vec{\mathbf{x}}_i)$) is a normalized measure ($\in [0,1]$) of how far is the distribution of previously generated samples including the new one ($\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_{i-1}, \vec{\mathbf{x}}_i$) from the uniform distribution. In fact, we compare the estimated average-nearest-neighbor-distance ($\boldsymbol{A}_i$) for i samples uniformly distributed within an n-dimensional unit hypercube with the actual average-nearest-neighbour-distance ($\boldsymbol{a}_i$) for $\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_{i-1}, \vec{\mathbf{x}}_i$ generated samples (i.e., after $\boldsymbol{i}$th generated sample $\vec{\boldsymbol{x}}_i$ in $\boldsymbol{n}$-dimensional unit hypercube) as follows:

$$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\vec{\mathbf{x}}_i) = 1 - \frac{\mathbf{MIN}(A_i, a_i)}{\mathbf{MAX}(A_i, a_i)}, \tag{5}$$

where $\boldsymbol{A}_i = \boldsymbol{i}^{-\frac{1}{n}}$ is a good heuristic approximation for the intended average nearest neighbor distance from [17];

$a_1 = 1;\ a_i = \frac{1}{i}\cdot\sum_{j=1}^{i}\mathbf{min}_{k=1,\ldots i; k\neq j}\left[\boldsymbol{d}(\vec{\boldsymbol{x}}_j, \vec{\boldsymbol{x}}_k)\right]$ ($\boldsymbol{d}$ – Euclidian distance).

One may see that this calculation regarding each generated sample takes into account information on previously generated samples. However, it does not consume much memory for it. See example in **Fig. 2**. It shows how the “Uniformity” loss is computed during the iterative process of ten samples’ generation. For each previously generated sample, this process keeps in its memory only the distance to its nearest neighbor (i.e., collecting the minimal distances needed to calculate $\boldsymbol{a}_i$) and refines it when a new sample arrives. One may see these minimal distances, to be kept in memory, outlined within the distance matrixes shown in **Fig 2**.



$$n = 2 \to A_i = \frac{1}{\sqrt{i}};\ A_1 = a_1 = 1;\ a_i = \frac{1}{i}\cdot\sum_{j=1}^{i}\min_{\substack{k=1,\dots,i\\ k\neq j}}\left[d(\vec{x}_j,\vec{x}_k)\right]$$

$$\mathrm{Loss}_{\mathbb{G}_U}(\vec{x}_i) = 1 - \frac{\mathrm{MIN}(A_i, a_i)}{\mathrm{MAX}(A_i, a_i)}$$

	$\vec{x}_1$	$\vec{x}_2$	$\vec{x}_3$
$\vec{x}_1$	🟥	0.707	0.707
$\vec{x}_2$	0.707	🟥	1.414
$\vec{x}_3$	0.707	1.414	🟥

n	i	$\vec{x}_i$	A_i	a_i	$\mathrm{Loss}_{\mathbb{G}_U}$
2	1	(0.50,0.50)	1.000	1.000	0.000
	2	(0.00,0.00)	0.707	0.707	0.000
	3	(1.00,1.00)	0.577	0.707	0.184

	$\vec{x}_1$	$\vec{x}_2$	$\vec{x}_3$	$\vec{x}_4$	$\vec{x}_5$	$\vec{x}_6$	$\vec{x}_7$	$\vec{x}_8$	$\vec{x}_9$	$\vec{x}_{10}$
$\vec{x}_1$	🟥	0.707	0.707	0.707	0.5	0.707	0.5	0.5	0.5	0.25
$\vec{x}_2$	0.707	🟥	1.414	1.000	0.5	1.000	1.118	0.5	1.118	0.559
$\vec{x}_3$	0.707	1.414	🟥	1.000	1.118	1.000	0.5	1.118	0.5	0.901
$\vec{x}_4$	0.707	1.000	1.000	🟥	0.5	1.414	1.118	1.118	0.5	0.559
$\vec{x}_5$	0.5	0.5	1.118	0.5	🟥	1.118	1.000	0.707	0.707	0.25
$\vec{x}_6$	0.707	1.000	1.000	1.414	1.118	🟥	0.5	0.5	1.118	0.901
$\vec{x}_7$	0.5	1.118	0.5	1.118	1.000	0.5	🟥	0.707	0.707	0.75
$\vec{x}_8$	0.5	0.5	1.118	1.118	0.707	0.5	0.707	🟥	1.000	0.559
$\vec{x}_9$	0.5	1.118	0.5	0.5	0.707	1.118	0.707	1.000	🟥	0.559
$\vec{x}_{10}$	0.25	0.559	0.901	0.559	0.25	0.901	0.75	0.559	0.559	🟥

n	i	$\vec{x}_i$	A_i	a_i	$\mathrm{Loss}_{\mathbb{G}_U}$
2	1	(0.50,0.50)	1.000	1.000	0.000
	2	(0.00,0.00)	0.707	0.707	0.000
	3	(1.00,1.00)	0.577	0.707	0.184
	4	(0.00,1.00)	0.500	0.707	0.293
	5	(0.00,0.50)	0.447	0.541	0.174
	6	(1.00,0.00)	0.408	0.569	0.283
	7	(1.00,0.50)	0.378	0.500	0.244
	8	(0.50,0.00)	0.354	0.500	0.292
	9	(0.50,1.00)	0.333	0.500	0.333
	10	(0.25,0.50)	0.316	0.425	0.256

Fig 2. Example illustrating "Uniformity" component of Cloning-GAN (based on iterative computing regarding 10 generated samples)

2.4. Trade-off among the GENERATOR's loss components

Performance of the **GENERATOR** and, therefore, the whole Cloning-GAN will depend on the choice of the importance weights $\lambda_T, \lambda_C, \lambda_U$, which correspond to different components of the **GENERATOR**'s loss from formula (2). Let us consider different approaches to balance with these weights, which will correspond to different options of the Cloning-GAN architecture.

