Leveraging Synthetic Data for DNN-based Visual Analysis of Passenger Seats

Nerea Aranjuelo[1,2*], Jose Luis Apellaniz[1], Luis Unzueta[1], Jorge Garcia[1], Sara Garcia[1], Unai Elordi[1,2] and Oihana Otaegui[1]

[1*]Vicomtech, Basque Research and Technology Alliance (BRTA), San Sebastian, Spain.
[2*]Basque Country University (UPV/EHU), San Sebastian, Spain.

*Corresponding author(s). E-mail(s): naranjuelo@vicomtech.org;
Contributing authors: jlapellaniz, lunzueta, jgarciac, sgarcia, uelordi, ootaegui@vicomtech.org;

Abstract

Deep Neural Network (DNN)-based vision systems could improve passenger transportation safety by automating processes such as verifying the correct positioning of luggage, seat occupancy, etc. Abundant and well-distributed data are essential to make DNNs learn appropriate pattern recognition features and have enough generalization ability. The use of synthetic data can reduce the effort of generating varied and annotated data. However, synthetic data usually present a domain gap with real-world samples, that can be reduced with domain adaptation techniques. This paper proposes a methodology to build simulated environments to generate balanced and varied synthetic data and avoid including redundant samples to train classification DNNs for passenger seat analysis. We show a practical implementation for detecting whether luggage is correctly placed or not in an aircraft cabin. Experimental results show the contribution of the synthetic samples and the importance of correctly discarding redundant data.

Keywords: Simulated Environments, Synthetic Data, Deep Neural Networks, Video Surveillance, Domain Gap

1 Introduction

In recent years, the irruption of Deep Neural Networks (DNNs) has generated significant gains in various computer vision tasks: object detection [1], object segmentation [2], image captioning and visual relationship detection [3] or action recognition [4]. DNNs are Artificial Neural Networks (ANNs) with more than one hidden layer; therefore, many parameters need to be tuned by a learning algorithm. This learning process requires a lot of data, especially when the neural network starts far from the required solution, which is the usual case. Collecting such data can be challenging specifically for two principal reasons.

First, the data collection process should comply with privacy-related regulations (e.g., EU's GDPR). This process should put in place appropriate technical and organizational measures to implement the data protection principles, such as data protection or data anonymization.

Second, the time and cost we need to devote to the data collection process, considering that the quantity is substantial and the variety and the required balance in labeled visual appearances to cover all possibilities during training. Some data may even penalize the DNN training results if we do not consider these characteristics.

In most cases, we cannot extract data from the Internet since we try to solve a particular problem in a specific scenario with a particular camera setup. Thus, we need to create custom datasets. It implies executing several actions such as dataset specification, camera setup installation, technical preparation, recording protocols, and even actors simulating particular guide notes. All these actions, together with the well-known problem of labeling data [5] due to the lack of effective tools and the annotation complexity, make the dataset generation process more challenging.

There are several ways to extend training data: (i) augmentation techniques [6], (ii) domain adaptation techniques and [7] (iii) synthetic data generation [8]. Augmentation techniques or domain adaptation approaches are suitable for those cases in which we already have some annotated data corresponding to the domain. Alternatively, before we start collecting real data, we could tackle the initial stages of the system's design by generating some synthetic data. They would allow us to train initial versions of DNNs that could help us annotate real data, start applying the other mentioned techniques, and progressively improve the system's reliability [9]. However, the domain gap limits the real-world performance of machine learning models trained only with synthetic data. Closing the gap is important for effectively using synthetic data or data from different domains.

As stated in [8], we could generate synthetic data following different kinds of strategies, such as compositing real data, relying on generative models, or using simulated environments. The latter has an additional advantage when building multi-camera systems for surveillance applications: they typically provide virtual cameras to help set up the camera system to cover the use cases in the targeted real scenario. This feature is particularly relevant in scenarios not easily accessible to system designers and when the camera positions and lens

characteristics are not predefined. These challenges arise, for example, designing multi-camera intelligent vision systems for passenger transportation safety, which is the main motivation of our work. Our final goal is to build intelligent vision systems for automated passenger seat analysis in different types of transport (e.g., aircraft, trains, buses, cars, etc.). In these scenarios, the vision system could be designed to detect seat occupancy, whether luggage is correctly positioned, the passenger's body pose, whether seat belts are fastened, whether tray tables are in the stowed position, etc. We consider DNNs to solve these kinds of problems as they are currently the kind of Machine Learning techniques that obtain the best accuracy results for image classification problems with complex scenes such as those that could be captured in such types of scenarios.

However, there are no general simulated environments ready to solve the camera setup and data generation tasks for all scenarios and use cases like those we consider. Typically, we need to build 'ad hoc' environments. On the contrary, a more flexible solution should allow: (1) including the required scenario-related graphical assets in an easy manner, (2) configuring context- and use-case-based scenes with user-friendly parameters, (3) capturing images from virtual camera viewpoints quickly, including camera-related effects, such as the geometric distortion introduced by the lenses, and (4) generating a broad and balanced range of plausible situations of interest randomly, applying suitable noise to the labeled data for the appropriate training of DNNs.

Generating a random wide range of situations may improve the DNN generalization ability and minimize the problem of the domain gap between the synthetic and real data domains [10]. However, if both real and synthetic samples are available for the training, we can adopt additional strategies to reduce their domain gap and guarantee the contribution of the synthetic samples.

To respond to all these challenges, in this work, we present a methodology to build an appropriate training dataset for training DNNs using both real and synthetic samples for automated passenger seat visual analysis, extending the approach presented in [11]. More specifically, the contributions of our work are the following:

- A procedure to build simulated environments to generate balanced and varied synthetic data.
- A procedure to discard redundant samples that may penalize the DNN accuracy.
- A practical implementation example for detecting luggage's correct or incorrect positioning in aircraft cabins.
- A qualitative and quantitative analysis of the data generation process compared to alternative approaches.
- An evaluation of the accuracy improvement obtained with our approach compared to other alternatives when training Domain Adversarial Networks (DANNs) with real and synthetic data.

The rest of the paper is organized as follows: section 2 describes prior works related to simulated environments, real data acquisition, and DNN training with real and synthetic data; section 3 explains our proposed methodology; section 4 presents the mentioned practical implementation example and experiments; finally, section 5 presents the conclusions and future lines of work.

2 Related Work

2.1 Synthetic Data Generation

The recent survey on synthetic data for deep learning [8] shows that in the last years there has been a shift from static synthetic datasets to interactive simulation environments, grouped in the following categories: (1) outdoor urban environments for learning to drive, (2) indoor environments and (3) robotic and aerial navigation simulators. Current state-of-the-art environments have been built upon *Grand Theft Auto V (GTA V)* [12], *Unity3D* [13, 14], *Unreal Engine* [15, 16], *CityEngine* [17] or *Blender* [18], among others.

