Journal of Artificial Intelligence and Control Systems

CoSci Press

Research Article

Comparative Validation of a 3D Synthetic Data Approach for At-risk Patient Monitoring with Computer Vision: An Analysis of Frameworks and Strategies

Bruno J. Santos

Department of Information Technology, University of Sao Judas Tadeu, Sao Paulo 03166-000, Brazil

Article History:

Received: 9 July 2025

Revised: 7 August 2025

Accepted: 17 October 2025

Published: 1 April 2026

Abstract: Training Computer Vision (CV) models for patient monitoring in clinical settings, such as rehabilitation beds, is often hindered by the impracticality of collecting real-world image data for at-risk scenarios. This study proposes and validates a methodology based on generating a synthetic dataset using 3D modeling. A virtual environment replicating a Reconfigurable Assistive Technology Platform (RATP) was created in Blender software, and animated human models were used to generate 960 images depicting 'safe' (sleeping) and 'at-risk' (fallen) states. A series of eight experiments was carried out to systematically evaluate eight distinct model architectures (MAs), comparing two development frameworks (TensorFlow/Keras and PyTorch), two training strategies (Transfer Learning with MobileNetV2 and training from scratch), and the effect of an image preprocessing pipeline. The results revealed a significant performance disparity between frameworks when evaluated on a benchmark dataset. The best-performing model, a PyTorch-based Transfer Learning architecture (MA3), achieved 95.8% precision in detecting at-risk states, substantially outperforming its TensorFlow counterpart (MA1) at 58.7%. Furthermore, the effect of preprocessing proved to be highly context-dependent, improving performance in one framework while severely degrading it in the other. This work demonstrates the viability of using synthetic data for this application but concludes that model performance is critically sensitive not only to the learning strategy but also to the underlying software framework and preprocessing choices, highlighting the need for careful experimental environment configuration.

Keywords: computer vision; image processing; pattern recognition; health-care management; embedded systems

1. Introduction

Bedridden patients are individuals who, due to conditions such as multiple traumas, spinal fractures, body paralysis, or medical recommendations, remain confined to bed for extended periods. Rehabilitation hospital beds, also known as therapeutic beds, are specialized medical devices designed to facilitate the recovery of patients who have sustained injuries, experienced severe illnesses, or undergone surgical procedures. Utilizing these beds allows patients to participate in rehabilitation therapies more safely and comfortably, accelerating recovery and reducing hospitalization time [1].

Despite their broad individual and societal benefits, rehabilitation services are often neglected and underfunded. They are frequently mistakenly perceived as exclusive to people with disabilities, thereby limiting access to only a tiny portion of the population [2]. To address this challenge, it is recommended that rehabilitation services be incorporated into primary healthcare as an essential component. In this context,

* Corresponding author: Bruno J. Santos, University of Sao Judas Tadeu, brunojsantos@usp.br

the so-called Health 4.0 can play a crucial role by enabling the development of new treatments, enhancing patient monitoring, and optimizing resource management in healthcare facilities [3].

Smart hospital beds, equipped with sensors and monitoring systems, allow for continuous tracking of patients' physical conditions, providing real-time feedback to healthcare professionals. Additionally, these technologies contribute to increased patient comfort and improved sleep quality by automatically adjusting the bed's position as needed [4].

Rehabilitation may be delivered in hospital or at home, depending on patients' clinical needs and individual circumstances. Comparative studies indicate that both settings effectively support recovery; notably, home-based rehabilitation is associated with higher patient satisfaction and reduced costs to health systems [5]. Against this backdrop, the Reconfigurable Assistive Technology Platform (RATP) developed at the Robotics Laboratory of the Federal University of Santa Catarina (UFSC) offers a flexible assistive solution that can operate across care settings. Using linear actuators, the system provides graded support for tasks such as safe transfers between wheelchairs and beds, physiotherapy assistance, and postural adjustments tailored to each patient's clinical profile, thereby facilitating personalized and potentially more accessible home-based care [6,7].

In intelligent control systems integrated with hospital beds, computer vision (CV) enables the automatic detection and interpretation of the surrounding environment by combining image and video processing with machine learning to classify and predict events. Nevertheless, progress in the field remains constrained by the acquisition of real-world data, a process that is typically costly, time-consuming, and limited in diversity. A promising strategy is the use of synthetic data generated in three-dimensional virtual environments, which enables large-scale creation of datasets across a wide range of conditions, many of which would be impracticable to capture in reality.

This study investigates a novel training methodology for a CV based system for autonomous patient monitoring. The primary research question is: Is it feasible to develop an autonomous CV system to effectively analyze a patient's in-bed movements and generate relevant safety alerts for healthcare staff?

This is addressed through four sub-questions:

- RQ1: Does the choice of development framework significantly influence performance?
- RQ2: What is the impact of data preprocessing on model performance?
- RQ3: Can a pre-trained CV model trained with synthetic images achieve adequate performance?
- RQ4: Can a CV model trained from scratch with synthetic images achieve adequate performance?

This work presents a preliminary assessment of the feasibility of using synthetic images generated via 3D modeling to train CV models for detecting patient risk states, offering an ethical alternative when the use of real data is infeasible. Its main contribution is a reproducible pipeline for data generation and use, with targeted studies of key parameters, training from scratch vs. transfer learning, framework choice, and preprocessing, adaptable to other clinical tasks. The study establishes a methodological paradigm to accelerate the development and validation of assistive technologies in biomedical engineering.

The remainder of the paper is organized as follows. Section 2 reviews related work on synthetic image generation for training AI models; Section 3 details the materials and methods, including experimental design, 3D scene generation, data preprocessing, software frameworks, and model architectures; Section 4 presents the results and comparative analyses; Section 5 discusses implications, limitations, and generalizability; and Section 6 concludes by summarizing the findings and outlining directions for future research.

2. Related Works

In intelligent control systems integrated into smart hospital beds, CV enables the automatic detection and interpretation of visual information in the bed's surroundings [8]. CV is an interdisciplinary field that combines image and video processing techniques with Machine Learning (ML) to understand camera-captured content, and is applied in security, healthcare, agriculture, and transportation owing to its ability to automatically predict and classify events of interest [9]. However, recent advances in CV are often constrained by the challenges of acquiring real-world data, a costly, time-consuming process that is limited in the diversity and representativeness of scenarios [10].