The ***Basic Cloning-GAN*** (B-C-GAN) architecture supposes manual initialization and control of weights $\lambda_T, \lambda_C, \lambda_U$ (assuming that $\lambda_T + \lambda_C + \lambda_U = 1$), which will be considered as constants until changed manually if needed.

The ***Dynamic Cloning-GAN*** (D-C-GAN) architecture assumes the dependence of the weights, which correspond to different components of the **GENERATOR**'s loss from formula (2), on the current learning iteration (epoch) $\boldsymbol{i}$ as follows:

$\boldsymbol{\lambda_T}(\boldsymbol{i}) + \boldsymbol{\lambda_C}\,(\boldsymbol{i}) + \boldsymbol{\lambda_U}(\boldsymbol{i}) = 1$;

$\boldsymbol{\lambda_T}(\boldsymbol{i}) = 0.5$, i.e., constant, which takes half of the overall importance always during training;

$\boldsymbol{\lambda_C}(\boldsymbol{i}) = \frac{i}{2\cdot(i+i^*)}$, i.e., monotonously increasing importance from 0 (at the beginning) to 0.5 (at infinity);

$\boldsymbol{\lambda_U}(\boldsymbol{i}) = \frac{i^*}{2\cdot(i+i^*)}$, i.e., monotonously decreasing importance from 0.5 (at the beginning) to 0 (at infinity), where $\boldsymbol{i}^*$ is the controlling parameter (integer >1, which indicates at which iteration $\boldsymbol{\lambda_C} = \boldsymbol{\lambda_U} = 0.25$).

See illustrative explanation in **Fig**. 3.

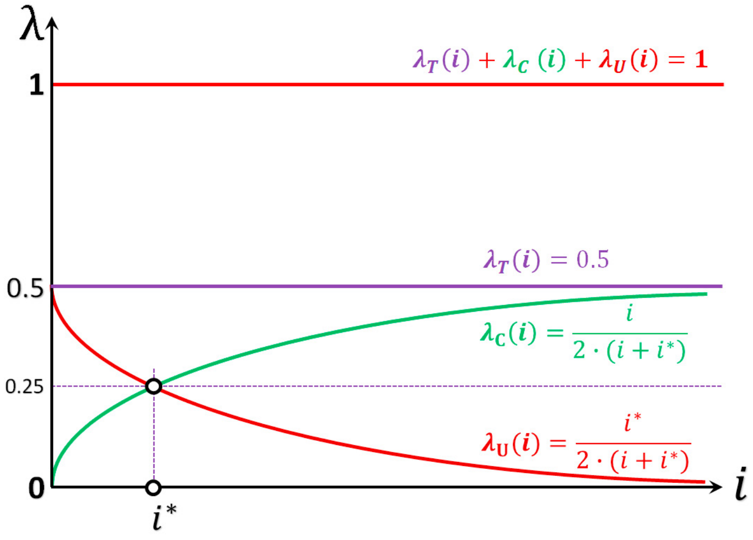


Fig 3. Plots show a trade-off among the importance of the GENERATOR's loss function components during training.

Therefore, for D-C-GAN, the loss of the **GENERATOR** from formula (2) on each generated sample $\vec{\mathbf{x}}_i$ could be updated as follows:

$$LOSS_{\mathbb{G}}(\vec{\mathbf{x}}_i) = 0.5\cdot \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{\mathbf{x}}_i) + \frac{i}{2\cdot(i+i^*)}\cdot \mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{\mathbf{x}}_i) + \frac{i^*}{2\cdot(i+i^*)}\cdot \mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\vec{\mathbf{x}}_i). \quad (6)$$

Training process performance regarding the **GENERATOR** and, therefore, the **CLONE** can be controlled just by one parameter i^* depending on the cloning case specifics.

We consider the "Turing" component to be equally important (50% of the overall importance) during the training process because it directly influences the performance of the **CLONE** to simulate the **DONOR** closely enough.

We assume that the ability of the **GENERATOR** to generate "diverse" (everywhere within the decision space) rather than "difficult" (challenging, adversarial) inputs for the **DONOR-CLONE** couple is more important at the beginning at the training process, and the trend towards more challenging rather than different inputs will become influential later in the training process. The trend itself (i.e., how fast the capability to generate challenging inputs will become as important as the capability to generate diverse inputs) could be controlled by parameter $\boldsymbol{i}^*$.