GTA V is an action-adventure video game with realistic graphics of a large detailed city and surrounding areas from which we could extract diverse data, involving virtual people, animals, cars, trucks, motorbikes, planes, etc. Saleh et al. [13] state that the main drawback of video-game-based environments is the limited freedom for customization and control over the scenes to be captured. These limitations make obtaining a large diversity and good balance of classes difficult due to the complicated procedure required to obtain ground-truth instance-level annotations. On the contrary, they claim that their environment, built upon the game engine *Unity3D*, can be set up by one person in one day. In comparison, much less effort as it allows access to a virtually unlimited number of annotated images with the object classes of state-of-the-art real urban scene datasets. It captures synthetic images and instance-level semantic segmentation maps simultaneously and in real-time, with no human intervention. While generating the data, their system renders the original textures and shaders of included 3D objects and other automatically created unique ones for their corresponding instances. *Unity3D* is also the basis of the indoor environment proposed by [14] for the generation of synthetic data for object detection from an omnidirectional camera placed on the ceiling of a room. It generates the 3D assets using the skinned multi-person linear model proposed in [19]. This work points out that *Unity3D* only provides a camera model for perspective and orthographic projection. They overcome this limitation by combining four perspective cameras following the procedure proposed in [20].

Lai et al. [15] proposed a universal dataset and simulator of outdoor scenes such as pedestrian detection, patrolling drones, forest fires, shooting, and more. It is powered by the *Unreal Engine* and leverages the *AirSim* [16] plugin for hardware simulation. As it focuses on training dynamic systems relying on deep reinforcement learning [21], it prioritizes the real-time interactivity of the virtual agents over photorealistic rendering. Therefore, it differs from static systems for surveillance applications, like those motivating this work. In our

case, achieving a better rendering quality for the data generation process is more important than the real-time interactivity during training.

CityEngine is a program that allows generating 3D city-scale maps procedurally from a set of grammar rules. It is used in [17] as part of their method to generate an arbitrarily large, semantic segmentation dataset reflecting real-world features, including varying depth, occlusion, rain, cloud, and puddle levels, while minimizing required effort.

Blender is an open-source 3D graphics software with Python APIs that facilitate loading 3D models and automating the scene rendering. Rajpura et al. [18] used it to generate a dataset to train a DNN-based detector for recognizing objects inside a refrigerator. It used *Cycles Render Engine* available with *Blender*. This engine is a physically-based path tracer that allows getting photorealistic results, which is beneficial for avoiding a big domain gap between the feature distribution of synthetic and real data domains.

2.2 Real Data Acquisition

Big amounts of data have been crucial for DNNs' success. Large datasets such as ImageNet [22] or OpenImages [23] have played an important role in the advances and development of DNNs. However, generalist datasets do not cover all possible computer vision tasks for all possible use-cases. Smart video surveillance systems, for example, may need to cover specific use cases or camera perspectives that are not available in any open dataset. Training techniques that rely on pretrained DNN models, such as transfer learning, fine-tuning, or BiT [24], can be used to benefit from models already trained with generalist datasets and avoid collecting and annotating big amounts of data. Moreover, works like [25–27] suggest that pretraining on domain-specific datasets leads to improved accuracy compared to pretraining with generalist datasets

Other techniques, such as few-shot learning [28], try to train DNNs using a limited number of samples per target category. However, we would need to replicate the camera system to record the target scenes or situations. In these situations, usually, we would capture redundant samples, for example, because the captured scene remains the same for some time. Besides, redundancy could also arise when gathering data from different sources.

Images with the same content add less additional information than images with new content, but beyond this, they can be problematic during the DNN training. The redundant data can lead to model overfitting during the training process.

2.3 DNN Training with Synthetic and Real Data

The domain gap that data from different domains present is a challenge for training DNNs. Training a model with data from a source domain and then using it with data from another domain often leads to poor accuracy. This domain gap is especially noticeable when working with real and synthetic data. Domain adaptation can be defined as a type of transfer learning where

there are two domains with different data distributions. This problem has been researched in computer vision using various strategies to reduce or deal with the difference between the two domains.

The most basic method is to first train on the synthetic domain and then fine-tune the model on the real domain [29]. An alternative method is to jointly train with both domains' images by using mini-batches from source and target domains [30, 31]. These methods require having an adequate number of annotated real samples for optimal accuracy.

Other works apply the domain randomization technique to reduce the domain gap's effects when training [10]. During the data simulation process, they use random parameters (e.g., random light, objects' pose or texture) to generate non-realistic images, which force the model to learn the essential features of the objects of interest. Some other works apply image augmentations to the generated samples instead of focusing on the 3D scene generation and rendering step. [32] generate new augmented synthetic samples using a semi-supervised errors-guide method to improve the DNN accuracy on cross-domain datasets.

Another way to solve the domain differences is the image-to-image translation method. Several works have emerged about Generative Adversarial Networks (GANs), which try to convert an image from one domain to another different one [33]. These methods are making remarkable advances, but they tend to have some difficulties accurately preserving the image's structure. Rather than forcing the synthetic samples to look more like the real ones, there is another research line that focuses on extracting domain-invariant features [34]. These methods investigate how DNNs can be trained with different domains' data better. These approaches commonly consist of two main steps: (i) learn features that minimize the target task loss and (ii) make the features from both domains as indistinguishable as possible to make models trained in one domain work correctly in a target domain.

3 Proposed Methodology

3.1 Synthetic Data Generation

Figure 1 shows the architecture of the proposed methodology for generating synthetic data for training deep learning models. The input data are the 3D assets of the scene and the scene's description file. The outputs are the synthetic images and the annotations for training DNNs. The principal modules in our approach are: (i) the scene manager who is responsible for loading the scene configuration; (ii) the engine which sets up the 3D scene according to the provided configuration and generates the training images by rendering different camera viewpoints and; (iii) the label generator, which generates an output file containing the annotations corresponding to the generated images.

As it happens in the data collecting process carried out in a real environment, the first step is gathering all the necessary 3D graphical assets to reproduce the scene of interest. Specifically, two different groups of graphical

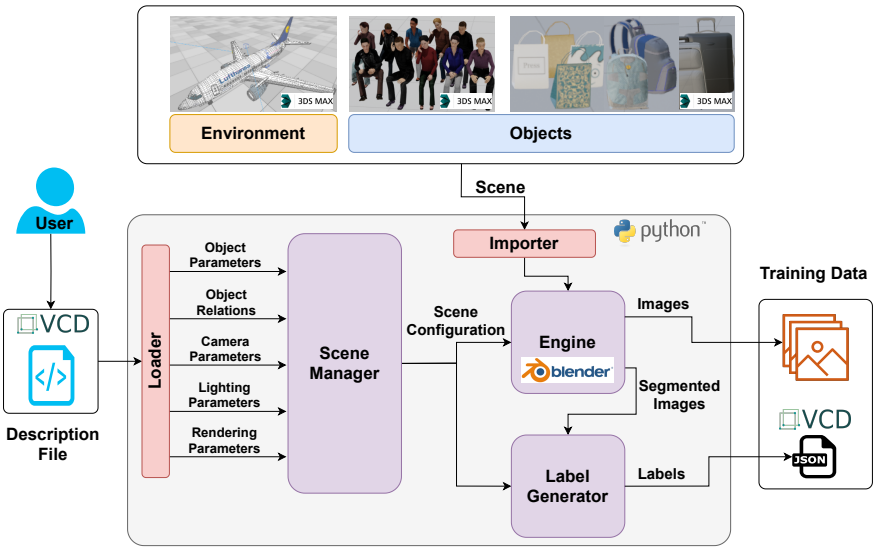

Fig. 1 Software architecture of the proposed methodology for creating synthetic training data for vision-based systems. [11]

assets are required. The first type contains those graphical assets representing the environment (i.e., the scenario where we should accomplish the recordings). We assume that such assets belonging to the environment are static since they represent the background. The second group contains those graphical assets representing the dynamic objects of the scene. Combining the locations of these assets, their poses, sizes, appearances, and the variations of the lighting sources and the camera properties will provide a wide variety for generating our custom training data.