To mitigate these limitations, the use of synthetic data generated in three-dimensional (3D) virtual environments has gained prominence. This approach enables the large-scale creation of tailored datasets, simulating a wide range of conditions that would be logistically or financially infeasible to capture in the real world [11]. The utility of virtual environments extends to multiple domains in which real-world collection is impractical, risky, or insufficient: in high-risk areas such as autonomous vehicles and industrial robotics, simulations are crucial for safely testing collision scenarios and human–robot interactions prior to practical deployment [12–14]. Moreover, these environments allow the development of

customized monitoring systems, for example, patient rehabilitation platforms adapted to individual needs. When real data are scarce, as in medical diagnoses of rare conditions or in safety analyses, ML models can be trained exclusively with synthetic images [15,16]. The effectiveness of this method stems from the precise control of scene parameters (lighting, camera viewpoints, textures, and physical phenomena) to produce variations that enrich datasets and improve the generalization capacity of the models [17]. Specialized software is central to this process; the open-source Blender offers a comprehensive suite of tools for complex 3D modeling, physical simulation, and realistic rendering [18], establishing itself as a valuable resource for generating high-fidelity images suitable for training CV models.

In the context of robotics, the evolution of deep learning has intensified the demand for labeled data, making simulation a strategic alternative for generating, at scale, images with precise annotations. This alternative, however, introduces the so-called reality gap, arising from visual and physical discrepancies between virtual environments and the physical world. The central aim of sim-to-real approaches is to reduce this discrepancy so that models trained in simulation operate robustly and reliably on real platforms, including in clinical applications, such as perception around smart beds [19,20].

A prominent line of work is domain randomization, which broadens the training distribution by programmatically varying textures, lighting, backgrounds, and physical parameters. Empirical evidence indicates direct transfer to the real world without the use of real images during training: grasp detection with mean errors on the order of centimeters [21], indoor drone navigation with monocular vision [22], and complex manipulation with automatic and progressive randomization [23]. The underlying mechanism consists in inducing invariances through exposure to wide synthetic variability.

Another avenue invests in photorealistic environments and large-scale synthetic datasets to approximate the appearance of the real world and provide abundant ground truth. SYNTHIA demonstrated gains by combining real and synthetic data for semantic segmentation [24]. CARLA established an interactive simulation ecosystem for autonomous driving, with virtual sensors and controllable scenarios [25], while data extracted from high-fidelity commercial games (e.g., GTA V) have been used for pretraining and evaluation. In indoor robotics, platforms such as Gibson and AI Habitat have enabled navigation agents in photorealistic scenes, reducing the visual gap and facilitating subsequent adaptation to the real domain. Nevertheless, limitations persist related to the cost of developing realistic simulations and residual sensor differences.

Complementarily, domain adaptation seeks to align the synthetic and real distributions at the pixel level or within internal representations. In grasping, the combination of simulation with adversarial style translation (GraspGAN) and training with unlabeled real data reduced, by orders of magnitude, the need for labels to achieve comparable performance [26]. Refinement methods based on GANs with unlabeled real images (SimGAN) [27] and mappings from the randomized domain to a canonical domain (RCAN) [28] show that intermediate steps of normalization or statistical alignment increase data efficiency in sim-to-real transfer, albeit at the cost of greater training complexity.

3. Materials and Methods

The proposed method for autonomous patient risk detection at the RATP comprises the following four steps: (1) CV dataset creation, (2) CV preprocessing, (3) CV model design and training, and (4) CV performance evaluation.

The methodology for synthetic image generation to dataset (Stage 1) deviated from conventional video production. Instead, each frame of the animation was rendered as an individual static image to compose the dataset. This methodology was presented in detail in a scientific conference workshop; a link to the presentation is provided in the Supplementary Materials.

Synthetic image generation for this project began with the 3D models of the RATP's physical assembly, which were developed in Solidworks software (Dassault Systèmes S.A, Vélizy-Villacoublay, France). These models were exported as a GLTF file and subsequently imported into Blender software. Within the Blender 3D environment, a material with a luminous white texture was applied to the graphical elements to replicate the appearance of the physical assembly. Finally, a lighting setup was configured for the scene. The RATP physical assembly and the 3D models in both environments are illustrated in Figure 1.

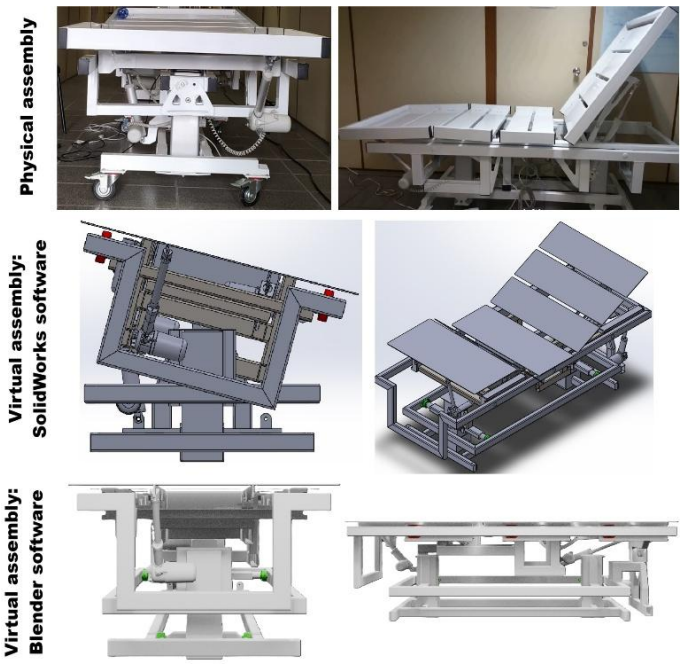

Figure 1. RATP assembly: Physical, 3D Model in Solidworks, and 3D Model in Blender

For the simulation of patients within the virtual environment of the Blender software, the online platform Mixamo (Adobe Inc, San Jose, USA), a service specializing in the automation of 3D character animation, was employed. The methodology consisted of utilizing Mixamo's resources to obtain and animate 3D models of human figures, which would represent the patients. Technically, the platform operates based on machine learning algorithms to perform the auto-rigging process, the application of a digital skeleton to a 3D model and for animation retargeting. This allows a vast library of pre-existing movements, captured via motion capture technology, to be efficiently applied to the models. The adoption of this tool significantly accelerated the implementation of animated characters by eliminating the need for complex manual rigging and animation processes.

For this study, the 'Elizabeth' (female) and 'Brian' (male) patient models were selected from the Mixamo platform (Figure 2). Although a larger number of suitable models were available, these were chosen to establish a proof of concept.

Figure 2. Patient models used for the proof-of-concept simulation.

Two distinct animations were applied to these models to denote the patient's state: 'sleeping' representing a safe condition in the RATP, and 'fallen to the ground' representing an at-risk condition (Figure 3). The selection of these two animations from the many available also aligns with the proof-of-concept scope, focusing on the distinction between risk scenarios for autonomous patient identification.