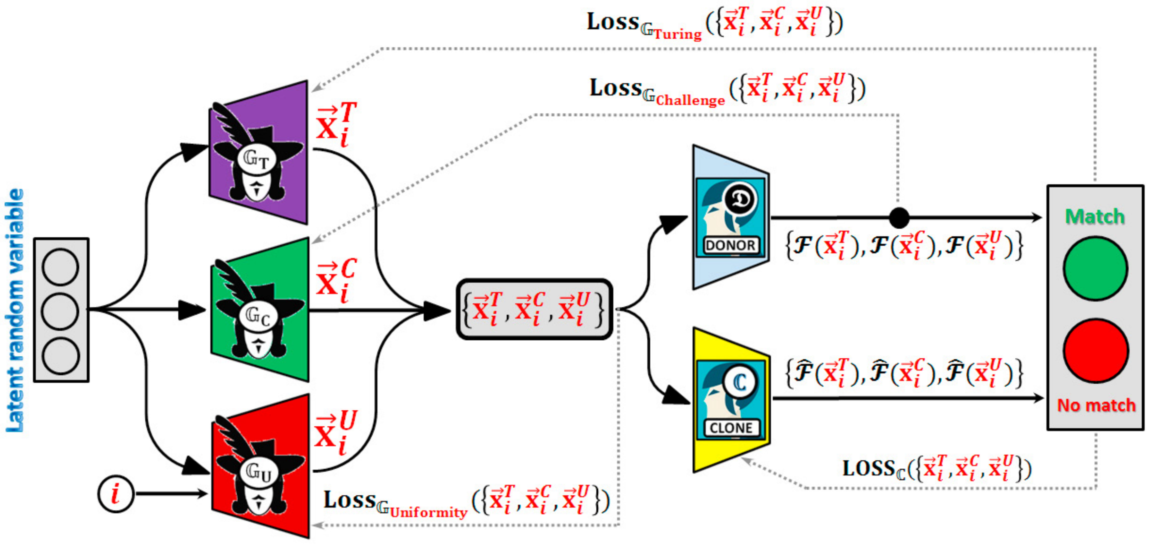
The reasonability for dynamic weights of different loss components is yet to be checked experimentally. The intuition is that Cloning-GANs (of the D-C-GAN option) may converge better when some criterion is dominating at the beginning of the process and another one - at the end. **GENERATOR**, like a kind of "young journalist", in the beginning, learns to ask "different" questions and, when mature with this skill, switches to training his own capability of asking "difficult" questions also.

The ***"Three Musketeers" ("All for one and one for all") Cloning-GAN*** (3M-C-GAN) architecture differs significantly from the previous two architectures. It supposes splitting the **GENERATOR** to three **GENERATOR**s ("Turing" or $\mathbb{G}_{\mathbf{T}}$, "Challenge" or $\mathbb{G}_{\mathbf{C}}$, and "Uniformity" or $\mathbb{G}_{\mathbf{U}}$), one responsible for each loss component from formula (2). Before that, we considered only the incremental learning option for the Cloning-GAN architecture, which could be extended to the "minibatch" learning case when all the loss components are computed as aggregates over few generated samples. In particular, the 3M-C-GAN architecture assumes that, at i-th iteration, each of three **GENERATOR**s independently generates one sample each and, therefore, resulting in a minibatch of three samples $\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\}$ as shown in **Fig. 4**. This minibatch goes through the same process as in normal Cloning-GAN described above so that the component losses are aggregated and distributed among the **GENERATOR**s as follows:

$$\mathbf{LOSS}_{\mathbb{G}_T}\left(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\}\right) = \frac{1}{3} \cdot \left(\mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{x}_i^T) + \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{x}_i^C) + \mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\vec{x}_i^U)\right); \quad (7)$$

$$\mathbf{LOSS}_{\mathbb{G}_C}\left(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\}\right) = \frac{1}{3} \cdot \left(\mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{x}_i^T) + \mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{x}_i^C) + \mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\vec{x}_i^U)\right); \quad (8)$$

$$\mathbf{LOSS}_{\mathbb{G}_U}\left(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\}\right) = \frac{1}{3} \cdot \left(\mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\vec{x}_i^T) + \mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\vec{x}_i^C) + \mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\vec{x}_i^U)\right). \quad (9)$$



$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\})$
$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Challenge}}}(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\})$
Latent random variable
$\vec{x}_i^T$
$\vec{x}_i^C$
$\vec{x}_i^U$
$\mathbb{G}_T$
$\mathbb{G}_C$
$\mathbb{G}_U$
i
$\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\}$
DONOR
CLONE
$\{\mathcal{F}(\vec{x}_i^T), \mathcal{F}(\vec{x}_i^C), \mathcal{F}(\vec{x}_i^U)\}$
$\{\hat{\mathcal{F}}(\vec{x}_i^T), \hat{\mathcal{F}}(\vec{x}_i^C), \hat{\mathcal{F}}(\vec{x}_i^U)\}$
Match
No match
$\mathbf{LOSS}_{\mathbb{C}}(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\})$
$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Uniformity}}}(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\})$

Fig 4. Architecture of the "Three Musketeers" ("*All for one* and *one for all*") Cloning-GAN (3M-C-GAN).

The main feature of 3M-C-GAN architecture is that each of the **GENERATOR**s, after generating just one sample, will be punished for the loss created by all the minibatch (i.e., will be responsible for "colleagues'" performance also). For example, all the loss $\mathbf{Loss}_{\mathbb{G}_{\mathbf{Turing}}}(\{\vec{x}_i^T, \vec{x}_i^C, \vec{x}_i^U\})$ computed according to formula (7) will be backpropagated through the "Turing" **GENERATOR** $\mathbb{G}_\mathbf{T}$, although it is actually responsible only for one sample of the three. In this way, the architecture ensures that all the three **GENERATOR**s are responsible for each one and each one is responsible for all. Therefore, we used "Three Musketeers" (with the motto "All for one and one for all") as a name for such an architecture. One of the advantages of this architecture is that we do not need to worry about the weights $\boldsymbol{\lambda_T}, \boldsymbol{\lambda_C}, \boldsymbol{\lambda_U}$ from formula (2) because each **GENERATOR** addresses independently only its own specific loss criteria. We can also expect better convergence of such a training process managed by multiple **GENERATOR**s compared with one **GENERATOR** with complex and conflicting criteria.

If necessary, the missing "D'Artagnan" from the "Three Musketeers" architecture can be enabled as an additional (fourth) generation quality objective and, therefore, an additional **GENERATOR** to be responsible for the so-called "hostility" of generated samples. Such a component could enable not only challenging content for the **DONOR**, which is the duty of the "Challenge" component, but rather to ensure that more challenging ("hostile") areas in data space will be covered more with the diverse-by-labeling generated samples.