For use cases where some graphical assets are unavailable, the user should create them using 3D modeling software applications. We use *3DS Max* but other options such as *Autodesk Maya*, *Lightwave-3D*, *Vectary*, *Blender*, etc. are also suitable. Depending on the complexity, additional plugins such as *Populate* (for *3DS Max*) can alleviate the effort of designing the objects. Working with a very detailed 3D model can become a challenge due to the vast number of polygons and materials the render engine has to process. This is directly related to the rendering time of the scene, and it could be a troublesome bottleneck in the data flow when a user tries to generate thousands of samples. However, using very detailed 3D assets or more lightweight ones should be considered based on the scope of the use case.

Once all the scene assets are designed, our method allows us to configure different camera setups, placing objects at multiple locations and poses, having various illumination sources, or configuring the rendering parameters by a minimum user interaction. Furthermore, the user can easily define multiple combinations of parameters for generating different training data samples in a single iteration.

Specifically, the user has to define a configuration file in which the parameterization of the scene is specified in a user-friendly way. Then, a loader interprets the content related to the different entities according to the nature of the parameters (objects, cameras, illumination, rendering). Such information is received by the scene manager, which interprets and handles these data to build the scene configuration for the user-defined sequence. This configuration is used to replicate the target scenes in the 3D synthetic environment, which implies loading the environment and dynamic assets with the corresponding configurations and setting all the cameras, lighting, and rendering parameters to get the desired results. Then, the engine renders the images from the defined camera perspectives. We use *Blender* for this purpose, although the same methodology could be applied using similar alternative programs.

The label generator is in charge of generating the corresponding annotations based on generated segmentation masks. The user can choose different annotation types depending on the target computer vision task (e.g., object detection or semantic segmentation).

3.1.1 Scene Management

The synthetic scene is replicated based on the information provided by the user. The user is in charge of the configuration through the description file. For this purpose we adopt the Video Content Description (VCD) structured JSON-schema file format [35]. VCD is an open-source metadata structure able to describe complex scenes, including annotations and all the needed scene information, in a very flexible way. In addition, it allows an effortless and fast user interaction for the file generation process. The VCD format is compatible with the OpenLABEL standard [36]. The configuration file contains the cameras and lighting setup information, the 3D assets in the scene and the relation between them, and the rendering parameters. Modifying something in the scene, such as the position of an object or the camera specifications, is done by changing the corresponding field in the configuration file.

3.1.2 Camera Setup and Lighting Sources

The camera setup controls how the scenario and the objects are represented in the 2D images. In order to obtain realistic training data, the virtual cameras should simulate the same properties of the sensor and the lens as the expected cameras, which will be installed in the real environment. One of the major benefits of using *Blender* as an engine for creating and rendering the scene, in contrast with others such as *Unity3D*, is the multiple choices of camera models. In particular, we can generate the image projection of the virtual camera by using an orthographic model, a perspective model, or a panoramic model. Each camera model has different properties and effects on the resulting image (e.g., distortion), so the model needs to be selected depending on the target application. These models allow emulating any combination of the image sensor and lens type mounted in a real camera. Table 1 shows the parameters that need to be added to the configuration file to add as many cameras as

desired with their corresponding parameters. More specifically, the user needs to define the camera's position regarding the coordinate origin, the camera's orientation, the sensor size, the field of view, and the focal length.

Table 1 Camera Parameters

Camera model	Camera model (different distortion types).
Resolution	Output image resolution (pixels).
Position	Sensor position (m) in X, Y and Z axis.
Orientation	Sensor orientation (degrees) in X, Y, and Z axis.
Size	Sensor size (mm).
FOV	FOV of the sensor (degrees).
Focal length	Focal length of the sensor (mm).
Custom params	Extra params defined by the user.

Another important factor affecting the projection of the visual information from 3D to 2D is the scene's lighting. Depending on the target scenario, the user may want to add light coming from single points which emit light in all directions or from spots with a single direction (e.g., indoor lamps) or outdoor lighting simulating the sun. Table 2 shows the parameters that should be defined in the configuration file to add as many different light sources as desired to the 3D environment.

Table 2 Lighting Parameters

Light model	Type (point, spot, sun).
Position	Light position (m) in X, Y and Z axis.
Orientation	Light orientation (degrees) in X, Y and Z axis.

The scene configuration is automatically replicated in the 3D environment from the description file. During this step, the scene configuration is exported as a *Blender* project for those cases in which the user wants to explore the camera setup interactively. This way, it can be used as an interactive tool that helps to design a proper setup for a target application. The users can modify the intrinsic and extrinsic camera parameters, their positions, or even the number of needed cameras. At the same time, they visualize the images that would be captured. Simulating a specific setup with no need to deploy it physically can help avoid wrong decisions that lead to a not optimal or lousy configuration.

This way, it may help define the appropriate number of cameras, their locations, poses, and viewpoints. Thus, the proposed methodology includes the following two bidirectional features:

- VCD2Scene: The user defines the VCD description file, and the scene is replicated in the 3D environment, including the 3D assets, camera configurations, and lighting sources.
- Scene2VCD: The user loads a 3D scene, and after making the desired modifications, then exports the new setup to a VCD description file.

3.1.3 3D Assets in the Scene

The 3D assets' configuration in the scene is also defined in the VCD file. The user can add as many objects as desired to the scene in specific configurations. These objects should belong to the available 3D asset types (e.g., humans, cars). Then the 3D environment simulator interprets this information through the local relation between the assets. The user should have previously defined the relations present in the scene and interactively selected the positions they would belong to. For example, if the target application is about detecting abandoned objects in an airport, some local relations that would be necessary could be "on" or "below." These relations would let the user relate different assets, for example, by describing that particular objects (e.g., a bag, a suitcase) are *on* a desk or *below* the waiting seats. At the same time, the environment simulator would relate these positions with the user-selected positions. In addition, the user defines in the VCD file the time interval when each specific asset is present in the scene in the described configuration. This method provides high flexibility for the user to generate various configurations in a swift and user-friendly way.

In order to add a higher degree of variety to the generated data, when each asset is placed in a specific position, some random noise is added to slightly perturb its position and orientation. In addition, the user can apply random colors to the 3D assets to include more diversity in the data or maintain the original textures.

3.1.4 Rendering Parameters

The rendering step turns the 3D scene into the output 2D image. The configuration of the rendering affects both the image quality and rendering time. The optimum configuration is closely related to the target task requirements and the available hardware. Therefore, our approach lets the user change the most influential parameters (shown in Table 3) in the configuration file. The user can choose between the available rendering engines (in the case of *Blender* there are three available engines), computing caustics or not, the rendering tile size, the device to be used (CPU or GPU), and the maximum allowed light bounces. When the light hits a surface, it bounces off the surface and hits another one, and then the process is repeated. This is very expensive in terms of rendering time. Decreasing the maximum bounces implies limiting the number of times a ray can bounce before it is killed, reducing the time spent computing rays. Depending on the scene and the task, the user can adjust this parameter to get the desired output.