Figure 3. Patient state animations representing the 'safe' condition (sleeping) and the 'at-risk' condition (fallen to the ground).

Each model, with its respective animations, was exported in FBX format and subsequently imported into the Blender environment containing the 3D model of the RATP (Figure 4). Within this scene, four virtual cameras were strategically positioned to capture multiple perspectives: frontal, rear, left lateral, and right lateral.

Figure 4. Integration of the animated patient model into the 3D environment of the RATP within Blender.

The synthetic image generation culminated in a dataset of approximately 960 images (Stage 1). The animations, rendered at 30 frames per second, were structured to create a balanced distribution: 480 images representing the 'safe' state (patient sleeping) and 480 for the 'at-risk' state (patient fallen), combining outputs from both character models.

The CV preprocessing (Stage 2), whose pipeline was detailed in a previous work [8], was implemented in Python

using the OpenCV library. OpenCV is an open-source library for computer vision, machine learning, and image processing. Its Python interface allows the implementation of algorithms for analyzing and manipulating images and videos. Its features range from basic operations, such as reading and color transformations, to advanced techniques for object detection, facial recognition, and motion analysis [29].

The primary objective of this stage is to optimize the image for contour detection by reducing the computational load and isolating the patient within the environment. The process consists of the following five sequential steps, as illustrated in Figure 5:

0. Mask Application: An occlusion mask was applied to remove visual textures and noise from the background (floor and walls), thereby focusing the analysis solely on the region of interest.
1. Grayscale Conversion: The image was converted to a single intensity channel to simplify chromatic information and reduce the computational complexity of subsequent steps.
2. Image Smoothing: A Gaussian blur filter was employed to attenuate high-frequency noise in the grayscale image.
3. Adaptive Binarization: The grayscale image was converted into a binary (black and white) image using Gaussian adaptive thresholding, which calculates an optimal threshold for different image regions, simplifying edge detection.
4. Contour Detection: The Sobel operator was used on the binarized image to detect horizontal and vertical contours, resulting in the extraction of a list of contours that includes the patients' silhouettes.

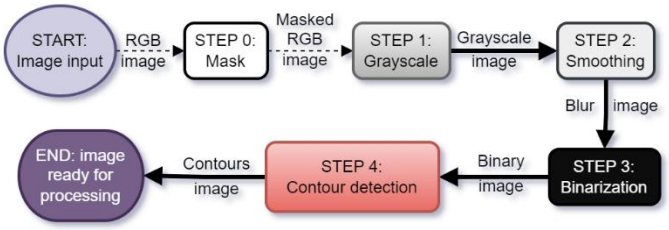

Figure 5. CV pipeline of preprocessing: mask, grayscale, smoothing, binarization and contours

The output of preprocessing pipeline (Figure 6), involved generating and saving a corresponding contour image for each original, thereby doubling the total dataset size.

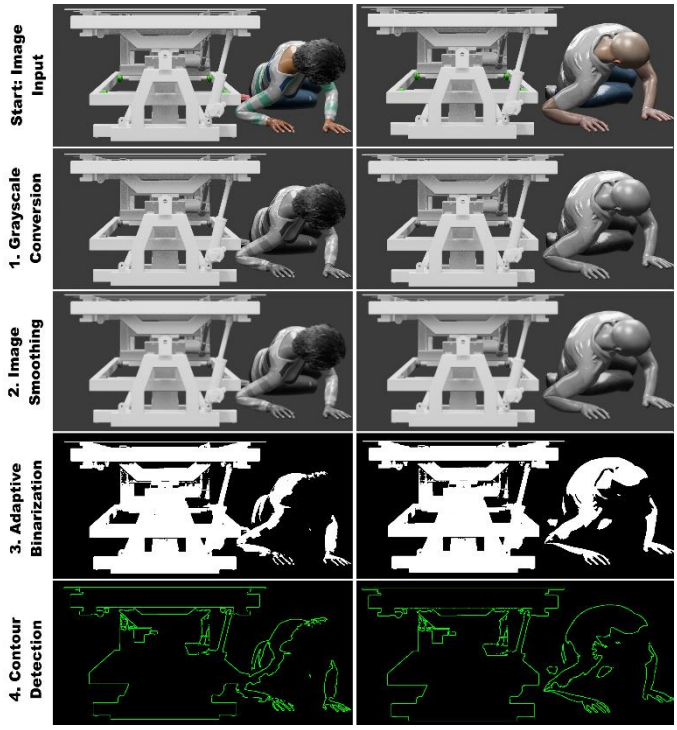

Figure 6. Outputs of CV pipeline of preprocessing: grayscale, smoothing, binarization and contours

For CV model design and training (Stage 3), the development environment used Python with the TensorFlow/Keras and PyTorch libraries.

TensorFlow is an end-to-end open-source platform for machine learning, developed by Google. It offers a comprehensive ecosystem of tools, libraries, and resources that enable the construction and training of neural network models. Its flexible architecture supports the deployment of computation across various platforms (CPUs, GPUs, TPUs) and is widely used in both research and production for large-scale tasks [30].

Keras is a high-level Application Programming Interface (API) for neural networks, written in Python and capable of running on top of frameworks. Designed for rapid experimentation, its main characteristic is simplicity and ease of use, allowing developers to build and test deep learning models with minimal code [31].

PyTorch is an open-source machine learning framework, primarily developed by Facebook's Artificial Intelligence Research lab (Meta AI). It is known for its flexibility and "Pythonic" approach, utilizing tensors (multi-dimensional arrays) and an automatic differentiation system for building dynamic computational graphs. PyTorch is widely favored in the academic research community for its ease of prototyping and debugging complex models [32].

The dataset was extracted, with the classes (safe, at-risk) being inferred from the directory structure. Subsequently, the

data was partitioned into training (70%) and validation (30%) subsets, using a deterministic seed to ensure reproducibility.

The ML models were implemented employing the Transfer Learning technique and from scratch. The architecture with Transfer Learning strategy is based on the MobileNetV2 model pre-trained on ImageNet. MobileNetV2 is a convolutional neural network architecture designed to reduce computational complexity and the number of parameters while maintaining classification accuracy. Its fundamental unit is the inverted residual bottleneck block. This block operates on a low-dimensional input representation, expands it with a 1x1 pointwise convolution, applies a 3x3 depthwise convolution for spatial filtering, and subsequently projects the representation back to a low dimension using another 1x1 pointwise convolution. Crucially, the final projection layer is linear, meaning it does not have a non-linear activation function, which prevents the destruction of information in low-dimensional representations. Residual connections are employed between the bottleneck blocks (low-dimensional layers) to enable the training of deeper networks [33].