Assume that:

- n-dimensional sample $\vec{\mathbf{x}}_{NN}$: $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n)$, which belongs to previously generated samples $(\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \dots, \vec{\mathbf{x}}_{i-1})$ and labeled (to c classes) by the **DONOR** as $\mathcal{F}(\vec{\mathbf{x}}_{NN})$: $(\tilde{p}_1, \tilde{p}_2, \dots, \tilde{p}_c)$, is the nearest neighbor (NN) of the newly generated sample $\vec{\mathbf{x}}_i$: $(x_1, x_2, \dots, x_n)$ labeled by the **DONOR** as $\mathcal{F}(\vec{\mathbf{x}}_i)$: $(p_1, p_2, \dots, p_c)$;
- The "uncertainty gain" of $\vec{\mathbf{x}}_i$ in comparison with its nearest neighbor $\vec{\mathbf{x}}_{NN}$ is the following asymmetric measure:

$\Delta\boldsymbol{\sigma}(\vec{\boldsymbol{x}}_{NN},\vec{\boldsymbol{x}}_i) = \boldsymbol{\sigma}(\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_{NN})) - \boldsymbol{\sigma}(\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i))$, where: $\boldsymbol{\sigma}(\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_i)) = \boldsymbol{\sigma}(p_1, p_2, \ldots, p_c) = \frac{1}{c} \cdot \sqrt{c \cdot (p_1^2 + p_2^2 + \cdots + p_c^2) - 1}$ and $\boldsymbol{\sigma}(\boldsymbol{\mathcal{F}}(\vec{\mathbf{x}}_{NN})) = \boldsymbol{\sigma}(\tilde{p}_1, \tilde{p}_2, , \ldots, \tilde{p}_c) = \frac{1}{c} \cdot \sqrt{c \cdot (\tilde{p}_1^2 + \tilde{p}_2^2 + \cdots + \tilde{p}_c^2) - 1}$;

- The Euclidean distance between $\vec{\mathbf{x}}_i$ and its nearest neighbor $\vec{\mathbf{x}}_{NN}$ is $\boldsymbol{d}(\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{x}}_{NN})$ and it is computed as follows: $\boldsymbol{d}(\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{x}}_{NN}) = \sqrt{\sum_{\boldsymbol{k}=\mathbf{1}}^{\boldsymbol{n}} (x_k - \tilde{x}_k)^2}$.

Hostility of each newly generated sample $\vec{\mathbf{x}}_i$ is a normalized (by sigmoid function) heuristic measure ($\in [0,1]$) of a **DONOR**-based class-uncertainty-gain gradient in the vicinity of this sample:

$$\mathbf{Hostility}(\vec{\mathbf{x}}_\mathbf{i}) = \mathrm{SIG}(\mathbf{Gradient}) = \frac{1}{1+e^{-\mathbf{Gradient}}}, \text{ where } \mathbf{Gradient} = \frac{\Delta\boldsymbol{\sigma}(\vec{\boldsymbol{x}}_{NN},\vec{\boldsymbol{x}}_i)}{\boldsymbol{d}(\vec{\boldsymbol{x}}_i,\vec{\boldsymbol{x}}_{NN})}.$$

Assuming that maximizing the hostility of generated data is one of the **GENERATOR**'s objectives, the hostility component of the **GENERATOR**'s loss (denoted as $\mathbf{Loss}_{\mathbb{G}_{\mathbf{Hostility}}}(\vec{\mathbf{x}}_i)$) could be estimated as being opposite to the hostility as follows:

$$\mathbf{Loss}_{\mathbb{G}_{\mathbf{Hostility}}}(\vec{\mathbf{x}}_\mathbf{i}) = 1 - \mathbf{Hostility}(\vec{\mathbf{x}}_\mathbf{i}) = \frac{e^{-\mathbf{Gradient}}}{1+e^{-\mathbf{Gradient}}}. \quad (10)$$

The ***"Hostility" component*** of the **GENERATOR**'s loss (either as an extra objective for formula (2) or as a separate "musketeer"-**GENERATOR**) ensures class-label-diversity of samples, which are close to the corner cases' areas, helping the **CLONE** to faster learn the individual biases of the **DONOR**. See some examples below:

$n = 2; c = 3;\ \vec{\boldsymbol{x}}_i{:}\ (0{,}0); \boldsymbol{\mathcal{F}}(\vec{\boldsymbol{x}}_i){:}\ (0.4{,}0.4{,}0.2); \vec{\boldsymbol{x}}_{NN}{:}\ (1{,}1); \boldsymbol{\mathcal{F}}(\vec{\boldsymbol{x}}_{NN}){:}\ (0.1{,}0.2{,}0.7) \Rightarrow \overline{\boldsymbol{d}}(\boldsymbol{\mathcal{F}}(\vec{x}_i), \boldsymbol{\mathcal{F}}(\vec{\boldsymbol{x}}_{NN})) = \sqrt{0.19}$;
$\overline{\boldsymbol{d}}(\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{x}}_{NN}) = \frac{\sqrt{2}}{\sqrt{2}} = 1 \Rightarrow \mathbf{Gradient} = \frac{\sqrt{0.19}}{1} \approx 0.436 \Rightarrow \mathbf{Loss}_{\mathbb{G}_{Hostility}}(\vec{\boldsymbol{x}}_i) = \frac{1}{0.436+1} \approx 0.696.$