Table 3 Rendering Parameters

Device	Device used for rendering (CPU/GPU).
Engine	Engine type used for rendering.
Tile	Area of the image considered during the rendering.
Bounces	Maximum light bounces to be applied.
Caustics	Caustics computation.

3.1.5 Labeled Data Generation

Training machine learning models in a supervised way implies collecting the needed data and the corresponding annotations. Annotating the data is an expensive process for which synthetic data generation can be beneficial, thanks to automatic label generation. Our approach provides flexible annotations with different levels of detail depending on the target computer vision task (object detection, object segmentation, or visual relationship detection).

We opt for rendering instance-level masks. Each asset in the simulated scene is given a unique ID apart from the object class ID it belongs to. These instance IDs are used to compute an alpha mask per object. These masks are combined in a single segmentation mask. When the data generation starts, the synthetic images are rendered simultaneously as the instance-level semantic segmentation maps.

The segmentation masks are used to generate the annotations stored in the output VCD file. This file contains both the input configuration data and the annotations. By default, the masks are used to get the minimum bounding boxes containing each object's pixels, represented in the VCD file by each box's corner coordinates. These annotations can be used to train object detectors. However, our approach can also be configured to save the objects' segmentation masks. These data are already in the instance-level masks and are stored as object contours, which are described as polygons in the output file. In addition, the output VCD file contains the description of the relations between the assets in the scene. These data are complemented with the objects' bounding boxes. These annotations can be used to train visual relationship detectors. The annotations need to be parsed to the required format depending on the chosen deep learning framework (e.g., *TensorFlow*, *PyTorch*).

The annotations in the output VCD file are stored per object and frame-wise. Consequently, each annotation is stored along with its corresponding image path.

3.2 Real Data Acquisition

The synthetic environment built following the proposed method helps to design an appropriate camera system for a smart video surveillance system (Section 3.1.2). Once the virtual camera setup is done, it can be replicated for real data acquisition.

When capturing real data for video surveillance systems, it is common to get some very similar or redundant images, especially in scenarios where

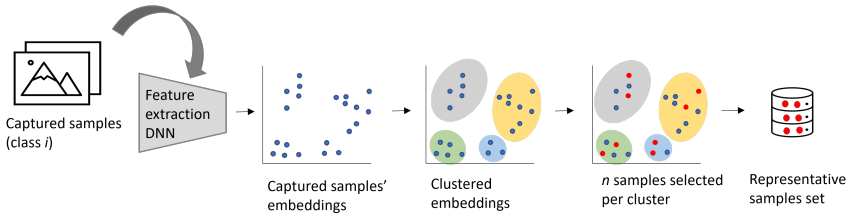

Fig. 2 Clustering-based pipeline to select samples of interest from all the captured data of each target class.

events happen from time to time and stay the same the rest of the time. The advantage of capturing data continuously is that skipping consecutive frames may reduce data redundancy. However, this process can exclude interesting samples as it only relies on time consistency but does not consider the content of the images. We propose an alternative process for selecting varied samples in Figure 2. The goal of the pipeline is to choose specific samples of interest to train the target DNN.

For that purpose, we do the following steps:

- Use a pretrained generalist DNN's backbone as a feature extractor to encode all the images in image embeddings (m-dimensional vectors).
- Cluster the embeddings using a clustering algorithm, such as K-Means.
- Select a specific number of n samples randomly from each cluster to generate the new subset of images.

The optimum number of clusters to be used in a clustering algorithm such as K-Means can be computed using the Elbow method. This method consists of plotting the sum of squared distances from each point to its assigned center as a function of the number of clusters, and selecting the elbow of the curve as the optimum number of clusters. The process is applied to all the images of a specific target category i so that a balanced dataset is generated. The number of samples per cluster depends on the target size of the dataset.

Once the samples are selected, they are combined with the rest of the synthetic data to complete the final dataset.

4 Practical case and experiments

In order to validate the proposed methodology, we apply it to a real problem in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based smart sensing system. Currently, verifying Taxi, Take-off, and Landing (TTL) requirements in aircraft cabins is a manual process, which entails an increased workload for the crew, operational inefficiencies, and the risk of human errors in handling safety-related procedures. One of the requirements the cabin crew members must check is that all the luggage is correctly placed in each TTL phase. During these phases, the luggage should not be situated so that an emergency evacuation of the aircraft would be delayed or hindered. Figure 3 shows the allowed and not allowed positions for the cabin

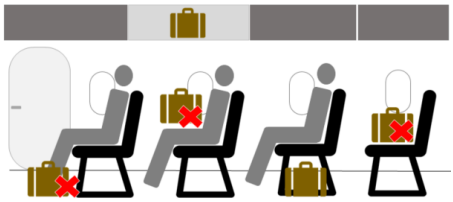

Fig. 3 Authorised positions for luggage during TTL. [11]

luggage during TTL. The verification done by the crew members could be automated with the development of a vision-based system. This system would be beneficial in terms of operational efficiency and safety. Developing a system capable of detecting the luggage positions in the cabin entails designing an appropriate camera setup and generating suitable and enough training data for the corresponding machine learning models. This setup could automate additional tasks, such as checking the seat occupancy or whether tray tables are in stowed positions.

4.1 3D Assets for the Synthetic Environment

To address this problem, the first step of our methodology is the generation of the assets for the 3D synthetic environment. We first generate all the involved assets for the scene of interest. In this case, we model 22 different object types to simulate typical cabin luggage (e.g., backpacks, magazines, laptops), a cabin model representing a Boeing 737 aircraft (with 19 seats), and a group of different human models with various poses and appearances for the seated passengers. The generated 3D assets are shown in Figure 4.

Fig. 4 Generated 3D assets for the use case: passengers, cabin luggage, and aircraft cabin.

4.2 Cabin Camera Setup Design

Our feature VCD2Scene allows loading the aircraft model within some of the modeled 3D assets using an initial configuration file and visualizing the 3D scene from the virtual camera viewpoints for the camera setup task. The initial setup idea could be to place some cameras on top of the seats to control the luggage in these areas (e.g., backpacks partially below the seats) and other cameras above the corridor to verify a clear exit. However, how many cameras should be installed? Where should they be to guarantee the system can see

every seat without occlusions? What lens parameters should these cameras have?

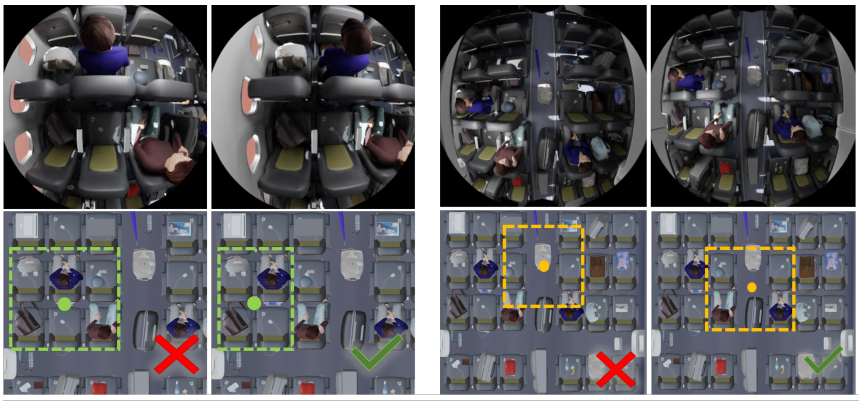

Fig. 5 Captured scene from different camera positions in the seats (left) and corridor (right) areas. Some of the configurations are discarded because of occlusions.