In TensorFlow framework, the images were standardized to a 160x160 pixel resolution and organized into batches of 32 samples. To optimize the data pipeline performance, TensorFlow's autotune feature was employed to asynchronously prefetch data batches.

The Transfer Learning strategy consisted of using its convolutional base as a fixed feature extractor by freezing its 2,257,984 parameters to prevent updates during training. Upon this base, a new classification head with 1,281 trainable parameters was sequentially constructed, comprising:

- A MobileNetV2 preprocessing layer to normalize pixel values to the [-1, 1] range.
- The frozen MobileNetV2 base.
- A Global average Pooling 2D layer for dimensionality reduction.
- A Dropout regularization layer with a rate of 0.2 to mitigate overfitting.
- A Dense output layer with a single neuron and a sigmoid activation function for binary classification.

Within the PyTorch framework, a transformation pipeline was applied to each image in the dataset, comprising the following steps:

- Resizing to a standard resolution of 160×160 pixels.
- Conversion to the PyTorch tensor format.
- Normalization of pixel values using the ImageNet dataset's means and standard deviations (mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]).
- A Dropout regularization layer with a rate of 0.2 to mitigate overfitting.
- A Dense output layer with a single output for binary classification.
- The model was compiled using the Adam optimizer with a learning rate (η) of 0.0001 and the Binary Cross-Entropy loss function. Accuracy was monitored as the performance metric. The model was configured to train for 100 epochs. Finally, the trained model was serialized and saved in the HDF5 format for subsequent evaluation.

The model from scratch's architecture is a sequential Convolutional Neural Network (CNN). A CNN receives an input tensor $x \in \mathbb{R}^{\{H\times W\times Cin\}}$ (height, width, and input channels) and processes it with C_{out} learnable filters. Each filter has weights $W \in \mathbb{R}^{\{K\times K\times Cin\}}$ and a bias term b. At spatial location (i, j), the output of filter k is obtained by a local weighted sum followed by a nonlinearity σ (e.g., ReLU).

$$y_k(i,j) = \sigma(\Sigma_{\{u,v,c\}} W_{k(u,v,c)} \cdot x(i+u, j+v, c) + b_k)$$

The sequential CNN is composed of a feature extraction base and a classification head. The feature extraction base begins with a Rescaling layer that normalizes image pixel values from the [0,255] range to [0,1]. This is followed by three sequential convolutional blocks. Each block comprises a Conv2D layer with 3x3 filters and a ReLU activation function, followed by a MaxPooling2D layer (2x2) for spatial downsampling. The filter depth increases hierarchically across the blocks (32, 64, and 128, respectively) to enable the learning of features with increasing complexity.

Four hyperparameters define the convolutional geometry. The kernel size K (e.g., 3×3) determines the local field of view; larger kernels capture broader patterns at higher computational cost. The stride S sets the scanning step; larger strides reduce the spatial resolution of feature maps (downsampling). Padding P adds border elements (typically zeros) to preserve spatial dimensions and mitigate boundary effects. Dilation d spaces the kernel elements, expanding the effective receptive field K_{eff} without increasing the parameter count. These choices determine the output shape according to the standard formula:

$$K_{eff} = d\cdot(K-1) + 1$$

$$H_{out} = floor\left(\frac{(H + 2P - K_{eff})}{S}\right) + 1,$$

$$W_{out} = floor\left(\frac{(W + 2P - K_{eff})}{S}\right) + 1$$

The activation function introduces nonlinearity. The Rectified Linear Unit (ReLU) is ubiquitous due to its simplicity and favorable gradient propagation; variants such as Leaky-

ReLU include a slope parameter α for negative inputs to mitigate the dying-ReLU issue:

$$\sigma(z) = max(0, z)$$

Pooling layers summarize local neighborhoods and control spatial dimensionality. With a window (e.g., 2×2) and a pooling stride (often equal to the window size), max pooling returns the largest activation within the region, whereas average pooling returns the mean value:

$$y(i,j) = max_{\{(u,v)\in R\}}\, x(i+u, j+v)$$

$$y(i,j) = \left(\frac{1}{|R|}\right) \Sigma_{\{(u,v)\in R\}}\; x(i+u, j+v)$$

Normalization stabilizes and accelerates training. In Batch Normalization (BatchNorm), each activation z is standardized using the mini-batch mean μ and variance σ^2, then rescaled and shifted by trainable parameters γ and β. A small ε prevents division by zero. Running estimates of μ and σ^2 are maintained for inference:

$$\hat{z} = \frac{(z - \mu)}{\sqrt{\sigma^2 + \varepsilon}}$$

$$y = \gamma \cdot \hat{z} + \beta$$

For classification, the head typically applies global average pooling (or flattening) followed by a fully connected layer that produces class scores a_k. Probabilities are obtained via the softmax function, and the network is trained by minimizing the cross-entropy loss. Label smoothing with parameter ε can be employed to regularize targets:

$$\hat{p}_k = e^{a_k} \Sigma_{\{\ell\}} e^{a_\ell}$$

$$L = -\, \Sigma_k\, y_k \cdot log(\hat{p}_k)$$

Training dynamics are governed by optimization hyperparameters. The learning rate η controls the step size; stochastic gradient descent (SGD) with momentum uses a momentum parameter μ, while Adam employs β_1, β_2, and ε hyperparameters. Batch size mediates the trade-off between gradient variance and memory usage; the number of epochs specifies how many passes over the dataset are performed. Regularization techniques include weight decay (λ) to penalize large weights and dropout with probability p to reduce overfitting.

During backpropagation, gradients flow from the loss to the filters. Parameters are updated by gradient descent. For a convolutional filter W_k, the gradient equals the correlation between the input and the error map δ_k; the gradient with respect to the input equals the convolution of the spatially flipped kernel with δ_k (summed over output channels). In practice, gradient clipping and learning-rate schedules (e.g., step decay, cosine annealing, warmup) are additional hyperparameters that often improve stability:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta L$$

$$\frac{\partial L}{\partial W_{\{k,c\}}} = x_c \star \delta_k$$

$$\frac{\partial L}{\partial x_c} = \Sigma_k \widetilde{W}_{\{k,c\}} * \delta_k$$

As a practical starting point for many vision tasks: 3×3 kernels, stride 1, padding 1, ReLU activations, BatchNorm, occasional 2×2 max pooling, and an Adam optimizer with learning rate in the 10^{-4}–10^{-3} range, weight decay between 10^{-4} and 10^{-2}, and dropout p ≈ 0.2–0.5 are reasonable defaults. These should then be tuned to the data resolution, the desired capacity, and observed overfitting.