3. Related Work

Knowledge transfer. Suggested Cloning-GAN architecture is supposed to perform as some kind of knowledge transfer between **DONOR** and **CLONE**. Actually, knowledge transfer is a popular term referring to a wide range of methods, techniques and applications. In machine learning, it is the process of transferring knowledge learned from one machine learning model, called "teacher model" (aka **DONOR**), to another (smaller) model, called "student model" (aka **CLONE**), with the goal of improving the performance of the student model. This is often done by leveraging the pre-existing knowledge of the teacher model to guide and enhance the learning of the student model, resulting in a more accurate and efficient model. These techniques are widely used in smart manufacturing and particularly in intelligent fault diagnosis. See, e.g., review in [9]. Some applications require artificial intelligence and machine learning tasks to be done on edge devices, which have limited resources for running the large and complex models needed for these tasks. Therefore, knowledge transfer is used as one of possible solutions to compress the models from a larger network to a smaller one to improve performance while preserving accuracy. Study in [18] discovers the effectiveness of knowledge transfer for learning on edge devices, depending a lot on architectures and transfer techniques. Knowledge transfer is not necessarily considering a **DONOR** to be a black box, but it simply attempts to design the **CLONE** as an optimized version of the accessible **DONOR**. Knowledge transfer involves feature extraction, transfer learning, and sometimes "knowledge distillation" (a specific type of knowledge transfer, which needs special attention as it is closer to the objectives of our study).

Knowledge distillation. Knowledge distillation is a specific technique for knowledge transfer that involves training a smaller model to mimic the behavior and predictions of a larger, more complex model. We may say that knowledge distillation approaches the problem of creating a functional copy of some classifier in a way that it has the same biases as the target classifier. To do this, one would need to collect a set of input-output pairs from the target classifier that represent the confusing cases and biases to be replicated. Then these pairs will be used to train the student model using knowledge distillation. This is typically done by using the outputs (probabilities of the classes involved) of the teacher model as so-called "soft targets" to train the student model, rather than using the ground truth labels ("hard targets" or one-hot encoded vectors). In other words, the student model is trained to predict the same output probabilities for all the classes involved as the teacher's model does, rather than the true labels of the winning classes. The idea of using the outputs of one network to train another one was first proposed in [19] as a method for compressing large ensembles of classifiers into smaller and faster models with minimal performance loss. Ensembles of hundreds or thousands of classifiers can achieve the best performance in supervised learning, but storing and executing them in applications with large test sets or limited storage and computational power is not feasible. Therefore, the major need for knowledge distillation has been and is optimizing resource consumption in classification models, i.e., similar to the generic knowledge transfer objectives. Our interest is particularly in *data-free* knowledge distillation, which means that distillation is made with no access to the data used for training teacher's model (i.e., **DONOR**). In data-free knowledge distillation, the teacher model is trained on a dataset in the same way as traditional supervised learning. However, instead of using the original training data to distill knowledge into the student model, data-free methods use other techniques to generate synthetic data that the student model can learn from. This can be useful in situations where the training data is not available or where there are privacy concerns surrounding the use of sensitive data [20]. The status of such knowledge distillation methods has been provided in a comprehensive survey [10] from the perspectives of knowledge categories, training schemes, teacher-student architecture, distillation algorithms, performance comparison and applications. The survey concludes that, in order to improve the performance of knowledge distillation, it will be useful to integrate it with other learning schemes (adversarial, reinforcement, etc.) for practical challenges in the future. Generally, the existing distillation methods can be categorized into *offline distillation* (see, e.g., [21]), *online distillation* (see, e.g., [22]), the hybrid of both (see, e.g., [23]), and *self-distillation* (see, e.g., [24]). The main difference between online and offline distillation is in the way the teacher and student models are trained. In offline distillation, the teacher model is pre-trained separately and then used to distill knowledge into the student model, while in online distillation, both models are trained together and the student model learns from the teacher model throughout the training process. Self-distillation involves a single model that acts both as the teacher and as the student, for example, when deeper layers of a neural network model train the shallow layers.

Adversarial distillation. An important and relevant variation of knowledge distillation in our study is known as adversarial distillation. This is due to the fact that we want to teach the student model (**CLONE**) to copy not only correct answers but also the mistakes and biases of the teacher's model (**DONOR**). Adversarial distillation supposes the use of an adversarial network (e.g., GAN) to generate samples (aka soft targets), which are difficult ones for the student's model to predict correctly, while still conveying useful information about the underlying distribution of the data. In traditional distillation, the soft targets are generated by the teacher model itself, while, in adversarial distillation, these challenging examples are generated by an adversarial network that is specifically trained to do it. Adversarial distillation typically involves a discriminator network that is used to distinguish between the labeled samples generated by the adversarial network and the true labels. A good example of adversarial distillation, which uses GAN architecture, where a student, a teacher, and a discriminator models are trained adversarially, is provided in [11]. They suggested knowledge distillation GAN (KDGAN) architecture, in which the student and the teacher learn from each other via distillation losses and are adversarially trained against the discriminator via adversarial losses. By simultaneously optimizing the distillation and adversarial losses, the student will learn the true data distribution at the equilibrium. The discrete distribution learned by the student (or the teacher) is approximated with a concrete distribution, from which continuous samples are generated to obtain low-variance gradient updates to speed up the training. Data-free adversarial knowledge distillation related to image classification has been performed in [25] using architecture with an adversarial generator. Such a generator is trained to search for images on which the student model poorly matches the teacher's model and, after that, uses these images to train the student model. This process uses a metric to quantify the degree of teacher vs student belief matching near decision boundaries. In [26], it is also admitted that the generalization performance of a classifier depends a lot on the adequacy of its decision boundary, and, therefore, transferring knowledge closer to it is a key to successful knowledge distillation. A dual discriminator adversarial distillation architecture has been studied in [27]. In such an architecture, the generator creates samples and uses for that not only the pre-trained intrinsic statistics of the teacher's model but also obtains the maximum discrepancy from the student's model. Therefore, two different discriminators are employed for training the generator. The first discriminator encourages the generator to produce samples, which could mimic the distribution of the original training data. The second discriminator is employed to customize samples for the student's network aiming for better distillation performance. Generated samples are used to train the compact student network under the supervision of the teacher. The approach has been successfully evaluated on several public datasets.