We interactively change the virtual cameras' extrinsic and intrinsic parameters to check the results we would obtain with each configuration to answer these questions. We move the camera positions and orientations to guarantee that the minimum number of cameras captures all the regions of interest.

Figure 5 (left) shows an example of accepted and discarded camera positions for capturing the luggage in the seat areas. At first, we could think the cameras should be on top of the middle seats to capture the status of 6 individual seats (left images). As can be observed, the passenger seated in the front middle generates an occlusion that does not allow visualizing if some luggage is incorrectly placed. To avoid this blind spot, we test moving the camera towards the windows to capture four individual seats (right image). From this viewpoint, we can see that there are no occlusion problems, so we opt for this configuration. Then, we test setting some cameras in the corridors to verify that area and the seats next to the corridor. Figure 5 (right) shows the tested configurations. Even though the corridor space is well visualized in both shown camera positions, the camera should be aligned with the seats (right image) to guarantee the minimum possible occlusions in the feet are of the passengers for as many seats as possible.

These tests resulted in a setup design of 20 perspective cameras on top of the seats and 19 on top of the corridor. To guarantee a good visualization of the target areas, we set the parameters of all the cameras to a FOV of 118 degrees, a focal length of 2.13mm, and a sensor size of 4mm. In the end, the final decision of how to configure the cameras comes from testing the proposed camera configurations and iteratively reviewing and testing the proposals until the system's functionality requirements are fulfilled, with the minimum number of cameras possible to improve the economic viability of the system. In order

to make passengers feel more comfortable, the manufacturers should hide the cameras, the transporters should inform passengers appropriately about their rights according to regulations (e.g., EU's GDPR and AI Act, etc.), and those regulations should protect privacy. Nevertheless, in this use case, the chosen camera viewpoints are not suitable for recognizing individuals as faces are generally not visible.

4.3 Configuration Files Generation

Once we have the final camera configuration, we export it to an updated VCD configuration file with the Scene2VCD functionality. The file should also contain the 3D assets' configuration so that our environment generator replicates the defined sequences. The situations' variety and the number of object instances of each class can be easily controlled but depends on the configuration file we use. Generating a balanced dataset is important to guarantee that the trained model does not have a bias toward the most common objects in the dataset. Consequently, we define the following requirements for generating the VCD files:

1. An object sample from each object category should be placed at all the possible configurations and placed at least once.
2. All the object classes should be present in the generated sequences' frames the same number of times.
3. Samples should show a wide variety of object appearances.

Following these criteria, we generate three VCD files. The first file describes a sequence where each frame contains an object type placed in a specific configuration at all the cabin places (e.g., backpacks on all the seats). This sequence aims to generate enough samples of each object type's appearance in simple configurations (with no interaction with other object types). The second file randomly combines all the objects in all the possible configurations the same number of times. We define a sequence of 500 frames. Each of these frames contains different object configurations. The last file follows the same strategy of random combinations, but it is focused on appearance variations. In addition to the default random variations aggregated to the 3D assets position, it enables the color randomization for objects (Section 3.1.3). Consequently, this file extends the already defined object combinations with new ones that contain objects with a wide variety of random appearances. Providing a 3D asset with a random color can make some objects look less realistic but increases the variety of samples and benefits the trained models' robustness. The strategy used for generating the VCD files is not limited to the current use case. It can be applied to other tasks and scenarios. The defined sequences are summarized in Table 4.

Table 4 Summary of the generated data based on the configuration files.

Sequence	VCD 1	VCD 2	VCD 3	Total
Description	Single object class/frame	Random, balanced	Random appearances	-
Number of frames	45	500	500	1,045
Number of rendered images	1,755	19,500	19,500	40,755
Number of 3D object instances	5,966	33,000	33,000	71,966

4.4 3D Scenes and Synthetic Data Generation

We use the VCD files to automatically replicate the 3D scenes described in them with no manual intervention. The 3D environment is configured with all the data regarding the cameras, lighting, rendering, and 3D assets to replicate the defined scenes in the cabin. The environment is dynamically configured when the data generation starts.

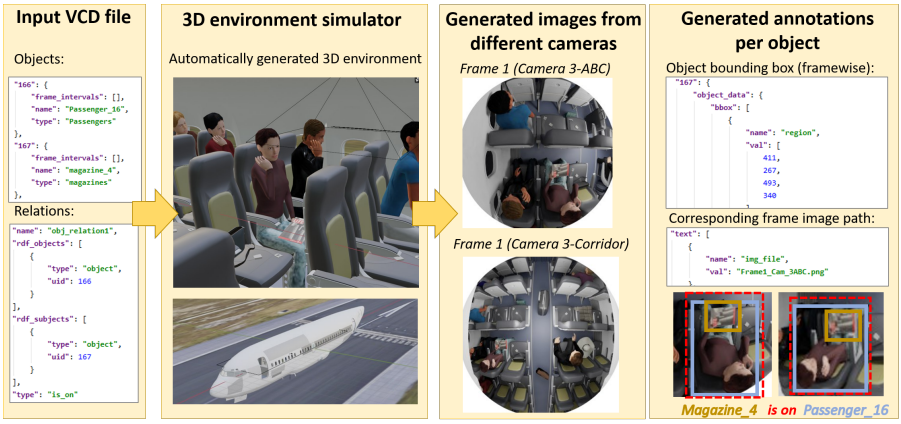

Fig. 6 Data generation pipeline example. The input VCD file describes the scene that is replicated by the synthetic 3D environment generator, including present objects and their relations. The tool outputs rendered images and corresponding annotations in VCD format, which correspond to data from different camera perspectives. [11]

Each frame in the sequences is captured from all the defined cameras, so 39 images are rendered from different camera viewpoints for each frame. An output VCD file is generated for each sequence containing the data already in the input file (e.g., camera configurations, relations between objects) and the addition of the output data. The output data include the paths to the generated images and the corresponding bounding box annotations for each object or passenger. Figure 6 shows the data generation process from the input VCD file to the output synthetic data. The left image shows an example of the objects and their relations as defined in the VCD file. The 3D environment simulator processes this information to configure the 3D scene. It can

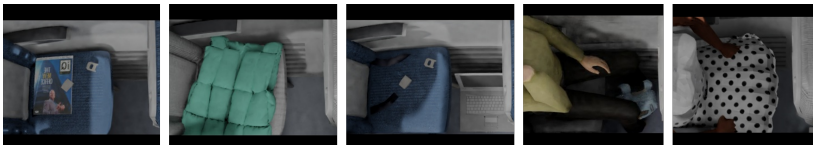

Fig. 7 Examples of synthetic images for a specific seat in the image.

be seen that the example data ('magazine-4 is on Passenger-14') is replicated in the synthetic 3D world). This scene is captured from the defined cameras to produce the corresponding rendered images and annotations. In the output synthetic images it can be observed that the same 3D assets can be observed from different cameras at the same time. The right images of Figure 6 show an example of the additional information that the output VCD file contains (objects' bounding box coordinates in each rendered frame). This information can be used for training object and visual relationship detectors.