The ML models were compiled using the Adam optimizer with a learning rate (η) of 0.0001 and the Binary Cross-Entropy loss function. Accuracy was monitored as the performance metric. The model was configured to train for 100 epochs. Finally, the trained model was serialized and saved in the HDF5 and PTH format for subsequent evaluation.

In CV performance evaluation (Stage 4), as a proof-of-concept within the context of this study, an evaluation of the model's generalization capability for classifying the "at-risk" state was conducted. This evaluation utilized a combined dataset created by merging two public data sources: (i) 361 images from the study by Carraro et al. [34]; (ii) 941 images from the study by Maldonado-Bascon et al. [35]. In both datasets, the selected images explicitly depict a person falls and non-falls.

In Carraro et al. [34], the dataset is organized into two parts. The static portion comprises the folders “raw” (360 RGB frames and point clouds with calibrations), “segmented_fallen_people” (manually segmented point clouds of fallen individuals), “training_with_cad_room_and_nyudv2” (random positive samples, 70%, including 24 clouds from Lab A and 31 from NYU Depth v2), and “test_with_lab_room” (the remaining 30%, with 32 clouds from Lab B). The dynamic portion includes “training” (four ROS bags totaling 15,932 RGB-D frames from four patrols in Lab A), “test” (four ROS bags with 9,391 frames from Lab B—an apartment-like environment with heavier clutter and a sofa), and “maps” (2D maps of both environments and ground-truth centroid positions). This organization aligns with the ROS- and Kinect v2-based mobile platform described in the paper.

Maldonado-Bascon et al. [35] introduced the Fallen

Person Dataset (FPDS), captured by a single camera mounted 76 cm above the floor on a patrol robot. The dataset covers eight environments with variations in illumination and reflections (defining eight splits) and contains 2,064 images annotated with bounding boxes and labels y=1 (fall) or y=-1 (non-fall), plus checkerboard images for calibration. The evaluation protocol promotes cross-scenario generalization: training uses splits 1–3 (1,084 images; 681 falls and 432 non-falls) and testing uses splits 4–8 (973 images; 391 falls and 830 non-falls). The paper also reports the per-split distribution, 1,072 falls and 1,262 non-falls in total and publicly releases the dataset for reuse.

The performance evaluation was conducted using standard classification metrics derived from the confusion matrix [36], which categorizes predictions into true positives (TP), representing at-risk events correctly identified as such, and false positives (FP), corresponding to safe events erroneously classified as at-risk (Type I error). From these classifications, the Precision metric was calculated for each in silico analysis as the ratio of TP/(TP+FP), in accordance with established methodologies.

To systematically investigate our RQs, we implemented a full 2x2x2 factorial experimental design, resulting in eight distinct model architectures (MA). This design enables us to analyze the isolated and combined effects of three primary factors, as detailed in Table 1. The RQs are addressed as follows:

- RQ1: Effect of the Development Framework: We compare the results of models implemented in TensorFlow/Keras (MA 1, 2, 5, 6) with those implemented in PyTorch (MA 3, 4, 7, 8).
- RQ2: Influence of Preprocessing: We analyze the impact of the preprocessing pipeline by comparing models with this technique applied (MA 2, 4, 6, 8) against their counterparts without it (MA 1, 3, 5, 7).
- RQ3 & RQ4: Feasibility of Training with Synthetic Data: We evaluate the suitability of using synthetic data under two conditions:
  - For models employing Transfer Learning (RQ3), by analyzing the performance of MA 1 through 4.
  - For models trained from scratch (RQ4), by investigating the performance of MA 5 through 8.

Table 1. Experimental Design of the Model Architectures

MA	TS	Pp	Framework
1	TL	No	TensorFlow/Keras
2	TL	Yes	TensorFlow/Keras
3	TL	No	PyTorch
4	TL	Yes	PyTorch
5	FS	No	TensorFlow/Keras
6	FS	Yes	TensorFlow/Keras
7	FS	No	PyTorch
8	FS	Yes	PyTorch

*MA: Model Architecture, TS: Training Strategy, Pp: Preprocessing, TL: Transfer Learning, FS: From Scratch

4. Results

The entire 100-epoch training process for all eight architectures was completed in approximately 120 minutes.

All eight evaluated MAs demonstrated exceptional training performance, achieving rapid convergence and maximum validation accuracy. As illustrated by the learning curves (Figures 7-10), the validation accuracy for all versions reached a stable plateau of 100%.

Notably, the transfer learning-based models achieved this performance by approximately the 8th epoch, whereas the from-scratch architectures converged even more rapidly, before the 1st epoch.

Correspondingly, the validation loss for the transfer learning model architectures (MA) exhibited a sharp, monotonic decrease. Initial loss values were approximately 0.8 (MA1), 0.65 (MA2), and 0.6 (MA3 and MA4), with all models converging to a final value near 0.01.

In contrast, the from-scratch architectures displayed a negligible validation loss, which remained near-zero from the earliest epochs of training.

Table 2 presents the model's performance metrics on the benchmark dataset, focusing on the classification of the 'at-risk' class. The best-performing model was MA3, which combined the PyTorch framework with Transfer Learning, achieving 95.8% accuracy without the use of specific preprocessing. In contrast, the models trained from scratch with TensorFlow (MA5 and MA6) failed completely to learn, registering 0% accuracy.

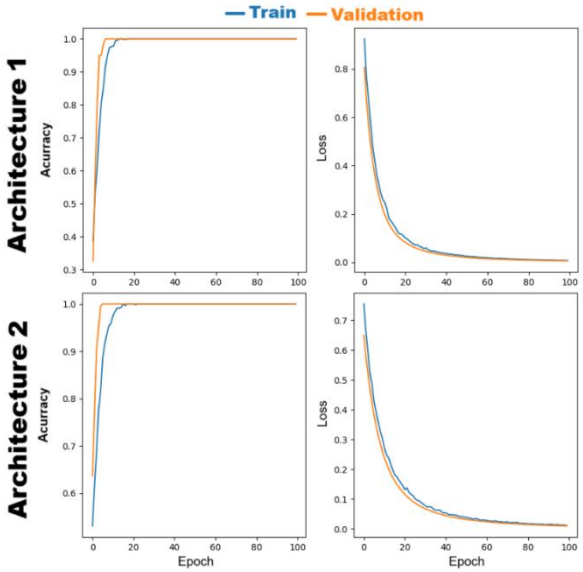

Figure 7. Training and Validation Performance of the model architectures 1 and 2 over 100 Epochs