In our Cloning-GAN architecture, the work on generating training samples for the **CLONE** is divided between the **GENERATOR**, who finds the coordinates for potentially challenging samples, and the **DONOR**, who labels it. This enables reaching adversarial distillation objectives without the use of a specific discriminator network component. However, the **DONOR**-**CLONE** pair in our architecture can be considered as a kind of complex discriminator, which is an update of "Turing" discriminator used in our former studies (see, e.g., [13]).

4. Discussion

In this section, we are going to show the evolution of the cloning concept and cloning architectures in our former studies to make explicit the role and added value of this study.

We started exploration of the digital cognitive cloning concept from the "smart resource" concept, which is a smart autonomous industrial asset (aka agent-driven digital twin). The concept of smart resources and enabling infrastructure (so-called "Global Understanding Environment" driven by UBIWARE as a smart middleware) evolved during two projects, SmartResource (2004-2006) and UBIWARE (2007-2010) and summarized in [28], [29], and [30] (see also: http://www.cs.jyu.fi/ai/SmartResource_UBIWARE.html). That time, we believed that digital copies of industrial assets must be proactive (agent-driven) to enable autonomous operation, interoperability and coordination among complex industrial assets, systems, and processes. Therefore, we developed mainly the dimension of autonomy for the emerging digital clone (twin) concept.

Later in [12], we summarized previously obtained results and extended them under the umbrella of Pi-Mind (so-called "patented intelligence") technology as an enabler of digital cognitive clones of humans. Pi-Mind is supposed to enable capturing, cloning, and patenting decision models from a particular human. That time, our solution was based on transparent ontology-driven modelling and Pi-Mind clones are supposed to make decisions based on an explicit personalized value system and preferences explicitly modelled for each intended individual. One of the key features of Pi-Mind has been the possibility for human experts to own, license, share, and sell their clones elsewhere. Therefore, the main objective and contribution of [12] were scenarios and business models where industrial processes would benefit from such human ubiquity and increased efficiency.

With the development of deep learning technologies, there appears a possibility to extend an explicit design of clones towards trainable ML modelling. In [13], we reported the first practical experiments on using adversarial learning to train digital cognitive clones of human decision makers. We suggested a Turing-GAN (T-GAN) extension to traditional GAN architecture, which is capable of minimizing the difference between the original decision-maker and its clone due to a Turing-discriminator component. The T-GAN architecture has been extended in [7] to enable cloning group decision-making in addition to individual clones. That time, T-GAN was a straightforward solution (quite a naïve loss function), which worked slowly and with relatively modest accuracy. Later, in [14], we discovered the similarity between digital cloning and cybersecurity (particularly, data poisoning and evasion attacks). In both cases, we generate adversarial samples using T-GANs. In the case of cloning, we ask **DONOR** to label adversarial samples to facilitate learning of the **CLONE**. In the case of training digital immunity against adversarial attacks, we used correctly labelled generated adversarial samples as a "vaccine" to improve the robustness of ML models.

In [6], we summarized our previous digital cognitive cloning results in a comprehensive study where complex ML-driven cloning (based on T-GAN) has been integrated with the autonomous and ontology-driven Pi-Mind cloning and tested within several application domains.

However, the weaknesses of the heuristic T-GAN implementations required adding more solid analytics to facilitate the cloning process and improve the quality of the clones due to generating better adversarial samples for training. Therefore, in this study, we upgraded T-GAN (which had just one major "Turing" quality component for generated samples) towards Cloning-GAN, which combines several important quality criteria ("Uniformity", "Challenge", "Hostility" in addition to "Turing"). We have also suggested several alternatives for balancing these conflicting criteria, aiming to improve each particular case of cloning. Therefore, we consider this study as an important puzzle in the process of finding suitable and strong solutions for digital cloning. The analytics and architectures presented in this paper still need thorough testing with complex industrial cases and big industrial data, which will be a subject of our future research.

Within the context of Industry 4.0 and Smart Manufacturing, the Cloning-GAN-driven solutions could have practical applications in optimizing manufacturing processes and enhancing human-machine collaboration. One practical application could involve improving the efficiency of quality control in manufacturing. By training operational clones of experienced quality control experts, one could develop models that quickly identify defects and deviations in products on the production line. These clones would inherit the expertise and biases of the original experts, leading to more accurate and consistent identification of quality issues. Another application could focus on predictive maintenance. Operational clones of maintenance specialists could be trained to predict equipment failures and recommend maintenance actions. These clones would learn from the insights of skilled maintenance professionals, incorporating their decision-making processes and biases. As a result, the manufacturing facility could proactively address maintenance needs, minimizing downtime and optimizing equipment performance. In the context of humanrobot collaboration, operational clones of experienced workers could guide robots in performing complex tasks. The clones could transfer their expertise to the robots, allowing them to replicate human-like decision-making processes and adapt to unforeseen challenges on the factory floor. This would lead to more flexible and effective collaboration between humans and machines, contributing to increased productivity and overall process optimization.