In Figure 7 we show some examples of the appearance of a specific seat in the generated synthetic images. Different object types can be seen belonging to cabin luggage (e.g., laptop) or non-cabin luggage (e.g., magazine) placed in various positions, and different subject appearances and body poses.

4.5 Real Data Capture

We use the results of the camera setup design (Section 4.2) to replicate a cabin mock-up with the chosen camera configuration to capture also some real images.

We prepare a recording protocol for a group of 20 participants. Each participant is asked to place a piece of cabin luggage correctly or incorrectly to generate different scenes. We also ask them to act naturally and stay calm, so artificial situations with a lot of movement are avoided. Otherwise, blurry images which belong to unnatural behaviors would be captured. As a disadvantage, staying calm reinforces the possible redundancy between the captured frames. Figure 8 shows an example of this redundancy in the image crop of one of the seats. A participant stays calm with a suitcase between his legs for some seconds. We can observe that consecutive frames look very similar.

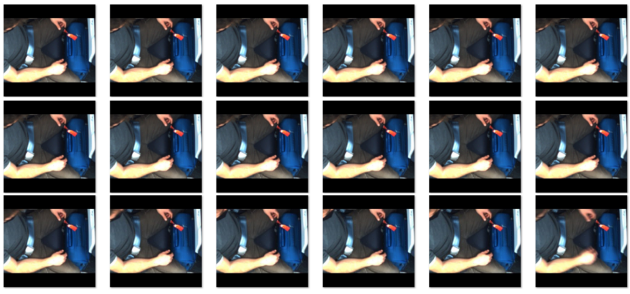

Fig. 8 Similar captured images of a participant staying calm with a suitcase between his legs.

We capture 24,000 images, half of them belonging to situations where the luggage was incorrectly placed. In order to avoid including many redundant samples in the training dataset, we apply the clustering-based image selection pipeline (Section 3.2). We use the classification model EfficientNet-B0 [37], already pretrained on ImageNet, without the last classification layers as a feature extractor. We chose EfficientNet-B0 for the backbone as it obtains high accuracy results with better efficiency than other alternative state-of-the-art DNNs. In this context, efficiency is also a key factor as the system should be energetically sustainable to go onboard. We resized input images to resolutions of 300x300, as it was the minimum resolution that allowed visualizing smaller objects with sufficient size for the classifications. This way, each input image with size 300x300 pixels is embedded as a 1280-length feature vector. Then, we use the Elbow curve for both target categories (correct and incorrect situations) to choose the number of clusters for applying the K-Means algorithm. This results in 300 clusters. From each cluster, we randomly select ten samples. We combine these images with the already generated synthetic ones.

4.6 Analysis of the Proposed Approach

This section provides a qualitative and quantitative analysis of the proposed methodology.

4.6.1 Synthetic 3D Environment

Table 5 shows a qualitative comparison of some state-of-the-art approaches to our synthetic simulated environment approach.

Feature	[15]	[14]	[13]	[18]	[17]	Our approach
Environment modeling	Manual	Manual	Manual	Manual	Manual (data priors)	Manual
Scene modifications	Manual	Manual	Manual	Manual	Manual	Automatic
Scene capture	Single-sensor	Single-sensor	Single-sensor	Multisensor	Single-sensor	Multisensor
3D assets relations	Users' interactions	None	None	None	None	Spatial configuration
Annotations	Obj. detection, sem. segmentation, reinforcement learning	Obj. detection, sem. segmentation	Sem. segmentation	Obj. detection	Sem. segmentation, depth estimation	Obj. detection, sem. segmentation, visual relationship detection
Generality	Default scenarios	Limited	Driving scenarios	Limited	Urban driving scenarios	Surveillance scenarios

Table 5 Comparison between state-of-the-art synthetic dataset generation methodologies and our approach.

The environment and 3D assets involved in all the data generation methods are manually modeled with the help of a 3D modeling software or gathered from public repositories. [17] also uses data priors such as OpenStreetMap data to model different city environments but still needs manual work to complete and adjust the scene data.

Once the 3D environment is prepared, if the user wants to change specific scene configurations with our method, such as the light properties, the objects in the scene, or the relationship between these objects, he/she can define the

modified scene in the configuration file using a user-friendly parameterization. Then the new 3D scenario will be automatically replicated. Our approach is the only one that proposes to dynamically configure all the environments when the data generation process starts. Scene modifications can be done in the different approaches, but none of them provides a high-level mechanism like ours to vary the environment. Related to the possible 3D assets relations, [15] allows adding some limited user interactions. Our approach allows adding spatial location relations between the 3D assets.

Regarding the image capture, all the works propose a single source to capture the scene, except for [18] and our approach, which allows capturing the same time interval from cameras with different viewpoints.

The data generated by the presented works are oriented to the training of DNNs. Typical output includes annotations for object detection or semantic segmentation tasks, to which [15] adds reinforcement learning. Our approach also generates labels suitable for training visual relationship detectors.

One of the main advantages of our approach over the others is its generality and the possibility to adapt it with little effort to new scenarios and tasks for passenger transportation safety, compared to building 'ad hoc' simulated environments each time. This feature is very limited to the predefined scenarios in the state-of-the-art works.

The rendering time depends on different factors such as the rendering engine, the complexity of the scene, the number of 3D assets, the light configuration, or the polygon number of the modeled objects. For the current aircraft use case, we configure the rendering to be as fast as possible using the parameters in the configuration file, maintaining a good output quality. We use the real-time viewport shading rendering for the camera setup design (Section 4.2) so that we can see the modifications' effect interactively. For the data generation, we use the *Cycles Rendering Engine*, which is slower but provides more photorealistic results. It takes 9.3 seconds to render an image of 640x480 pixels of the cabin environment, including 50 3D assets (e.g., suitcases, passengers) apart from the background ones, 16 point lights, and outdoors sunlight. We use an Nvidia Tesla T4 GPU. Regarding the rendering time of related state-of-the-art approaches, some works based on game engines claim to render in real-time [14, 17]. As stated in [17], a simplified lighting model allows real-time rendering of massive amounts of geometry with limited realism. The *Cycles Rendering Engine*, as a physically-based path tracer, allows generating good quality results in a reasonable time. Domain adaptation techniques are important to solve the domain gap that DNNs trained with synthetic data can present. Generating data with a certain degree of realism minimizes the problem to be solved by those techniques. However, *Blender* also has a real-time rendering engine (*Eevee*) that can be used in tasks where the rendering time is considered to be a bigger priority than the data quality.

We apply our methodology to the aircraft use case, but adapting it to another surveillance scenario requires little effort from the user. Once the user collects all the 3D assets of the new environment, the flexibility of the method

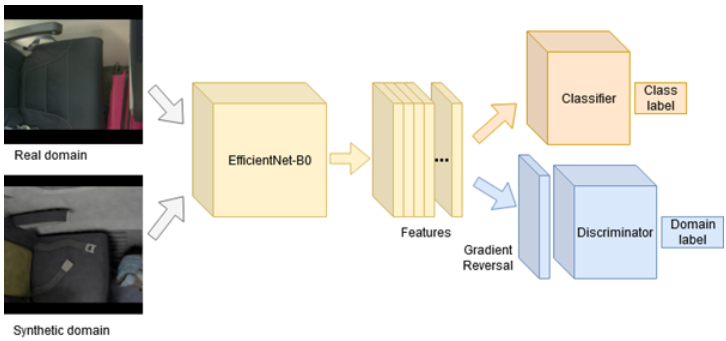

Fig. 9 Trained DANN for learning domain-invariant image features.

allows building a new scene. The user interactively designs a suitable camera setup, along with the selection of target places. Then, the user can start gathering variate training data for the considered task based on the configuration files.