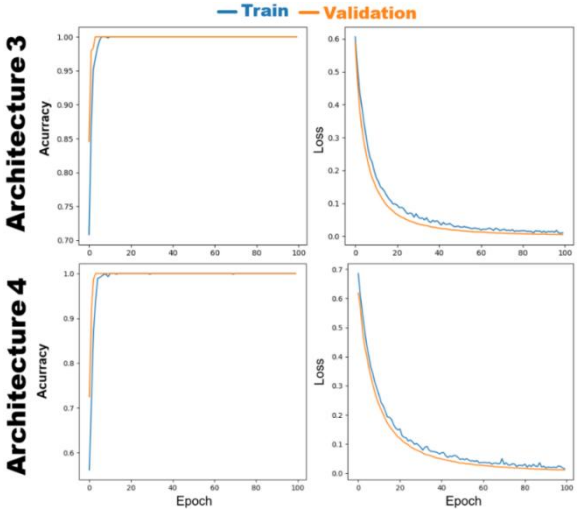

Figure 8. Training and Validation Performance of the model architectures 3 and 4 over 100 Epochs

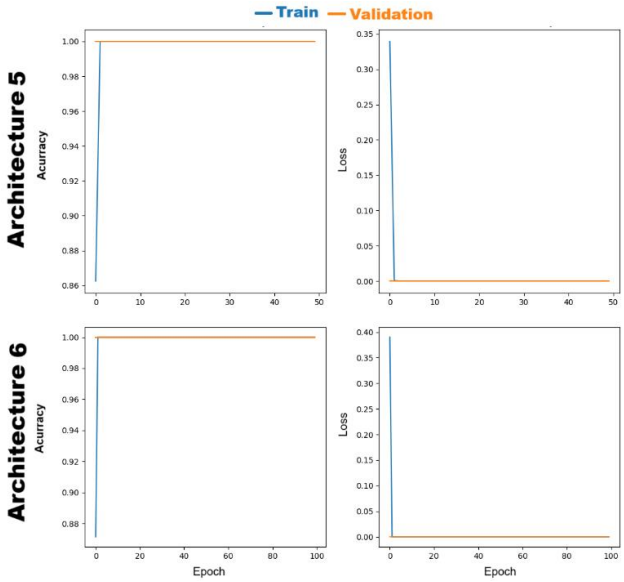

Figure 9. Training and Validation Performance of the model architectures 5 and 6 over 100 Epochs

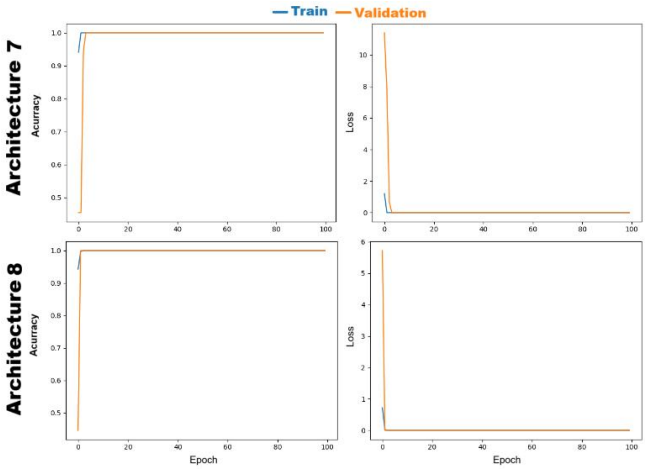

Figure 10. Training and Validation Performance of the model architectures 7 and 8 over 100 Epochs

Table 2. Comparative Performance of Models in 'At-Risk' State Detection on the Benchmark Dataset

MA	TP	FP	Precision (%)
1	753	529	58.7
2	956	346	73.4
3	1248	54	95.8
4	114	1188	8.75
5	0	1302	0
6	0	1302	0
7	1051	251	80.7
8	791	511	60.7

*MA: Model Architecture, TP: True-positive, FP: False-positive

5. Discussion

In addressing RQ1, the choice of deep-learning framework emerged as a decisive factor. Under identical experimental conditions, the PyTorch transfer-learning model (MA3) achieved 95.8% accuracy, substantially outperforming its TensorFlow counterpart (MA1) at 58.7%. The gap widened for from-scratch training: the PyTorch model (MA7) reached 80.7% accuracy, whereas the TensorFlow models (MA5 and MA6) failed to learn (0%).

This discrepancy does not imply task-agnostic superiority of one framework. Rather, multiple low-level choices likely interacted with our small, synthetic dataset. Differences in default initialization (Kaiming/He in PyTorch vs. Glorot/Xavier in Keras/TensorFlow) affect early signal propagation with ReLU-like activations. Optimizer semantics also diverge: distinct Adam ε defaults (e.g., 10^{-8} in PyTorch vs. 10^{-7} in Keras), bias-correction details, and the use of decoupled

weight decay (AdamW) versus coupled L2 regularization produce measurably different update rules. Batch Normalization is parameterized with non-equivalent conventions, PyTorch typically uses "eps≈1e−5, momentum=0.1" (update coefficient), while Keras often uses "epsilon≈10^{-3}, momentum=0.99" (EMA decay), yielding different running-stat dynamics under small batch sizes.

Additional sources of divergence arise from data handling and loss semantics. Pretrained pipelines expect distinct input normalizations (e.g., torchvision's ImageNet mean–std vs. architecture-specific "preprocess_input" in Keras); even minor mismatches in scaling or channel order degrade transfer. Loss–activation coupling must also be aligned: "CrossEntropyLoss" in PyTorch consumes logits, whereas Keras's categorical cross-entropy requires "from_logits=True" to avoid double softmax or probability/logit mismatches; label encoding (sparse vs. one-hot) and loss reductions (mean vs. sum) further rescale gradients. Stochasticity in the input pipeline (e.g., "DataLoader" shuffling vs. "tf.data" buffer size and "drop_remainder") alters batch statistics seen by BatchNorm and can bias optimization.

Training dynamics may also differ due to schedule granularity and numerics. Learning-rate schedules are typically applied per epoch in PyTorch and per step in Keras; mis-specified decay steps or warm-up policies can induce premature LR decay and underfitting. Mixed-precision implementations use distinct loss-scaling heuristics, changing effective step sizes and stability with small batches. Finally, execution models and tensor layouts (NCHW vs. NHWC/"channels_last") lead to different cuDNN kernel selections and rounding noise; pretrained weight provenance and head initialization recipes may not be identical across ecosystems; and train/eval mode semantics (e.g., "model.train()" vs. "layer.trainable" + "compile()") together with incomplete seed/determinism control can push runs toward different optimization basins.