5. Conclusions

A highly automated and human-centric future of smart manufacturing requires process modeling, simulation, and control based on digital replicas of physical objects and smart entities, including humans. In this paper, we adapt and modify traditional GAN architecture to be used for cognitive cloning of hidden decision-making capability of a human or another smart model. The suggested Cloning-GAN architecture is based on new objectives and philosophy. In traditional GAN, the purpose is a perfect **GENERATOR**, and the **DISCRIMINATOR** is just a facilitator in training such a perfect **GENERATOR**. Traditional GAN converges when the **DISCRIMINATOR** will not be able anymore to distinguish between the real samples (from the target distribution) and the generated samples and only "guesses" (fake or not) like "coin flipping" (i.e., the convergence (or "equilibrium") point here means that the average loss of both **GENERATOR** and **DISCRIMINATOR** will become close to 0.5). In the case of Cloning-GAN, the target is a perfect **CLONE**. The **GENERATOR** plays a role of a "challenger" and a "facilitator" for the **CLONE** training process by creating adversarial and unlabeled training data. The **DONOR** plays a role of a "supervisor" by providing labels to the generated data (i.e., both (trainable) **GENERATOR** + (untrainable) **DONOR** are collaboratively contributing to creating the best training data for **CLONE**). This new type of GAN converges when the **CLONE** is able to copy **DONOR** with minimal (close to 0) loss, whatever challenges **GENERATOR** is continuing to invent. The convergence (equilibrium) point here means that the loss of **CLONE** is almost zero, while the loss of **GENERATOR** reaches some value beyond 0.5.

In the cloning case, we do not have a target distribution to capture (like in traditional GAN) because the "reality" in this case is represented not by some set of data but by some hidden function (i.e., **DONOR**) to be learned and copied like in data-free knowledge distillation tasks. Therefore, we just need to trade-off between the diversity and challenge of the generated data in a way to get as good as possible training data for the **CLONE** to learn faster.

Yes, Cloning-GAN inherits (from typical GAN) the process of incremental synchronous training of two adversaries (in our case, **GENERATOR** vs **CLONE**). Therefore, the capable data **GENERATOR** and the desirable **CLONE** are co-evolving towards perfectness during training. However, comparably to the traditional GAN, the Cloning-GAN architecture is biased more towards the cloning rather than the generation objective.

Suggested architecture is a kind of adversarial knowledge distillation enabler facilitated by advanced multicriteria **GENERATOR** (B-C-GAN or D-C-GAN architectures) or by multi-**GENERATOR** (3M-C-GAN architecture). However, our overall objective is not related to optimizing the effectiveness and efficiency of classification models or addressing the data privacy issues like in traditional knowledge distillation, but rather getting a functional copy (aka digital "clone") of some hidden phenomena or cognitive behavior (like human decision-making), including its strengths and weaknesses, perfectness and imperfectness, biases and ignorance, etc., without taking care about its performance. Therefore, we use the term (digital, cognitive) "cloning" instead of "distillation" in our research.

References

[1] Krishnamurthi, R., and Kumar, A. (2020). "Modeling and Simulation for Industry 4.0". In: A. Nayyar, A. & A. Kumar (Eds.), *A Roadmap to Industry 4.0: Smart Production, Sharp Business and Sustainable Development* (pp. 127-141). Springer. https://doi.org/10.1007/978-3-030-14544-6_7

[2] Stavropoulos, P., and Mourtzis, D. (2022). "Digital twins in industry 4.0". In: *Design and Operation of Production Networks for Mass Personalization in the Era of Cloud Technology* (Chapter 10, pp. 277-316). Elsevier. https://doi.org/10.1016/B978-0-12-823657-4.00010-5

[3] Wang, B., Zheng, P., Yin, Y., Shih, A., and Wang, L. (2022). "Toward human-centric smart manufacturing: A human-cyber-physical systems (HCPS) perspective". *Journal of Manufacturing Systems*, **63**: 471-490. https://doi.org/10.1016/j.jmsy.2022.05.005

[4] Longo, F., Padovano, A., De Felice, F., Petrillo, A., and Elbasheer, M. (2023). "From "prepare for the unknown" to "train for what's coming": A digital twin-driven and cognitive training approach for the workforce of the future in smart factories". *Journal of Industrial Information Integration*, **32**: 100437. https://doi.org/10.1016/j.jii.2023.100437

[5] Gazzaneo, L., Padovano, A., and Umbrello, S. (2020). "Designing smart operator 4.0 for human values: a value sensitive design approach". *Procedia Manufacturing*, **42**: 219-226. https://doi.org/10.1016/j.promfg.2020.02.073

[6] Golovianko, M., Gryshko, S., Terziyan, V., and Tuunanen, T. (2022). “Responsible cognitive digital clones as decision makers: A design science research study”. *European Journal of Information Systems*. https://doi.org/10.1080/0960085X.2022.2073278

[7] Terziyan, V., Gavriushenko, M., Girka, A., Gontarenko, A., and Kaikova, O. (2021). ”Cloning and training collective intelligence with generative adversarial networks”. *IET Collaborative Intelligent Manufacturing*, **3(1)**: 64-74. https://doi.org/10.1049/cim2.12008

[8] Lee, J., Azamfar, M., Singh, J., and Siahpour, S. (2020). “Integration of digital twin and deep learning in cyber-physical systems: towards smart manufacturing”. *IET Collaborative Intelligent Manufacturing*, **2(1)**: 34-36. https://doi.org/10.1049/iet-cim.2020.0009

[9] Liu, G., Shen, W., Gao, L., and Kusiak, A. (2022). “Knowledge transfer in fault diagnosis of rotary machines”. *IET Collaborative Intelligent Manufacturing*, **4(1)**: 17-34. https://doi.org/10.1049/cim2.12047

[10] Gou, J., Yu, B., Maybank, S. J., and Tao, D. (2021). “Knowledge distillation: A survey”. *International Journal of Computer Vision*, **129**: 1789-1819. https://doi.org/10.1007/s11263-021-01453-z