4.6.2 Real Samples Selection

In order to analyze whether the images' redundancy problem affects the DNN training and, in that case, to analyze how much our clustering-based approach helps mitigate it, we do the following tests. We train the EfficientNet-B0 image classifier with different datasets but with the same training configuration to analyze the impact of varying data sampling strategies. We define a region of interest (ROI) for each seat, which will be classified as correct or incorrect by the DNN depending on the correctness of the cabin luggage. We generate different datasets following each of the subsequent strategies:

- Use all the captured data.
- Use a random subset of the recorded data.
- Use a subset of the recorded data, skipping some images when selecting them based on the order they were captured.
- Use a subset of the recorded data based on the proposed clustering algorithm that helps us select non-redundant images.

Then, we add the real samples to a subset of 4,000 synthetic samples for each option to complete the training dataset. We use these datasets in the DNN training experiments (Section 4.6.3).

4.6.3 DNN Training

In order to see how a DNN would perform with the generated synthetic and real data and to see the impact of different data sampling strategies, we train the model EfficientNet-B0 to classify images as correct or incorrect (e.g., cabin luggage correctly or incorrectly placed).

To minimize the domain gap between the synthetic and real domains during the training, we include the domain adaptation technique of [38]. The idea is

to learn to extract domain-invariant features from the images using a Domain Adversarial Neural Network (DANN), of which an overview schema is shown in Figure 9.

We use EfficientNet-B0 as the feature extraction backbone. The main classifier of the architecture classifies the images in the correct or incorrect category and contains a fully connected layer and a softmax activation. The discriminator is an adversarial domain classifier whose loss is maximized using a gradient reversal layer [38]. It tries to classify the domain an input sample belongs to using two fully connected layers output and a softmax activation. The fully connected layers have 1,280 and 2 outputs, respectively.

Sampling strategies. To see the impact of the different sampling strategies, we train the DANN with the different generated datasets (Section 4.6.2). We initialize the DANN backbone with pretrained weights on the ImageNet dataset [22] and fine-tune it with our datasets. The classifier and the discriminator are trained from scratch. We train each model for 50 epochs with a batch size of 40, and the RMSprop optimizer [39]. We split the data into training, validation, and test sets. We separate the captures of a recording session to use them in the validation and test sets so that neither the participants nor the lighting conditions used in training are repeated in the validation or test sets, which contain 2,000 and 500 images, respectively. The results are shown in Table 6.

Sampling Method	**Test Accuracy**
All images	86.9%
Random subset of images	86.7%
Skipping consecutive frames	92.2%
Selecting images based on the clustered embeddings	95.3%

Table 6 Comparison between achieved DNN accuracy in the test set for different sampling strategies. Synthetic images are added to the real ones in all the trainings.

As it can be seen, using all the captured images does not lead to the best results. Selecting a subset of the images (30% of the images) randomly does not help either, resulting in the lowest accuracy (86.7%). The image redundancy is confirmed as a problem because when we skip 10 consecutive frames, we obtain an accuracy boost of almost 6% compared to using all the samples. Selecting the training samples with the proposed approach improves an additional 3.1% of the final accuracy. Consequently, selecting the images based on their content helps to improve the training dataset.

Synthetic and real data combination. We compare the proposed DANN with other state-of-the-art methods that combine synthetic and real domain data using different strategies. There are no works applied to the same task, so we adopt the methods followed by other authors for different tasks and apply them to our DNN training. We summarize the results of the experiments in Table 7. The authors in [10, 29] train their DNN on the synthetic domain and then fine-tune the model on the real domain in a second step.

We replicate this process with our data. We train the classification DNN with no domain adversarial branch for this experiment. We use the EfficientNet-B0 backbone as in our proposed method and maintain the same training parameters configuration to make both models comparable. The model achieves an accuracy of 93.8%, which is 1.5% lower than our proposed methodology. We additionally compare our method with [30, 31], which jointly train with both domains' images by using mini-batches from the source and target domain. We follow the same data strategy and train the classification DNN with no domain adversarial branch. We use the EfficientNet-B0 backbone and maintain the same training parameter configuration. The model achieves an accuracy of 92.2% in the test set, which is 3.1% lower than our proposed method.

Synthetic and real data combination strategy	**Test Accuracy**
Pretrain with synthetic data, fine-tune with real data [10, 29]	93.8%
Train with real and synthetic mini-batches [30, 31]	92.2%
Train with a domain adversarial strategy (ours)	95.3%

Table 7 Comparison between achieved DNN accuracy in the test set for different data combination strategies.

Figure 10 shows some correctly and incorrectly classified test images using our proposed DANN for the synthetic and real data domains. In the first column, we show two samples of situations with cabin luggage incorrectly placed, which are correctly classified as so. There is no cabin luggage in the second and third column images, but the last ones are incorrectly classified. We think this is probably because of these images' dark or blurry characteristics. In the last column, we show two more samples of incorrect classifications. In both images, cabin luggage is barely visible on the egress. This is the most challenging situation for the classification, as the passenger or the seat might remarkably occlude the objects. In these situations, we think that temporal analysis of the consecutive frames' classification might help.

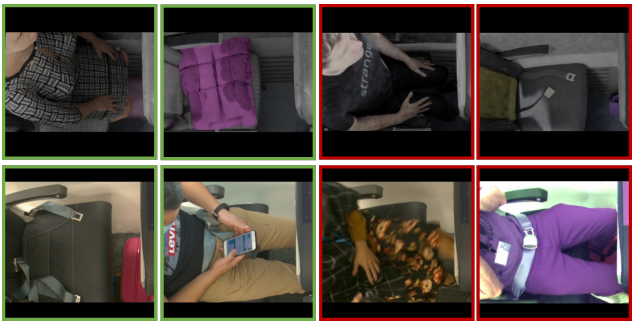

Fig. 10 Some examples of correctly (green) and incorrectly (red) classified images using the proposed DANN model. Images in the top row are synthetic and images in the bottom row are real.

5 Conclusions

This work presents a methodology to build an appropriate dataset for DNN training combining synthetic and real data. We propose a method for building simulated 3D environments for configuring and training multi-camera systems with enough generality to be used in different surveillance contexts with little effort. Our proposal helps design an appropriate camera system to cover the target use cases and avoid expensive system setup errors. Once the camera setup is done, our method allows generating a wide range of situations of interest for training DNNs with suitable synthetic data. The input configuration files allow controlling the content of the data with a user-friendly fast scene parameterization and consequently generating a balanced dataset. Including captured real data in the dataset helps minimize the domain gap between real and synthetic samples. However, captured data may present high redundancy. We present a method based on feature clustering to avoid including redundant samples which penalize the DNN accuracy.

We show a practical implementation example of our methodology in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based smart sensing system. We first design a 39 camera-based system and generate a set of synthetic samples in the cabin environment. We follow a data balancing strategy to guarantee the suitability of the generated data based on the configuration files. We also replicate the designed setup with a cabin mock-up to capture real samples of the target scenario. We determine the less redundant samples for the DNN training based on their content. To select them, we use a pretrained DNN to encode the images as feature vectors and cluster them based on their similarity.