Taken together, these factors, initialization, optimizer and weight-decay semantics, normalization hyperparameters, input preprocessing and labeling, pipeline stochasticity, schedule granularity, mixed-precision scaling, execution/layout differences, pretrained weight recipes, training-mode handling, and determinism, can jointly produce the observed gap. To isolate causes, future ablations should align these elements across frameworks under fixed seeds and deterministic backends, standardize preprocessing, enforce consistent loss/logit settings and label encodings, match BatchNorm and Adam hyperparameters (including ε and decay formulation), mirror shuffling/batching policies, and synchronize the learning-rate schedule and warm-up frequency. Such controls will clarify whether the observed PyTorch advantage reflects genuine algorithmic benefits or the accumulation of small implementation discrepancies.

In addressing RQ2, the effect of the data preprocessing pipeline was ambiguous and highly context-dependent. In the TensorFlow ecosystem, preprocessing was beneficial for the Transfer Learning model, increasing accuracy by nearly 15 percentage points (from 58.7% in MA1 to 73.4% in MA2). However, in PyTorch, the same preprocessing pipeline had a severely negative effect, causing a catastrophic drop in the Transfer Learning model's accuracy from 95.8% (MA3) to just 8.75% (MA4). A similar negative trend, albeit less drastic, was also observed in the model trained from scratch (MA7 vs. MA8). This indicates that preprocessing is not a universal improvement; its effectiveness can be nullified or even reversed by complex interactions with the framework and training strategy, potentially interfering with the data distribution the model expects.

In addressing RQ3 & RQ4,the results confirm the viability of using synthetic data to train Computer Vision models, albeit with important considerations.

In response to RQ3, a pre-trained model can indeed achieve adequate (and even excellent) performance with synthetic images. The overwhelming success of MA3 (95.8%) demonstrates that features learned by pre-trained models on real-world data can be effectively transferred to the synthetic domain, validating the Transfer Learning approach as the most promising strategy.

Regarding RQ4, the results show that a model trained from scratch can also achieve adequate performance, but this approach is more sensitive to the choice of tools. The success of MA7 (PyTorch) with 80.7% accuracy proves that it is possible to train a competitive model exclusively with synthetic data. However, the complete failure of the TensorFlow models (MA5 and MA6) serves as a strong cautionary note that this approach is less robust and more dependent on the experimental setup.

In line with the literature on computer vision for smart hospital-bed settings and sim-to-real transfer, our findings suggest that the "reality gap" is not only visual/physical but also implementation-level: framework choices, normalization, and the semantics of loss functions can act as a form of domain shift, analogous to the effects of textures, lighting, and sensors in simulations [8,19,20]. This perspective helps explain why domain randomization and photorealistic environments described in the state of the art mitigate pipeline idiosyncrasies, by expanding synthetic variability and control over scene

parameters, they reduce reliance on specific conventions for initialization, BatchNorm, optimizers, and preprocessing [11,17,18,21–25]. In parallel, the strong performance obtained by transferring pretrained weights from real data to synthetic images aligns with the hybrid strategy seen in benchmarks and simulators (e.g., gains from combining real + synthetic in SYNTHIA and interactive ecosystems such as CARLA), which anchor representations in real-world statistics and thereby narrow the domain gap [24,25]. By contrast, the greater brittleness of training from scratch underscores the need for explicit adaptation mechanisms (at the pixel or representation level) to stabilize optimization when only synthetic data are available, as demonstrated by adversarial approaches and canonical mappings (GraspGAN, SimGAN, RCAN) [27,28]. For bedside clinical applications, where safety and reproducibility are critical, these parallels point to two pragmatic fronts: (i) rigorously standardize preprocessing, update policies, and execution modes across ecosystems to avoid tool-induced artificial gaps; and (ii) combine synthetic variability (randomization/realism) with adaptation and pretraining, making performance less contingent on the framework and more faithful to domain requirements [8,9,17,21–28].

It is essential to acknowledge that the results presented here, while promising, arise from a preliminary investigation into using synthetic images to train CV models for detecting at-risk situations in hospital beds. The evaluation relied solely on precision, which is inadequate for medical safety applications: a classifier may achieve high precision yet exhibit dangerously low recall, failing to flag most critical events. Future work will broaden the assessment to include recall (sensitivity), specificity, F1-score, balanced accuracy, and area under the precision–recall curve (AUC-PR), along with calibration analyses (e.g., reliability diagrams), subgroup-stratified confusion matrices, and error-cost studies emphasizing false negatives and detection latency.

From a data standpoint, the synthetic dataset serves as a proof of concept but is limited to two binary states (“sleeping” and “fallen”), which does not reflect the complexity of real patient behavior (e.g., agitation, attempts to stand, sitting at the bed edge, visual occlusions, presence of a caregiver, and variations in lighting and equipment). This simplification increases the risk of a sim-to-real gap and constrains generalizability. We will address these limitations by (i) expanding the dataset with a larger and more diverse mix of synthetic and real images across multiple institutions and settings; (ii) modeling multi-label, temporally evolving states in video sequences; (iii) employing domain adaptation and style randomization; and (iv) conducting prospective external validation with uncertainty estimates (confidence intervals), thorough error analysis, and reproducibility safeguards (fixed random seeds, open code, and standardized protocols).

6. Conclusions

This study demonstrated the viability of using synthetic images, generated via 3D modeling, to train computer vision models for detecting at-risk patient states. The findings indicate that a Transfer Learning approach (RQ3) is particularly effective in this domain, outperforming models trained from scratch (RQ4). Furthermore, the investigation revealed an unexpected sensitivity of model performance to the choice of development framework (RQ1) and the application of preprocessing pipelines (RQ2), highlighting the critical importance of the experimental environment's configuration.

The proposed methodology establishes a systematic workflow for data generation in scenarios where real-world data collection is impractical, hazardous, or ethically complex. The controlled nature of 3D modeling allows this approach to be potentially adapted for other clinical monitoring problems, such as posture analysis or the detection of patient-equipment interactions. Consequently, this work not only validates a solution for the specific problem but also contributes a methodological paradigm that can accelerate the development and validation of future AI-based assistive technologies in biomedical engineering.

Conflicts of Interest: The author declares no conflicts of interest.

Supplementary Material: The methodology for generating synthetic images using Blender program was presented in detail during a scientific conference workshop; the presentation is available here: https://doi.org/10.13140/RG.2.2.25936.52482.

References

[1] Cardoso, R.; Parola, V.; Neves, H.; Bernardes, R.A.; Duque, F.M.; Mendes, C.A.; et al. Physical Rehabilitation Programs for Bedridden Patients with Prolonged Immobility: A Scoping Review. *Int. J. Environ. Res. Public Health* **2022**, *19*, 6420.

[2] World Health Organization. Rehabilitation. Available online: https://www.who.int/health-topics/rehabilitation (accessed on 9 February 2025).

[3] Paul, S.; Riffat, M.; Yasir, A.; Mahim, M. N.; Sharnali, B. Y.; Naheen, I. T.; et al. Industry 4.0 applications for medical/healthcare services. *J. Sens. Actuator Netw.* **2021**,

10, 43.