[11] Wang, X., Zhang, R., Sun, Y., and Qi, J. (2018). “KDGAN: Knowledge distillation with generative adversarial networks”. *Advances in neural information processing systems*, **31**. https://proceedings.neurips.cc/paper_files/paper/2018/file/019d385eb67632a7e958e23f24bd07d7-Paper.pdf

[12] Terziyan, V., Gryshko, S., and Golovianko, M. (2018). “Patented intelligence: Cloning human decision models for Industry 4.0”. *Journal of Manufacturing Systems*, **48** (Part C): 204-217. Elsevier. https://doi.org/10.1016/j.jmsy.2018.04.019

[13] Golovianko, M., Gryshko, S., Terziyan, V., and Tuunanen, T. (2021). Towards digital cognitive clones for the decision-makers: Adversarial training experiments. *Procedia Computer Science*, **180**: 180-189. Elsevier. https://doi.org/10.1016/j.procs.2021.01.155

[14] Branytskyi, V., Golovianko, M., Gryshko, S., Malyk, D., Terziyan, V., and Tuunanen, T. (2022). “Digital clones and digital immunity: Adversarial training handles both”. *International Journal of Simulation and Process Modelling*, **18(2)**: 124-139. https://doi.org/10.1504/IJSPM.2022.10048910

[15] Aggarwal, A., Mittal, M., and Battineni, G. (2021). “Generative adversarial network: An overview of theory and applications”. *International Journal of Information Management Data Insights*, **1(1)**: 100004. https://doi.org/10.1016/j.jjimei.2020.100004

[16] Endres, D. M., and Schindelin, J. E. (2003). “A new metric for probability distributions”. *IEEE Transactions on Information Theory*, **49(7)**: 1858-1860. https://doi.org/10.1109/TIT.2003.813506

[17] Bhattacharyya, P., and Chakrabarti, B. K. (2008). “The mean distance to the *n*th neighbour in a uniform distribution of random points: An application of probability theory”. *European Journal of Physics,* **29(3)**: 639-645. https://doi.org/10.1088/0143-0807/29/3/023

[18] Sharma, R., Biookaghazadeh, S., and Zhao, M. (2018). “Are existing knowledge transfer techniques effective for deep learning on edge devices?”. In: *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* (pp. 15-16). https://doi.org/10.1145/3220192.3220459

[19] Buciluă, C., Caruana, R., and Niculescu-Mizil, A. (2006). “Model compression”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 535-541). https://doi.org/10.1145/1150402.1150464

[20] Zhu, Z., Hong, J., and Zhou, J. (2021). “Data-free knowledge distillation for heterogeneous federated learning”. *Proceedings of Machine Learning Research*, **139**: 12878-12889. http://proceedings.mlr.press/v139/zhu21b/zhu21b.pdf

[21] Ba, J., and Caruana, R. (2014). “Do deep nets really need to be deep?”. *Advances in Neural Information Processing Systems*, **27**. https://proceedings.neurips.cc/paper_files/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf

[22] Chen, D., Mei, J. P., Wang, C., Feng, Y., and Chen, C. (2020). “Online knowledge distillation with diverse peers”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 4, pp. 3430-3437). https://doi.org/10.1609/aaai.v34i04.5746

[23] Li, L., and Jin, Z. (2022). “Shadow knowledge distillation: Bridging offline and online knowledge transfer”. *Advances in Neural Information Processing Systems*, **35**: 635-649. https://proceedings.neurips.cc/paper_files/paper/2022/file/040d3b6af368bf71f952c18da5713b48-Paper-Conference.pdf

[24] Zhang, L., Bao, C., and Ma, K. (2021). “Self-distillation: Towards efficient and compact neural networks”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **44(8)**: 4388-4403. https://doi.org/10.1109/TPAMI.2021.3067100

[25] Micaelli, P., and Storkey, A. J. (2019). “Zero-shot knowledge transfer via adversarial belief matching”. *Advances in Neural Information Processing Systems*, **32**. https://proceedings.neurips.cc/paper_files/paper/2019/file/fe663a72b27bdc613873fbbb512f6f67-Paper.pdf

[26] Heo, B., Lee, M., Yun, S., and Choi, J. Y. (2019). “Knowledge distillation with adversarial samples supporting decision boundary”. In: *Proceedings of the AAAI Conference* (Vol. 33, No. 1, pp. 3771-3778). https://doi.org/10.1609/aaai.v33i01.33013771

[27] Zhao, H., Sun, X., Dong, J., Manic, M., Zhou, H., and Yu, H. (2022). “Dual discriminator adversarial distillation for data-free model compression”. *International Journal of Machine Learning and Cybernetics*, **13**: 1213–1230. https://doi.org/10.1007/s13042-021-01443-0

[28] Terziyan, V. (2008). “SmartResource–proactive self-maintained resources in semantic web: Lessons learned”. *International Journal of Smart Home*, **2(2)**: 33-57. Global Vision Press. http://dx.doi.org/10.14257/ijsh.2008.2.2.03

[29] Katasonov A., Kaykova O., Khriyenko O., Nikitin S., and Terziyan V. (2008). ”Smart semantic middleware for the internet of things”. In: *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics* (vol. 1, pp. 169-178). https://doi.org/10.5220/0001489001690178

[30] Terziyan, V., and Katasonov, A. (2009). “Global Understanding Environment: Applying semantic and agent technologies to industrial automation”. In: *Emerging Topics and Technologies in Information Systems* (pp. 55-87). IGI Global. https://doi.org/10.4018/978-1-60566-222-0.ch003