We compare our synthetic environment to alternative state-of-the-art approaches to validate our methodology. Features such as the generality, flexibility, and multi-camera setting stand out from the features provided by the other approaches. Regarding selecting the real samples, we generate different datasets following different strategies to see their impact on the training of a DNN. We train a DANN with the generated synthetic and real samples. We show that incorporating our synthetic samples into the training dataset boosts the model's accuracy when tested on real images. In addition, the clustering-based sampling strategy for the real data showed a positive impact on the model accuracy. Consequently, the dataset generated by our methodology is suitable for training DNNs.

Future work includes the research of the various possible domain adaptation techniques for training DNNs and their suitability for different computer tasks involving synthetic and real samples.

Acknowledgments. This work has received funding from the Clean Sky 2 Joint Undertaking under the European Union's Horizon 2020 research and innovation program under grant agreement No. 865162, SmaCS (https://www.smacs.eu/).

On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

[1] Q. Hou, M. Cheng, X. Hu, A. Borji, Z. Tu, and P. H. S. Torr, "Deeply supervised salient object detection with short connections," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 4, pp. 815–828, 2019.

[2] K. K. Maninis, S. Caelles, Y. Chen, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool, "Video object segmentation without temporal information," *IEEE TPAMI*, vol. 41, no. 6, pp. 1515–1530, 2019.

[3] Y. Xi, Y. Zhang, S. Ding, and S. Wan, "Visual question answering model based on visual relationship detection," *Signal Processing: Image Communication*, vol. 80, p. 115648, 2020.

[4] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue, and N. Zheng, "View adaptive neural networks for high performance skeleton-based human action recognition," *IEEE TPAMI*, vol. 41, no. 8, pp. 1963–1978, 2019.

[5] A. Mujika, A. D. Fanlo, I. Tamayo, O. Senderos, J. Barandiaran, N. Aranjuelo, M. Nieto, and O. Otaegui, "Web-based video-assisted point cloud annotation for ADAS validation," in *Proc. International Conference on 3D Web Technology*, pp. 1–9, 2019.

[6] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, pp. 1–48, 2019.

[7] R. Singh, M. Vatsa, V. M. Patel, and N. Ratha, eds., *Domain Adaptation for Visual Understanding*. Springer, Cham, 2020.

[8] S. I. Nikolenko, "Synthetic data for deep learning," in *Springer Optimization and Its Applications*, vol. 174, 2019.

[9] V. Seib, B. Lange, and S. Wirtz, "Mixing real and synthetic data to enhance neural network training – a review of current approaches," *arXiv preprint arXiv:2007.08781*, 2020.

[10] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proc. IEEE CVPR workshops*, pp. 969–977, 2018.

[11] N. Aranjuelo, J. García, L. Unzueta, S. García, U. Elordi, and O. Otaegui, "Building synthetic simulated environments for configuring and training

multi-camera systems for surveillance applications.," in *Proc. VISIGRAPP (5: VISAPP)*, pp. 80–91, 2021.

[12] B. Hurl, K. Czarnecki, and S. L. Waslander, "Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception," in *IEEE IV*, pp. 2522–2529, 2019.

[13] F. S. Saleh, M. S. Aliakbarian, M. Salzmann, L. Petersson, and J. M. Alvarez, "Effective use of synthetic data for urban scene semantic segmentation," in *Proc. ECCV*, vol. 11206 of *LNCS*, pp. 86–103, 2018.

[14] T. Scheck, R. Seidel, and G. Hirtz, "Learning from theodore: A synthetic omnidirectional top-view indoor dataset for deep transfer learning," in *Proc. IEEE WACV*, pp. 932–941, 2020.

[15] K.-T. Lai, C.-C. Lin, C.-Y. Kang, M.-E. Liao, and M.-S. Chen, "VIVID: Virtual environment for visual deep learning," in *Proc. ACM MM*, pp. 1356–1359, 2018.

[16] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," *Field and Service Robotics*, pp. 621–635, 2017.

[17] S. Khan, B. Phan, R. Salay, and K. Czarnecki, "ProcSy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks," in *Proc. CVPR Workshops*, pp. 88–96, 2019.

[18] P. S. Rajpura, H. Bojinov, and R. S. Hegde, "Object detection using deep CNNs trained on synthetic images," *arXiv preprint arXiv:1706.06782*, 2017.

[19] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 34, pp. 248:1–248:16, Oct. 2015.

[20] P. Bourke and D. Felinto, "Blender and immersive gaming in a hemispherical dome," in *International Conference on Computer Games, Multimedia and Allied Technology*, vol. 1, pp. 280–284, 2010.

[21] P. Hernandez-Leal, B. Kartal, and M. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, pp. 750–797, 10 2019.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE/CVF CVPR*, pp. 248–255, Ieee, 2009.

[23] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, *et al.*, "The open images dataset v4," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[24] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," in *Proc. ECCV*, pp. 491–507, Springer, 2020.

[25] M. Raghu, C. Zhang, J. M. Kleinberg, and S. Bengio, "Transfusion: Understanding transfer learning for medical imaging," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 3342–3352, 2019.

[26] S. Niu, M. Liu, Y. Liu, J. Wang, and H. Song, "Distant domain transfer learning for medical imaging," *IEEE J. Biomed. Health Informatics*, vol. 25, no. 10, pp. 3784–3793, 2021.

[27] C. J. Reed, X. Yue, A. Nrusimha, S. Ebrahimi, V. Vijaykumar, R. Mao, B. Li, S. Zhang, D. Guillory, S. Metzger, K. Keutzer, and T. Darrell, "Self-supervised pretraining improves self-supervised pretraining," in *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV*, pp. 1050–1060, 2022.

[28] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proc. IEEE/CVF CVPR*, pp. 403–412, 2019.

[29] A. Shafaei, J. J. Little, and M. Schmidt, "Play and learn: Using video games to train computer vision models," in *Proc. BMVC*, 2016.

[30] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proc. IEEE CVPR*, pp. 3234–3243, 2016.

[31] N. Aranjuelo, S. García, E. Loyo, L. Unzueta, and O. Otaegui, "Key strategies for synthetic data generation for training intelligent systems based on people detection from omnidirectional cameras," *Computers & Electrical Engineering*, vol. 92, p. 107105, 2021.

[32] A. Cortés, C. Rodríguez, G. Vélez, J. Barandiarán, and M. Nieto, "Analysis of classifier training on synthetic data for cross-domain datasets," *IEEE Trans. on Intelligent Transportation Systems*, 2020.

[33] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[34] M. Tonutti, E. Ruffaldi, A. Cattaneo, and C. A. Avizzano, "Robust and subject-independent driving manoeuvre anticipation through domain-adversarial recurrent neural networks," *Robotics and Autonomous Systems*, vol. 115, pp. 162–173, 2019.

[35] Vicomtech, "VCD - video content description." https://vcd.vicomtech.org/, 2020.

[36] ASAM, "OpenLABEL." https://www.asam.net/project-detail/scenario-storage-and-labelling/, 2020.

[37] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proc. ICML* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *PMLR*, pp. 6105–6114, 2019.

[38] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[39] Y. N. Dauphin, H. de Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1504–1512, 2015.