[4] Santos, B.; Martins, D.; Leao, T.; Bock, E. Supervisory Control System for Hospital Rehabilitation Beds. In Proceedings of the 2021 9th International Conference on Control, Mechatronics and Automation (ICCMA), Belval, Luxembourg, 11-14 November 2021; pp. 130–134.

[5] Lu, Y.; Zhang, Y.; Tian, Y. Research Progress of Intelligent Management Model Application in Hospital-at-Home Care. *J. Phys.: Conf. Ser.* **2023**, *2425*(1), 012041.

[6] Barreto, R.L.P.; Simoni, R.; Martins, D. An initial assessment of mechanisms for the development of new hospital beds. In *Multibody Mechatronic Systems*: Proceedings of the MUSME Conference held in Florianópolis, Brazil, October 24–28, 2017; Springer International Publishing: Cham, Switzerland, 2018; Vol. 6, pp. 485–494.

[7] Campos, A.; Cortés, E.; Martins, D.; Ferre, M.; Contreras, A. Development of a flexible rehabilitation system for bedridden patients. *J. Braz. Soc. Mech. Sci. Eng.* **2021**, *43*, 361.

[8] Santos, B.J.; Leão, T.F.; Palma, R.; Martins, D.; Bock, E.G.P. A Supervisory Control System for Flexible Hospital Rehabilitation Beds Based on Computer Vision. In Proceedings of the 2023 International Conference on Robotics, Control and Vision Engineering, Tokyo, Japan, 21-23 July 2023; pp. 48–53.

[9] Szeliski, R. Computer Vision: Algorithms and Applications; Springer Nature, 2022.

[10] Khan, A.A.; Laghari, A.A.; Awan, S.A. Machine Learning in Computer Vision: A Review. *EAI Endorsed Trans. Scalable Inf. Syst.* **2021**, *8*, e4.

[11] Man, K.; Chahl, J. A Review of Synthetic Image Data and Its Use in Computer Vision. *J. Imaging* **2022**, *8*, 310.

[12] Leilabadi, S.H.; Schmidt, S. In-Depth Analysis of Autonomous Vehicle Collisions in California. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27-30 October 2019; pp. 889–893.

[13] Major, L.; Shah, J. What to Expect When You're Expecting Robots: The Future of Human-Robot Collaboration; Hachette UK, 2020.

[14] Islam, M.N.; Aadeeb, M.S.; Munna, M.M.H.; Rahman, M.R. A Deep Learning Based Multimodal Interaction System for Bed Ridden and Immobile Hospital Admitted Patients: Design, Development and Evaluation. *BMC Health Serv. Res.* **2022**, *22*, 803.

[15] Beery, S.; Liu, Y.; Morris, D.; Piavis, J.; Kapoor, A.; Meister, M.; et al. Synthetic Examples Improve Generalization for Rare Classes. In Proceedings of the 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), Snowmass, CO, USA, 1-5 March 2020; pp. 863–873.

[16] Gao, C.; Killeen, B.D.; Hu, Y.; Grupp, R.B.; Taylor, R.H.; Armand, M.; Unberath, M. Synthetic data accelerates the development of generalizable learning-based algorithms for X-ray image analysis. *Nat. Mach. Intell.* **2023**, *5*, 294–308.

[17] Zou, Z.; Chen, K.; Shi, Z.; Guo, Y.; Ye, J. Object Detection in 20 Years: A Survey. *Proc. IEEE* **2023**, *111*, 257–276.

[18] Blender. Blender Documentation. Available online: https://docs.blender.org/manual/en/latest/ (accessed on 4 July 2025).

[19] Mahler, J.; Liang, J.; Niyaz, S.; Laskey, M.; Doan, R.; Liu, X.; et al. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv* **2017**, arXiv:1703.09312.

[20] Tai, L.; Paolo, G.; Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24-28 September 2017; pp. 31–36.

[21] Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24-28 September 2017; pp. 23–30.

[22] Sadeghi, F.; Levine, S. Cad2rl: Real single-image flight without a single real image. *arXiv* **2016**, arXiv:1611.04201.

[23] Akkaya, I.; Andrychowicz, M.; Chociej, M.; Litwin, M.; McGrew, B.; Petron, A.; et al. Solving Rubik's cube with a robot hand. *arXiv* **2019**, arXiv:1910.07113.

[24] Ros, G.; Sellart, L.; Materzynska, J.; Vazquez, D.; Lopez, A.M. The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27-30 June 2016; pp. 3234–3243.

[25] Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the 1st Annual Conference on Robot Learning, Mountain View, CA, USA, 13-15 November

2017; pp. 1–16.

[26] Bousmalis, K.; Irpan, A.; Wohlhart, P.; Bai, Y.; Kelcey, M.; Kalakrishnan, M.; et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21-25 May 2018; pp. 4243–4250.

[27] Shrivastava, A.; Pfister, T.; Tuzel, O.; Susskind, J.; Wang, W.; Webb, R. Learning from simulated and unsupervised images through adversarial training. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21-26 July 2017; pp. 2107–2116.

[28] James, S.; Wohlhart, P.; Kalakrishnan, M.; Kalashnikov, D.; Irpan, A.; Ibarz, J.; et al. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15-20 June 2019; pp. 12627–12637.

[29] OpenCV. OpenCV Documentation 4.0.1. Available online: https://docs.opencv.org/4.x/ (accessed on 4 July 2025).

[30] TensorFlow. TensorFlow Documentation. Available online: https://www.tensorflow.org/api_docs (accessed on 4 July 2025).

[31] Keras. Keras Documentation. Available online: https://keras.io/guides/ (accessed on 4 July 2025).

[32] PyTorch. PyTorch Documentation. Available online: https://pytorch.org/docs/stable/index.html (accessed on 4 July 2025).

[33] Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18-23 June 2018; pp. 4510–4520.

[34] Carraro, M.; Antonello, M.; Tonin, L.; Menegatti, E. An open source robotic platform for ambient assisted living. In *AIRO@AIIA*; 2015; pp. 3–18.

[35] Maldonado-Bascon, S.; Iglesias-Iglesias, C.; Martín-Martín, P.; Lafuente-Arroyo, S. Fallen people detection capabilities using assistive robot. *Electronics* **2019**, *8*, 915.

[36] Ferrão, I.G.; de Oliveira, A.; Marçal, V.; Allão, D.; Espes, D.; Dezan, C.; et al. Supervised models for detecting GPS attacks and faults in UAVs: a comparative analysis. In Proceedings of the 2024 Latin American Robotics Symposium (LARS), November 2024; pp. 1–6.